PROGRAMMING LAW IN LOGIC

Allen Hustler

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-82-13

May 1982

PROGRAMMING LAW IN LOGIC

Allen Hustler

Department of Computer Science
University of Waterloo

ABSTRACT

This paper briefly reviews recent work on automating the legal research process; investigates the relationship between law and logic, and the work done by legal academics in this area; and describes a prototype interactive logic-based legal consultant written in the PROLOG language.

March, 1982

PROGRAMMING LAW IN LOGIC

Allen Hustler

Department of Computer Science
University of Waterloo

1. INTRODUCTION

This paper describes a representation of a portion of the law of battery in the computer language PROLOG. The representation is capable of performing basic legal analysis on simple sets of facts. Through a brief look at some of the work which has been done on the application of computers to law, and on the use of logic as a programming language, a case is made for the naturalness of PROLOG as a language for conducting artificial intelligence research in the field of law.

2. COMPUTERS AND LAW

Attempts to apply the power of computers to the task of legal research began more than twenty years ago. The amount of stored information required for research in even very limited areas of the law is so large, however, that significant practical results awaited the development of fast, random-access secondary storage devices. By 1973, Mead Data Central, Inc. had begun commercial operation of its LEXIS system in the United States [Rubin], and the QUIC/LAW project at Queen's University in Canada had conducted field trials [Lawford]. Both of these systems, along with the WESTLAW service in the U.S. and similar systems in Europe, stored the texts of thousands of statutes and appelate court decisions, either in whole or as summaries, and then attempted to retrieve information relevant to a lawyer's needs by searching the entire database for specific words or sets of words which the lawyer hoped all relevant documents would contain (and irrelevant documents would not contain).

The premises underlying this approach to legal research have been enumerated by Fabien (1979), a participant in a project of this type:

> "-it is preferable for the lawyer to do his research on the original statutory and case-law texts, not amended or altered by an intermediary whose intervention might insert an element of distortion between the lawyer and the sources of law;

> -a text in a data base, containing words identical to those used by the lawyer in formulating his question, is relevant and should be reported;
> -a lawyer generally has a thorough knowledge of the legislative and judicial language, and it is thus easier for him to identify the words used by the law-makers and judges, in order to produce those texts that are relevant to solving his problem. On the other hand, it is more difficult for him to find the categories into which an unknown intermediary has filed the same information in a classical research instrument." [Fabien, p.15]

It could be remarked that these premises don't fit very well with the actual practice of most lawyers, who rely almost entirely on headnotes, summaries, and indexes when researching a new area of law. For whatever reasons, the success of text-based retrieval systems over the past eight years has been mixed at best. Lawyers have turned out to be unwilling to master the bag of tricks necessary to construct the boolean logic queries needed to carry out their searches efficiently. More importantly, it has become well-recognized that even experienced database searchers too often fail to recover cases that any law student would easily find on his own, yet frequently retrieve large numbers of irrelevant cases. This seems to be an intrinsic failing of retrieval systems based only on words rather than on the information contained in the words.

Lower level problems have also hindered the effectiveness of some of these systems. In Canada, for example, QUIC/LAW has never been able to accumulate a large enough body of case law in its database because its sponsors were unable to negotiate agreements with several of the law publishing companies that own the copyright to published Canadian court decisions and summaries [Fabien, p.10].

Computer scientists have made a number of efforts to design systems that would overcome or avoid the problems of the commercial databases. Their goal has been to find a way to store and retrieve the knowledge contained in the primary legal materials, rather than just their words.

The TAXMAN project [McCarty, 1977] is an ongoing attempt to model corporate tax law. The system is implemented in a frame-based language called AIMDS, but has apparently not yet reached the stage of practical large-scale applications.

Hafner (1978) designed and implemented a system for the knowledge-based retrieval of legal documents in the area of negotiable instruments law. She developed a simple semantic network to represent knowledge about legal concepts and

relationships, and stored an analysis of each case and each section of a statute in terms of this semantic network. Phrasing queries in terms of her model enables the system to go some way towards matching the meaning of a query with the semantic content of the documents in the database.

The LEGOL project at the London School of Economics [Stamper, 1980] originated as an attempt to develop improved tools for computer systems specification. The participants soon realized, however, that the task of expressing in general yet sufficiently clear and unambiguous language how a procedure ought to be performed is exactly the problem faced by legal draftsmen. They therefore decided to use legislation as their experimental material, and set out to develop a language for writing statutes that would enable the law to be directly interpreted by a computer. This project has achieved some success in the area of routine administrative legislation.

Meldman (1975, 1977) designed a prototype system for computer-aided legal analysis in the area of the law of assault and battery. His representation is based to some extent on semantic networks, and uses a relation-based formalism that is difficult to penetrate. The formalism also suffers from the very great disadvantage that it was never fully implemented. Nonetheless, Meldman's work has provided a valuable starting point for the work described in the present paper.

Brief excerpts from published reports on each of the above projects can be found in McCarty (1981).

In spite of the rather meagre results achieved so far by both commercial and research efforts, the interest of the legal community in using computers in their practices has increased dramatically [see e.g. Bigelow, 1981]. Many law offices now rely on word processors for document preparation, and a project funded by the American Bar Foundation has attempted to apply computers more directly to the drafting of such common legal documents as contracts and wills.

This endeavour, described in Sprowl (1979), involved the development of a special "ABF" language which takes as its input a "normalized" statute or regulation or legal form, and produces a series of questions which the computer will ask a lawyer or legal technician who is attempting to draft a document that complies with the given law's requirements. The answers to these questions are then used by the computer to construct a finished document which meets those requirements. The following example illustrates many of the features of this approach:

> "Suppose that a probate library includes a form petition for probate that contains the following

passage:

[IF the estate may be probated without court appearance INSERT]
Petitioner requests permission to ...
...
[ENDIF]

Assume that the relevant statute permits probate without court appearance only when all the beneficiaries and children are at least 21 years old, are of sound mind, and have consented to an out-of-court settlement; the named executor is an in-state resident; and the decedent has in the will waived security on the executor's bond. A normalized version of this statute, drafted in the language ABF, might appear as follows:

IFF
the decedent's children and the named beneficiaries
ARE of sound mind and over 20 years old
AND
the named executor IS a resident of the state of Illinois
AND the decedent HAS waived security on the executor's bond in the will
AND the decedent's children and the named beneficiaries HAVE consented to an out-of-court settlement
THEN the estate may be probated without court appearance.

If the above statute is stored in the probate form library along with the petition, and if the library name of this statute is "the estate may be probated without court appearance", the processor will not ask whether the estate may be probated without court appearance. Instead, the processor will ask one or more of the following questions, all of which are derived from the language of the normalized statute:

Are the decedent's children and the named beneficiaries of sound mind and over 20 years old?
Is the named executor a resident of the state of Illinois?
Has the decedent waived security on the executor's bond in the will?
Have the decedent's children and the named beneficiaries consented to an out-of-court settlement?

After the attorney or legal assistant answers these questions, the processor automatically

> alters the form document being drafted in accordance with whether the estate is to be probated with or without court appearance." [Sprowl, p.45]

This technique, even though its achievements seem unremarkable in terms of the ultimate goal of an intelligent legal information retrieval system, nevertheless incorporates several important ideas which others may have missed.

Significantly, this is a system that will work properly from the very first day it is installed in a law office. Its initial use may be limited to some small task like drafting certain kinds of contracts or producing the forms necessary to incorporate a small business, but it will perform these tasks quickly and properly. As the number of forms and statutes available in the system gradually increases, so does the system's ability to take on other jobs. The lawyer is never faced, as he may be with text-based systems, with a computer program that promises to assist him with a broad spectrum of legal research which it can never satisfactorily deliver.

The ABF approach is important, as well, because it directs the performance of legal research towards the purpose of that research. For many of the systems designed to date, and certainly for the text-based systems, the purpose of a request for legal information is both unknown and irrelevant. Yet a lawyer will conduct his own research efforts differently depending on the ultimate use he intends to make of the research. The representation of legal knowledge in the ABF system is designed specifically for the purpose of preparing legal forms, where the law has a manageable number of possible twists and turns (as evidenced by the fact that such forms are almost invariably completed by secretaries and clerks, not by lawyers).

Most importantly, the ABF approach recognizes that the meaning of law can be stored in a computer only if the language of law can be simplified and regularized. As long as law is written in language that is impossible for an intelligent human to understand, the chances of a machine-operated procedure ever being able to understand it are practically zero. Any serious effort to represent legal rules in a computer must, therefore, be combined with a serious effort to simplify the language of the legal rules. Such an effort would certainly be worthwhile in its own right: the benefit to society of a successful method of making law more easily understandable for humans would likely outweigh the benefits of successful computerization of legal research.

One approach to the simplification of legal language,

and ultimately to its representation in a computer system, is the rigorous application of the rules of logic.

3. LAW AND LOGIC

In the third century B.C., the Stoic Chrysippus defined law as "right reason, proceeding from supreme Zeus, commanding what should be done and forbidding what should not be done" [Radin]. "Right reason", a translation of "orthos logos", is logic. From that early beginning, by way of Cicero in Rome and Coke and Blackstone in England, the concepts of logic and law have been inextricably intertwined in our culture.

The development of many of the world's legal systems has been described in terms of the application of logic to divinely revealed basic principles. Pound's major treatise on jurisprudence, for instance, includes an example of the development of Islamic law in exactly this way [Pound, p.32].

The history of the common law in England and America can in fact be seen, in one view, as the working out over centuries of the inherent tension between the concept of law as pure logic and the parallel need for the law to exhibit flexibility and humanity at the possible expense of reason and impartiality. As long as law is the divinely inspired word of a king or feudal lord, the universal desire in society is for a set of strict rules that will be applied impartially to all. And when, centuries later, the law has become a body of strictly applied rules which now form a nightmarish trap for anyone unlucky enough to be in need of justice, the great hope is for a new court of equity that will temper cold reason with individual fair treatment. The search for the proper balance between the requirements of logic and the need for fairness continues, in this view, to be at the heart of legal argument up to the present day.

Within the past sixty-five years, logic having long ago been put in its proper place as merely one of the components of legal decision-making, the idea began to surface that perhaps the modern, mathematical science of logic as formulated by Leibniz, Boole, De Morgan and others could be used as a tool in drafting and understanding legal rules. Between 1916 and 1955, several isolated articles on the possible uses of mathematical logic in legal drafting appeared in the law journals (e.g. Cohen(1916); Keyser(1929); Berkeley(1937); Oppenheim(1944); and Stoljar(1953)).

About twenty-five years ago this activity grew more intense, with the apprearance of influential papers by Tamello(1956) and Allen(1957). The thrust of these articles was that the intelligent use of the rigorously defined connectives of symbolic logic ("and", "or", "not", "if ...

then") would simplify the language of legislation and legal documents, remove ambiguities, and ensure that the intended meaning of a rule was conveyed to whoever had to understand or apply it. Allen, paraphrasing Tamello, summarized their perception of the importance of symbolic logic to law in the following observations. Note the effort he makes to allay lawyers' fears that logic is once again being promoted over fairness and experience:

> "(1)Logic in general can be used as a universal form of reference, an all-embracing theory of scientific research to coordinate all of the different disciplines.
>
> (2)Symbolic logic is more exact and more comprehensive than traditional logic.
>
> (3)Even though traditional logic is more easily communicated in our present state of learning because of its greater familiarity, nevertheless symbolic logic is capable of more effective rational penetration.
>
> (4)The increasing complexity of legal analysis, just as in any other disipline, produces a greater need for the simplification and precision of symbolic logic; and the tendency away from traditional logic is already visible in philosophy, natural sciences and theoretical economy.
>
> (5)Symbolic logic is not a new conception of law; it is not a source of experience, but only an intellectual tool to master human experience.
>
> (6)Symbolic logic employed in legal thinking will not deny the role of intuition; there is room in legal thinking for both analytical and intuitive approaches.
>
> (7)Exponents of symbolic juristic logic are not seeking to promote legal dogmatism by means of a 'super logic'; on the contrary, they suggest that it provides a logical means of penetration into the sociological substratum of law, and excels traditional logic in doing so." [Allen(1957), p.878]

Allen continued his work in this area, and his version of "normalized" legal rules became the one used twenty years later by the ABF computer program. In a 1963 paper, Allen and Caldwell anticipated by about ten years the development of logic as a programming language, and recognized its importance to automated legal analysis. A brief excerpt from

that paper illustrates some of these points:

> "[Consider, f]or example, the two sentences:
> S1 All persons who are lawyers draft some wills.
> and
> S2 If a person is a lawyer, then he drafts some wills.
> [B]oth express the same idea. If S1 were used in a written document, we could substitute S2 for S1 without changing the ideas expressed by the document. This is no startling revelation, but draftsmen who are aware of the need for rigorously consistent terminology may not be sensitive to a corresponding necessity for standarized syntax. In many contexts, one way of expressing an idea may be more useful than an alternative way; that is, there may be a distinct advantage in adopting one of several altogether permissible ways as a standardized, or 'normalized', way of expressing a given idea.
>
> If we wish to express the idea which both S1 and S2 express, which of these two statements should we choose to express that idea? The answer depends, of course, upon our objectives --- the goals we have in mind when we consider the advantages and disadvantages of one or another mode of expression. If one of our goals is to make the fullest use of computers for purposes of information retrieval (as contrasted with document retrieval), it will be advantageous to express universal ideas by statements like S2 rather than by statements like S1. When ideas are expressed by statements like S2, much of the logical deduction involved in analyzing that statement and its relation to other statements can be done by means of a computer or some other automatic processing device." [Allen & Caldwell,p.234]

While Allen clearly recognized and later to some extent exploited logic as a tool for automating legal information retrieval, the trend in recent papers has been towards the use of logic as a tool for simplifying and standardizing legal drafting in order to make the meaning of legal documents more clear to humans. Edwards and Barber (1975), for example, translate a section of a statue into the form

IF A THEN B

where A consists of a number of sentences joined by AND, OR and NOT, and B is a single sentence. They then employ a computer program to derive a minimal "sum of products" expression which is logically equivalent to A, and propose using this as the preferred normalized form of the original legal rule.

De Bessonet (1980) discusses the use of logic and normalized statutes as a tool for drafting and maintaining the codified law peculiar (in the United States) to the French-inspired civil law system of Louisiana, but he also clearly acknowledges its potential importance for the development of automated systems.

This brief survey demonstrates that there already exists a considerable body of work concerned with the application of logic to law. The importance of logic in simplifying and clarifying statements of legal rules for both human and computer interpretation has been recognized, and work has begun on the development of "normal forms". What has been missing up to now is a straightforward method of implementing these ideas in a computer program.

4. LOGIC AND COMPUTERS

The use of logic as a programming language was developed in the early 1970's, and is described in Kowalski (1974a, 1974b), and in much greater detail in Kowalski's book "Logic for Problem Solving" (1979). The original PROLOG language, based on the clausal form of logic, and a interpreter for it were created at the University of Marseille by Colmerauer and his colleagues (1973). Several PROLOG interpreters have subsequently appeared, including one written by Grant Roberts at the University of Waterloo (1977). PROLOG has developed a sizeable following in Europe as a language for artificial intelligence research, though it has not as yet attracted a large number of users in North America.

For a full discussion of logic programming, Kowalski's book is necessary reading. I can only attempt to give the reader enough of the flavour of the PROLOG language to enable him to read the PROLOG version of the law of battery which concludes this paper.

A PROLOG program consists essentially of a set of clauses of the form:

```
A1 <- B1.
A2 <- B2.
      |
      |
An <- Bn.
```

where "<-" is read "if". Each Ai is a "relation" of the form Ri(a1,a2,...,ak), where Ri is a constant, the ai may be constants or variables, and k>=0 (if k=0, then Ri=Ai). Each Bi consists of zero or more such relations, connected by the boolean operators "&" (and) or "|" (or). If Bi is empty, then it is customary not to write the arrow, and the statement "Ai." is called an "unconditional assertion".

So, for example, consider the set of logical interrelationships defined by the following sentences: (1)John is a man. (2)Jim is a man. (3)John loves Mary. (4)Jim loves Mary. (5)Ellen loves any man who loves Mary. The content of these sentences can be written in PROLOG as follows (where, by convention, names of variables start with an upper-case letter, and names of constants start with a lower-case letter):

```
(1) man(john).
(2) man(jim)
(3) loves(john,mary).
(4) loves(jim,mary).
(5) loves(ellen,X) <- man(X) & loves(X,mary).
```

A PROLOG program is "executed" when the interpreter receives a request to prove a statement. Such a request is called a "query" or "goal statement". For example, suppose we pose the query "loves(ellen,jim)?". The PROLOG interpreter will attempt to prove this goal statement using only the clauses in its database. It works by searching for the relation "loves" on the left side of any clause. If it cannot find such a statement, the query fails. If it does find such a statement, it attempts to match "ellen" and "jim" with the corresponding positions in that relation in the database. This is impossible in clause (3), so the interpreter tries clause (4). Again it fails, so clause (5) is tried. Finally a match is made, with "ellen" matching "ellen" and "jim" matching the variable "X". Having made a match, the interpreter now replaces its current goal with everything on the right side of clause (5), carrying through its subsitution of "jim" for "X".

So the goal is now "man(jim) & loves(jim,mary)?". Both of these clauses must be proved if the original query is to succeed. But since "man(jim)" is an exact match of clause (2), it will be proved. And since "loves(jim,mary)" is an exact match of clause (4), it will also be proved. Neither of these clauses has anything on the right side, so the original proof is complete.

The interpreter stops when it has found the first proof of the query, or when it has tried every possible proof and found they all fail. It searches the database strictly in the order in which clauses appear, and constructs its proofs in a strict depth-first, left-to-right order.

For improved readability, PROLOG allows relation names to be declared "infix", "prefix", or "suffix". If we declare "loves" to be infix, and replace "man" by "is_a_man" declared suffix, the database can be rewritten as:

```
(1)  john is_a_man.
(2)  jim is_a_man.
(3)  john loves mary.
(4)  jim loves mary.
(5)  ellen loves X <- X is_a_man & X loves mary.
```

This is not very different from our original English-language representation, yet its meaning can be used directly by the PROLOG interpreter to answer queries. We will store legal information in the computer in a way very similar to this.

PROLOG programs have been written for a number of artificial intelligence applications, including a planning system (Warren,1974); natural language understanding (Colmerauer, 1978); and chess playing (Van Emden, 1981). Because of the close relationship between logic and law, it seems natural to investigate whether a PROLOG representation of law might be possible and useful.

5. LAW IN PROLOG

An ideal representation of law in a computer program would have to meet at least the following requirements:

1. It would be written in language as close as possible to the original statutes and cases which it represents. Any changes would be in the direction of simplifying the language and removing ambiguities, and would be as useful to a human reader as to the computer.

2. The computer representation of law would be directly readable by lawyers and even by laymen. Lawyers would be able to decide for themselves whether any given part of it was an accurate representation.

3. The representation of law in the program would be completely separate from and independent of implementation- and application-based segments of the program, such as the style and content of input and output.

4. The program would be capable of performing at least one legal task in a manner at least as proficient as a trained human doing the same job.

5. The representation could be improved or altered a little bit at a time, without requiring a major rewriting of the program. Changes to the law caused by judicial decisions in individual cases could normally be effected by limited changes to the computer representation, and ideally only by the addition of material corresponding to each single case.

The PROLOG program presented in this paper, though brief and incomplete, attempts to meet the first three of these goals, and to demonstrate some potential for meeting the fourth. Its ability to eventually meet the fifth goal is at present not proved.

Certain areas of law seem to be better candidates than others for such a computer representation. In particular, fields of law which have been "codified" -- criminal law being a well-known example -- seem to be among the most likely targets. The approach taken here to converting such a codified statute into a computer program is as follows:

(a) Restate each section of the statute in a "normal form" based on logical connectives.

(b) Translate the restated sections of law into PROLOG clauses.

(c) Add a basic "dictionary" of PROLOG clauses defining terms used by but not defined in the statute.

(d) Supply an interface that enables users to query the program by posing a fact situation and asking for the potential legal consequences. As the PROLOG interpreter tries to construct a proof that, for example, someone has committed a crime, provide an interface that lets it ask the user for any missing facts necessary to complete the proof.

Specifically, following Meldman (1975, 1977), we have taken Prosser's Handbook of the Law of Torts as our source of "codified" law, and attempted to translate normalized excerpts from and paraphrases of Prosser's version of the law of battery into PROLOG clauses. Our goal has been to write a computer program which, starting from a fact situation supplied by the user and working from PROLOG translations of legal rules, will supply logical chains of argument to support or refute the contention that one person has committed battery against another.

For example, we want to be able to enter a fact situation like

John throws a rock.
The rock hits Bob.

and then ask the computer program

Did John commit battery against Bob?

Whether it is a lawyer or a computer program that is

attempting to answer this question, the starting point must be the codified law. Prosser's basic definition of "battery" is as follows:

> "One is liable to another for unpermitted, unprivileged contacts with his person, caused by acts intended to result in such contacts, or the apprehension of them, directed at the other or a third person."[p.43]

This definition is typical of many legal rules in that it defines one concept, "battery", in terms of a number of necessary events such as "contact", "intention", and "apprehension". In criminal law, the list of statutory requirements which must all be proved before the defendant can be convicted of a crime is called "the elements of the offence". But the reduction of one concept into a number of specific requirements, all of which must be proved if the original is to be considered proved, also forms the essence of a PROLOG program. The PROLOG translation of a number of pieces of Prosser's law of battery can be seen in the Appendix.

Given that it seems possible to express legal rules as PROLOG clauses, how can we then hope to link up these legal abstractions with the real world of thoughts, actions, and events? In other words, how do we connect our representation of the law with a representation of the facts?

Case law and precedent give us one basic connection. If a judge in some ancient case held that hitting another person with a rock constitutes the contact required for battery, that case can provide a necessary link between the law of battery and the fact situation we used as an example.

In the absence of directly applicable precedent, arguments about the meaning of phrases like "contact with his person" will form the basis of the ultimate legal decision. Such arguments seek to connect the "normal" or "dictionary" meaning of a phrase with any special meaning it picks up from the context of the legal rule and any decided cases. As Meldman did in his 1975 thesis, we provide in our computer program a very simple "dictionary" of relevant terms like "contact", linking them to included concepts such as "hit", "strike", and "touch". This taxonomy seems to be adequate for the simple fact situations we have used to demonstrate the program, but a better implementation of the concept of "dictionary" will certainly be a necessity in the future.

Unfortunately, it is not enough for a human or computer program merely to be able to draw inferences from the facts as they are presented. Any law student learns, the first

time he interviews a client, that the essence of lawyering is the ability to ask good questions. The client does not know the law, so he does not know which facts are relevant; it is never good enough merely to listen to the client's story and then give a legal opinion, because the client will almost certainly leave out legally relevant pieces of information.

The PROLOG representation of law must also be able to ask good questions if it is ever to be useful. In its present state, the program makes every effort to prove a needed assertion from the facts it already has; but if it reaches a dead end, it asks the user whether the missing connection is true or not. If the missing assertion is a factual one, the program is functioning as a lawyer would in eliciting facts from his client. But the missing connection may be at a higher level, such as "does a mother worrying about her sleeping child being hit by a rock constitute the 'apprehension' required for battery?". In this case, by asking such a question, the program is indicating to the lawyer the potentially contentious and important aspects of his case. The ability to ask such questions, and their quality, may be the most important test of the value of a legal consultant, human or automated.

It is also clear that there is a very strong connection between analyzing legal rules and understanding natural language. Our computer program is attempting to answer a legal question by drawing logical inferences from a simple set of facts within a rather complex legal framework. This is somewhat similar to constraint-driven systems for natural language understanding, such as those described in Wilks & Charniak (1976), Winston (1977, Chapt.3), and Schank & Riesbeck (1981).

The program's representation of fact sentences is simpler and less powerful than those described in the just-mentioned works, and this is perhaps its weakest element. What strength it does have comes from following some suggestions in Kowalski (1979, Chapt. 2), to the effect that binary relations are the most useful method of representing sentences in PROLOG. This led to representing a sentence such as "Bob throws a rock." in the following PROLOG clauses, where "has_actor" and "has_object" are binary relations declared "infix":

```
throwing(t).
t has_actor bob.
t has_object rock.
```

In other words, each event is given a name (e.g. "t"), and the components of the event are forced into slots like "object" and "actor".

Such a representation is useful for relatively simple sentences, and serves the purpose here of enabling simple fact situations to be entered into the program in a way that is easy to understand, yet allows the basic reasoning abilities of the program to be demonstrated. I am certain, however, that a great deal of work must be done in this area if the program is ever to be used for more than demonstrations.

While the present implementation of a legal consultant in PROLOG is less than ideal in many respects, it does seem to demonstrate some potential for a logic-based computer representation of law. Much work remains to be done.

REFERENCES

Layman E. Allen, "Symbolic Logic: A Razor Edge Tool For Drafting and Interpreting Legal Documents", Yale Law Journal 66 (1957), p.833.

Layman E. Allen & M.E. Caldwell, "Modern Logic and Judicial Decision Making: A Sketch of One View", in Jurimetrics, ed. Hans W. Baade, 1963, p.213.

Robert Bigelow, ed., "Computers and Law: An Introductory Handbook", American Bar Association, Chicago, 1981.

B. Buchanan and T. Headrick, "Some Speculation About Artificial Intelligence and Legal Reasoning", Stanford Law Review 23 (1970), p.40.

Morris R. Cohen, "The Place of Logic in the Law", Harvard Law Review 29 (1916), p.622.

A. Colmerauer, H. Kanoui, R. Pasero, P. Roussel, "Un Systeme de Communication Homme-Machine en Francais", Universite d'Aix Marseille, Luminy, France; Rapport, Groupe Intelligence Artificielle.

A. Colmerauer, "An Interesting Natural Language Subset", Toulouse, Proc. Workshop on Logic and Data Bases.

Cary DeBessonett, "A Proposal for Developing the Structural Science of Codification", Journal of Computers, Technology and Law 8 (1980), p.47.

T.H. Edwards and J.P.Barber, "A Computer Method for Legal Drafting Using Propositional Logic", Texas Law Review 53 (1975), p.965.

Claude Fabien, "Computerized Legal Research in Canada", Canadian Law Information Council, Working Paper #3, Ottawa, Canada, 1979.

C.D. Hafner, "An Information Retrieval System Based on a Computer Model of Legal Knowledge", Ph.D. Thesis, The University of Michigan, Ann Arbor, 1978.

Cassius J. Keyser, "On the Study of Legal Science", Yale Law Journal 38 (1929), p.413.

Robert Kowalski, "Logic for Problem Solving", Dept. of Computational Logic, University of Edinburgh, Memo No. 75 (1974).

Robert Kowalski, "Predicate Logic as a Programming Language", in Proceedings of the IFIP 74, North Holland Publishing Co., Amsterdam, 1974, p.569.

Robert Kowalski, "Logic for Problem Solving", Elsvier North Holland, Amsterdam, 1979.

Hugh Lawford, "QUIC/LAW: Project of Queen's University", in National Conference on Automated Law Research, Georgia Institute of Technology, 1972, p.67.

L. Thorne McCarty, "The Application of Artificial Intelligence to Law: A Survey of Six Current Projects", Laboratory for Computer Science Research, Rutgers University, New Brunswick, NJ., 1981.

L.Thorne McCarty, "Reflections on TAXMAN: An Experiment in Artificial Intelligence and Legal Reasoning", Harvard Law Review 90 (1977), p.837.

Jeffrey A. Meldman, "A Preliminary Study in Computer-Aided Legal Analysis", Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA., 1975.

J.A. Meldman, "A Structural Model for Computer-Aided Legal Analysis", Rutgers Journal of Computers and Law 6 (1977), p.27.

Felix E. Oppenheim, "Outlines of a Logical Analysis of Law", Philosophy of Science 11 (1944), p.142.

Roscoe Pound, "Jurisprudence", v.1, West Publishing Co., St. Paul, MN, 1959.

William L. Prosser, "Handbook of the Law of Torts, 4th ed.", West Publishing Co., St. Paul, MN., 1971.

M. Radin, "Law as Logic and Experience", Yale University Press, 1971.

Grant M. Roberts, "An Implementation of PROLOG", M.Math thesis, University of Waterloo, 1977.

Jerome S. Rubin, "LEXIS: An Automated Research System", in National Conference on Automated Law Research, Georgia Institute of Technology, 1972.

R. Schank and C. Riesbeck, "Inside Computer Understanding", Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.

J.A. Sprowl, "Automating the Legal Reasoning Process: A Computer that Uses Regulations and Statutes to Draft Legal Documents", American Bar Foundation Journal (1979), p.1.

R. Stamper, "LEGOL: Modelling Legal Rules by Computer", in Computer Science and Law: An Advanced Course, ed. B. Niblett, Cambridge University Press, 1980, p.45.

Samuel J. Stoljar, "The Logical Status of a Legal Principle", University of Chicago Law Review 20 (1953), p.181.

I. Tamello, "Sketch for a Symbolic Juristic Logic", Journal of Legal Education 8 (1956), p.277.

M.H. Van Emden, "Chess Endgame Advice", 1980.

D.H.D. Warren, "WARPLAN: A System for Generating Plans", Dept. of Artificial Intelligence, University of Edinburgh, DCL Memo 76.

D.A. Waterman and M. Peterson, "Rule-Based Models of Legal Expertise", Proceedings of the First Annual National Conference on Artificial Intelligence, Stanford University, (1980), p.272.

Yorick Wilks and Eugene Charniak, "Computational Semantics", North Holland, New York, 1976.

Patrick H. Winston, "Artificial Intelligence", Addison-Wesley, 1977.

```
/*************************************************************
                          BATTERY
One is liable to another for unpermitted, unprivileged
contacts with his person, caused by acts intended to
result in such contacts, or the apprehension of them,
directed at the other or a third person.
                                        <Prosser, p.43>

*************************************************************/

X is_liable_to Y if X commits_battery_against Y.

X commits_battery_against Y
   if person(X) &
      person(Y) &
      battery_event_occurs_with(X,Y,Contact,Act,Causation) &
    battery_intention_occurs_with(X,Y,Contact,Act,Causation) &
    print('I HAVE CONCLUDED THAT'.X.
          'COMMITS BATTERY AGAINST'.Y.nil).

battery_event_occurs_with(X,Y,Contact,Act,Causation)
 if contact(Contact) &
    Contact has_object Ysperson &
    Ysperson is_the_person_of Y &
    Contact occurs &

    event(Act) &
    Act has_actor X &
    not(Act has_direction X) &
    Act occurs &

    causation(Causation) &
    Causation has_actor Act &
    Causation has_object Contact &
    Causation occurs &
    not(permitted(Contact,Y,X)) &
    not(privileged(Act,Contact,Y,X)) &

    print('I HAVE CONCLUDED THAT THE ACT'.Act.'AND CONTACT'.
          Contact.'CONSTITUTE THE ACTS REQUIRED FOR BATTERY BY'.
          X.'AGAINST'.Y.nil).

battery_intention_occurs_with(X,Y,Contact,Act,Causation)
 if intention(Intention) &
    Intention has_actor X &
    ((Intention has_object Contact)
    |(apprehension(Apprehension) &
      Apprehension has_object Contact &
      Intention has_object Apprehension)) &
    intention(Intention) occurs &
    print('I HAVE CONCLUDED THAT'.X.
          'HAD THE INTENTION REQUIRED FOR BATTERY'.nil) &
```

```
/************************************************************
Permission includes actual consent. But a manifestation
of consent will be equally effective. In addition, the
defendant is sometimes at liberty to infer consent as
a matter of usage or custom.
                        <Prosser pp.117-120 (paraphrase)>
************************************************************/


permitted(Act,Plaintiff,Defendant)
 if consent(C) &
    C has_actor Plaintiff &
    C has_object Act &
    ((consent(C) occurs)
    |(manifestation(M) &
      M has_actor Plaintiff &
      M has_object C &
      M has_direction Defendant &
      manifestation(M) occurs)
    |(inference(I) &
      I has_actor Defendant &
      I has_object C &
      inference(I) occurs &
      I is_permitted_by_custom_or_usage)).



/************************************************************
Contact with the plaintiff is privileged if
it results from an act of self-defence.
                                    <Prosser, p.125>
************************************************************/

privileged(Act,Contact,Plaintiff,Defendant)
 if contact(Contact) &
    Contact has_object X &
    X is_the_person_of Plaintiff &
    event(Act) &
    Act has_actor Defendant &
    causation(Causation) &
    Causation has_actor Act &
    Causation has_object Contact &
    causation(Causation) occurs &
    self_defence(SD) &
    SD has_actor Defendant &
    SD has_object Plaintiff &
    Act has_purpose SD.




/************************************************************
                    INTENT
```

```
A person intends a result when he acts for the
purpose of accomplishing it, or believes that
the result is substantially certain to follow
from his act.
                                    <Prosser, p.40>
**********************************************************/

intention(I) occurs
 if I has_actor X &
    I has_object Result &
    event(Act) &
    Act has_actor X &
    ((Act has_purpose Result &
      event(Act) occurs)
    |(event(Act) occurs &
      causation(C) &
      C has_actor Act &
      C has_object Result &
      belief(B) &
      B has_actor X &
      B has_object C &
      B occurs)).
```

```
/*************************************************************
        This is the basic dictionary. It
        is really only a hierarchy of words;
        so, for example, it tells us that brick
        and weapon are both physical objects,
        but it doesn't tell us how to distinguish
        between them.
*************************************************************/


thing(X) if object(X).
thing(X) if event(X).
thing(X) if value(X).

object(X) if physical_object(X).
object(X) if person(X).
object(X) if legal_object(X).

event(X) if movement(X).
event(X) if contact(X).
event(X) if communication(X).
event(X) if mental_event(X).
event(X) if legal_event(X).

value(X) if size(X).
value(X) if colour(X).
value(X) if health(X).

physical_object(X) if brick(X).
physical_object(X) if weapon(X).
physical_object(X) if firearm(X).

person(X) if man(X).
person(X) if woman(X).
person(X) if child(X).
person(X) if boy(X).
person(X) if girl(X).

legal_object(X) if judgement(X).
legal_object(X) if court(X).

movement(X) if throwing(X).
movement(X) if running(X).

contact(X) if touch(X).
contact(X) if hit(X).
contact(X) if strike(X).

communication(X) if telling(X).
communication(X) if saying(X).
communication(X) if manifesting(X).

mental_event(X) if learning(X).
```

```
mental_event(X) if belief(X).
mental_event(X) if apprehension(X).

legal_event(X) if consent(X).
legal_event(X) if selling(X).
legal_event(X) if civilaction(X).
legal_event(X) if causation(X).
legal_event(X) if battery(X).
legal_event(X) if assault(X).
legal_event(X) if intention(X).
legal_event(X) if inference(X).
legal_event(X) if self_defence(X).
legal_event(X) if arrest(X).

size(X) if short(X).
size(X) if tall(X).
size(X) if long(X).

colour(X) if red(X).
colour(X) if yellow(X).
colour(X) if blue(X).

health(X) if healthy(X).
health(X) if ill(X).

X is_the_person_of X.       /* This definition of is the person of
                               is required by Prosser's definition
                               of battery.                          */
```

```
/************************************************************
      This is a PROLOG interpreter written in PROLOG.
      Its purpose is to create an interpreter that will ask
      the user as a last resort whether a missing step
      in a proof is true.
************************************************************/


prove(X) <- X.

prove(not X) <- prove(X) & / & fail.
prove(not X).

prove(X & Y) <- / & prove(X) & prove(Y).

prove(X | Y) <- / & (prove(X) | prove(Y)).

prove(X) <- (X if Y) & prove(Y).

(not X)? <- X? & / & fail.
(not X)?.

(X & Y)? <- / & (X?) & (Y?).

(X | Y)? <- / & ((X?) | (Y?)).

X? <- prove(X).

X? <- prove(X) & / & fail.

X? <- (X if Y) & (Y?).

X? <- (X if Y) & / & fail.

X? <- ask(X).

ask(X) <- print('IS THE FOLLOWING TRUE?'.nil) &
      write(' ') &
      write(X) &
      write(' ') &
      read(Answer) &
      / &
      ((Answer equals yes) |
       (Answer isnotequalto no & X equals Answer)) &
      addax(X).


X equals X.
X isnotequalto X <- / & fail.
X isnotequalto Y.

print(nil) <- newline.
print(X.Y) <- writech(X) & writech(' ') & print(Y).
```

```
/************************************************************
      These are the operator definitions for the
      program. They set the mode and binding level
      of each operator.
************************************************************/


op(if,rl,15).
op(?,suffix,33).

op(not,prefix,49).
op(equals,rl,50).
op(is_not_equal_to,rl,50).
op(occurs,suffix,50).

op(is_liable_to,rl,50).
op(commits_battery_against,rl,50).
op(is_the_person_of,rl,50).
op(is_permitted_by_custom_or_usage,suffix,50).

op(has_actor,rl,50).
op(has_object,rl,50).
op(has_direction,rl,50).
op(has_purpose,rl,50).
```

```
/************************************************************

            A FACT SITUATION

************************************************************/


throwing(e1).
e1 has_actor al.
e1 has_object rock.
e1 occurs.

hit(e2).
e2 has_actor rock.
e2 has_object bob.
e2 occurs.

person(al).
person(bob).
physical_object(rock).
```

```
WELCOME TO WATERLCO PROLOG 1.1                COMMENTARY
load(LAW)<-
loadlaw.
loadlaw<-
al isliableto bob?.                              The query.
IS THE FOLLOWING TRUE?                    PROLOG interpreter asks user.
 .
e1 hasdirection al.
 .
no.                                              User's response.
IS THE FOLLOWING TRUE?
 .                                    Was there a causation involved?
causation(*1).
 .
causation(cause1).
IS THE FOLLOWING TRUE?
 .
cause1 hasactor e1.
 .
yes.
IS THE FOLLOWING TRUE?
 .
cause1 hasobject e2.
 .
yes.
IS THE FOLLOWING TRUE?
 .                                    Did (al throws a rock) cause
                                      (the rock hits bob)?
cause1 occurs.
 .
yes.
IS THE FOLLOWING TRUE?
 .                                      Was a consent involved?
consent(*1).
 .
consent(con1).
IS THE FOLLOWING TRUE?
 .
con1 hasactor bob.
 .
yes.
IS THE FOLLOWING TRUE?
 .
con1 hasobject e2.
 .
yes.
IS THE FOLLOWING TRUE?
 .                          Did bob consent to (the rock hits bob)?
consent(con1) occurs.
 .
no.
IS THE FOLLOWING TRUE?
 .                                Was a manifestation involved?
manifestation(*1).
 .
```

```
no.
IS THE FOLLOWING TRUE?
.                                        Was an inference involved?
inference(*1).
.
inference(inf1).
IS THE FOLLOWING TRUE?
.
inf1 hasactor a1.
.
yes.
IS THE FOLLOWING TRUE?
.
inf1 hasobject con1.
.
yes.
IS THE FOLLOWING TRUE?
.                                  Did a1 infer that bob consents to
                                          (the rock hits bob)?
inference(inf1) occurs.
.
no.
IS THE FOLLOWING TRUE?
.
causation(cause1) occurs.
.
yes.
IS THE FOLLOWING TRUE?
.                                     Was self-defence involved?
selfdefence(*1).
.
no.
I HAVE CONCLUDED THAT THE ACT e1 AND CONTACT e2
CONSTITUTE THE ACTS REQUIRED FOR BATTERY BY a1 AGAINST bob

IS THE FOLLOWING TRUE?
.                                        Was an intention involved?
intention(*1).
.
intention(int1).
IS THE FOLLOWING TRUE?
.
int1 hasactor a1.
.
yes.
IS THE FOLLOWING TRUE?
.
int1 hasobject e2.
.
yes.
IS THE FOLLOWING TRUE?
.                                  A way of proving intention:
                                   Did (a1 throws a rock) have the
                                   purpose (the rock hits bob)?
e1 haspurpose e2.
```

.
yes.
IS THE FOLLOWING TRUE?
.
event(e1) occurs.
.
yes.
I HAVE CONCLUDED THAT a1 HAD THE INTENTICN REQUIRED
FOR BATTERY.
I HAVE CONCLUDED THAT a1 CCMMITS BATTERY AGAINST bob
a1 isliableto bob?<-