10
Representing Legislation as Logic Programs

M. Sergot
Department of Computing,
Imperial College of Science and Technology
London, UK

Abstract

The law is a rich and natural source of expert system applications of different and complementary types. In particular, legislation which is definitional in character can often be formalized as rules in logic programs so that, when executed by an augmented PROLOG system such as APES, it can be queried as though it were a data base. The system in turn queries the user for additional information which it needs, and it can explain and justify its conclusions in terms of the original legislation. The resulting system has many of the features associated with expert systems, but it can be regarded more usefully as a precise and executable specification of what the legislation tries to express. This suggests that executable formalizations can aid the drafting process itself, and that the techniques have application outside the law for formulating and applying regulations in all kinds of organizations.

1. INTRODUCTION

The law provides an ideal experimental domain for research in many areas of artificial intelligence (AI). It is a rich source of difficult and challenging problems which involve issues of knowledge representation, the analysis of natural language, and the automation of practical and common-sense reasoning. In contrast with many other experimental domains, however, the successful application of AI techniques in law will give rise to practical applications of immense social significance. Even now it is possible to build practical and useful systems requiring relatively unambitious techniques which are already well understood. This paper is concerned primarily with systems which represent the law as computer programs, as legal 'expert systems' which can apply the law to the solution of specific legal problems.

The rules and regulations which govern the running of all institutions and organizations have exactly the same character as legal provisions. The terms 'law' and 'legislation' are used here in this wider sense, to

include the laws of organizations as well as the laws of a state. The domain of potential applications is consequently very wide. It suggests, in particular, a non-conventional approach to the construction of software for conventional data-processing applications. A payroll system could be based directly on tax and sick pay legislation, and could include, for example, a representation of the company pension scheme, the rules which govern holiday allocation, and promotion regulations. In the Logic Programming Group at Imperial College we have constructed a number of experimental systems treating fragments of these components, some of which are described in later sections of this paper.

The drafters of legislation, and of regulations more generally, are normally expected to formulate the law as clearly and precisely as possible. This makes legislation an ideal application for logic programming. Many regulations are 'definitional' in character and can often be formulated as rules in logic programs. This formalization can then be queried as though it were a data base and it in turn will query the user for information which it needs to solve a given problem. Before discussing this approach in more detail, however, it is instructive to consider first a different and complementary type of legal expert system.

2. LEGAL EXPERT SYSTEM 1

There is one sense in which the law as an application area is no different from any other domain which requires the application of specialized professional skill and expertise. One could imagine constructing an expert system which incorporates in rule-based form the expertise of an experienced lawyer, for the sake of example, that of an advocate who gives good advice to clients accused of criminal offences. Such a system might well include rules like:

x should plead guilty of *y*
  if *x* accused of *y*
  and *x* did *y*
  and penalty for *y* is light
  and not *x* has reasonable defence for *y*

x should plead not guilty of *y*
  if *x* accused of *y*
  and not *x* did *y*
  and penalty for *y* is heavy
  and *x* has reasonable defence for *y*

and many more. Auxiliary rules would be needed to define amongst other things what constitutes a reasonable defence and when and which penalties are associated with the various offences.

The rules above are expressed as logical implications of the form

A if *B* and *C* and . . .

These implications are Horn clauses, extended as in PROLOG to allow negated conditions in rules. This fragment of first-order predicate logic is termed here 'extended Horn clause logic'.

In the two rules above, the symbols 'guilty', 'not guilty', 'light' and 'heavy' are constant symbols, '*x*' and '*y*' are variables, and predicate symbols like

'. . . should plead . . . of . . .'
'. . . accused of . . .'
'. . . did . . .'
'. . . penalty for . . . is . . .'
'. . . has reasonable defence for . . .'

have been written in distributed infix form to aid readability.

Legal Expert System 1 is intended to represent an expert system in the 'classic' style of MYCIN [1] or PROSPECTOR [2]. It would be constructed by a process in which the expert's implicit, or even subconscious, problem-solving methods are made explicit and expressed in rule-based form. These rules then combine to give a system which simulates the problem-solving behaviour of the original expert.

It is important to note that the rules of Legal Expert System 1 express an expert's opinion of the law, and as such have no legal authority. The law itself, for example, may well not prescribe what constitutes a 'reasonable defence'. This is a concept which the lawyer finds useful when advising clients, and one which he has built up by experience over the years. Legal Expert System 1 incorporates little knowledge of what the law actually says: it is more concerned with simulating the legal problem-solving process than with reasoning explicitly about the law itself.

Legal Expert System 1 is imaginary, and it is worth digressing to remark on the practicality of the approach it represents. There is nothing to suggest that building expert systems for legal practice would be any harder (or easier for that matter) than the construction of similar systems which already exist in the more traditional domains like medicine or geology. Waterman and Peterson have described a system, LDS [3, 4], which they constructed in the programming language ROSIE for the problem of settling claims in product liability cases. LDS attempts to model how such claims are actually settled in practice. Although it is primarily a research tool for analysing the effects of alternative liability rules, Waterman and Peterson have indicated on occasion [3] that LDS could be adapted straightforwardly for use as a consultative expert system in its specialist domain.

This paper is not concerned with systems which simulate a lawyer's practical problem-solving skills, however, but with systems which aim to represent the law itself, for whatever purpose. Legal Expert System 1 has been presented at this stage to introduce a recurring theme: the idea of legal expert system, not as decision *taker,* but as decision-taking *aid.*

Legal Expert System 1, potentially at least, can give conflicting advice. In general, there will be rules which suggest that the client should plead guilty, and others which advise him to do the opposite. One could, as in many expert systems, associate some kind of certainty factor with each rule, and then provide in the system's inference engine some method of combining the various certainties to eliminate the apparent inconsistency, ranking the system's advice according to these certainty measures. In many applications this is precisely what is required: a system which will weigh up the conflicting rules and facts, arriving at a small number of specific recommendations with some indication of how good they are likely to be in practice.

This approach is not recommended for the imaginary Legal Expert System 1. Real experts, particularly lawyers and accountants do give conflicting advice, and for good reason. It is the client himself and not the expert who must make the final decision: in this example, whether to plead guilty or not guilty. The expert's role is to present the various alternatives, and to point out the consequences of deciding one way or the other. Ultimately, the client must decide.

Legal Expert System 1, it is argued, is useful precisely because it has the ability to generate conflicting advice. Of course, an expert lawyer whose advice is 'either plead guilty or plead not guilty' is of little use. But expert systems have an important and characteristic feature. They can explain their recommendations by showing the assumptions on which these recommendations are based. When the recommendations are in conflict, these explanations take on the character of arguments. As an aside, we should not be worried that a single set of logical sentences can produce conflicting conclusions. There is nothing logically inconsistent about advising someone that he should plead guilty for one reason, and not guilty for another.

Thus the output from Legal Expert System 1 is not simply 'guilty' or 'not guilty', but rather the whole set of arguments which can be constructed from the facts of a specific case, some arguing for and others against a particular decision. The user of the system, the client as it were, is free to examine these arguments, compare them one against the other, and choose to accept one, or reject them all in favour of a more persuasive argument from some other source.

It would be tempting, of course, to extend the system so that it could also assist in the evaluation of arguments. A real expert may want to indicate the likelihood that his assumptions, and his recommendations,

are correct. It may be possible, for example, to estimate the probability that a particular defence will be accepted as 'reasonable' in court. One could envisage systems which apply (meta-level) criteria to distinguish weak arguments from strong, by considering, for example, the chain of reasoning involved, and the source of the various rules which are used. The evaluation of arguments is a separate exercise which needs further investigation, however, and it is not pursued further in this paper.

The importance of constructing arguments which are in conflict will be considered later when we look at the effects of vagueness and case law. For now, an example which contrasts directly with Legal Expert System 1 is given below. It is concerned with representing, not the legal problem-solving process, but the content of a piece of legislation.

3. LEGAL EXPERT SYSTEM 2

Legal Expert System 2 deals with entitlement of claimants to Supplementary Benefit, one of the Social Security benefits in the United Kingdom. The system was implemented by Peter Hammond [5] as a feasibility study for the United Kingdom's Department of Health and Social Security (DHSS), to demonstrate the usefulness of expert systems within the DHSS, and to estimate the effort involved in constructing such systems. The aim was not to build a system which would be used by DHSS officials in practice. Hammond was assisted by Ian Pickup, an expert on Supplementary Benefit from the DHSS, who supplied detailed knowledge of the relevant legislation and how it is interpreted and applied within the DHSS.

In Hammond's system, entitlement to Supplementary Benefit is described in extended Horn clauses, which are executed on a micro-computer by APES (Augmented PROLOG for Expert Systems) [6, 7]. APES is described below. The top-level rule for entitlement to Supplementary Benefit is expressed in the pseudo-natural language syntax accepted by APES, as

x is entitled to supplementary benefit
  if not *x* is disqualified by sex
  and not *x* is a juvenile
  and educational status of *x* is OK
  and *x* is a GB resident
  and *x* is excused or registered for work
  and *x* needs financial help
  and not *x* is disqualified by trade-dispute.

Here negative conditions like

'not *x* is disqualified by sex'

are treated by negation as failure [8]:

not *P* holds if and only if all the ways of showing *P* fail.

Most of the conditions in the entitlement rule are themselves defined by lower-level rules. For example

x is disqualified by sex
if sex of *x* is female
and *x* has partner
and *x* is living with partner

x needs financial help
if capital of *x* is less than max-capital for supplementary benefit
and requirements of *x* exceed resources of *x*

although some 'rules' have no conditions

max-capital for supplementary benefit is 2500
pensionable age for male is 65
pensionable age for female is 60.

Other conditions in the entitlement rule, such as

'*x* is a GB resident',

and in the lower level rules,

'sex of *x* is *y*',

for example, require information about the individual claimant, and are not defined by lower-level rules.

Specific information about individual claimants can always be supplied as rules in a supplementary PROLOG data base. Usually, however, it is more convenient and more realistic if the system requests this specific information when it is required. This eliminates the need to supply all information in advance, and it frees the user from having to know precisely what information is relevant before the consultation begins.

APES [6, 7] is an extended PROLOG interpreter which incorporates Query-the-User [9], a model of input–output for logic programs in which the user is regarded as an additional source of information which can contribute to the problem-solving process. When APES cannot solve a problem itself for lack of information, it assumes the information is available from the user and it asks the user to supply it. In Query-the-User (and APES), the user is regarded as a kind of external data base.

APES also maintains a trace of its execution, providing explanations on request which can be examined interactively by the user. In logic-based systems like APES, these explanations correspond to logical proofs that the answers computed follow logically from the combined knowledge of

the system and the information supplied by the user. APES itself is implemented in micro-PROLOG [10].

As with any logic program, an interaction with APES is initiated by a query from the user. In the following sample dialogue, user input is in boldface:

confirm (Peter is entitled to supplementary benefit)
What is Peter's sex? **male**
How old is Peter (in years)? **23**
Is Peter in full-time education? **why**

---

if Peter is not in full-time education
then educational status of Peter is OK

not Peter is disqualified by sex
not Peter is a juvenile, so
if educational status of Peter is OK
and Peter is a GB resident
and Peter is excused or registered for work
and Peter needs financial help
and not Peter is disqualified by trade-dispute
then Peter is entitled to supplementary benefit

Is Peter in full-time education? **no**

.
.
.

Is Peter currently involved in a trade-dispute? **no**

Yes, I can confirm that
Peter is entitled to supplementary benefit **how**

To deduce
Peter is entitled to supplementary benefit
I used the rule

x is entitled to supplementary benefit
if not *x* is disqualified by sex
and not *x* is a juvenile
and educational status of *x* is OK
and *x* is a GB resident
and *x* is excused or registered for work
and *x* needs financial help
and not *x* is disqualified by trade-dispute

I can show
1. not Peter is disqualified by sex
2. not Peter is a juvenile

3. educational status of Peter is OK
4. Peter is a GB resident
5. Peter is excused or registered for work
6. Peter needs financial help
7. not Peter is disqualified by trade-dispute

Explain? **6**

.
.
.

etc.

The clauses describing entitlement to Supplementary Benefit can function as a program to check an individual's claim. As Hammond points out, however, the same rules can be used for many other purposes: to determine who is entitled to Supplementary Benefit

find (x : x is entitled to supplementary benefit),

or to discover all the benefits to which Peter is entitled,

find (x : Peter is entitled to x),

or to find juveniles who may be in need of financial help,

find (x : x is a juvenile and x needs financial help).

Approximately 70 of Hammond's rules cover entitlement to Supplementary Benefit of about 90 per cent of all claimants. Deciding the claims of the other 10 per cent involves a great deal of detail, however, so that extending the system to cater for all claimants would increase the number of rules several times over. In fact, Hammond's 70 rules handle almost exactly those claims which are processed routinely, and ignore the problematic cases in which DHSS officials require assistance. A practical implementation for the DHSS (but not necessarily for anyone else) would have to be more concerned with the less routine cases. From the DHSS's point of view, however, the system did demonstrate what is possible in a limited amount of time.

The first version of the system, comprising about 50 rules (for 60 per cent of claimants), was produced in two days, including a half-day introduction to PROLOG for Ian Pickup. These rules ran in APES on a 64K micro-computer, although they were later moved to a 128K system. The number of rules was increased to 70 in two more days. In a subsequent exercise, the system was extended to calculate the amount of benefit payable and to determine the date on which the payments are made. The final system contained more than 200 rules and was produced in about one man-month overall.

Two points need to be stressed. Firstly, the hardware requirements for

the system are extremely modest. The feasibility of building legal expert systems on small computers has since been demonstrated by a number of other pilot projects at Imperial College, some of which are referred to in later sections of this paper. Secondly, Hammond and Pickup were able to avoid many of the time-consuming problems associated with knowledge elicitation for expert systems, and this is reflected in the large number of rules they were able to formulate in the short time available to them. Their experience seems to confirm a commonly expressed opinion. The law, and regulation based organizations more generally, are a natural source of expert system applications. The attraction of the application domain is clear. By its very nature, the law is well documented, and the existence of regulations and reported examples makes the process of knowledge elicitation very much easier. Even if the documentation is not already in a form which can readily be expressed in computer-intelligible terms, it provides at the very least a convenient framework around which the knowledge acquisition process can proceed. This is not to say that there are no knowledge acquisition or knowledge representation problems.

Hammond's Supplementary Benefit system is an expert system in every sense. Knowledge is expressed explicitly in rule-based form; the system can request missing information which it needs, and it can explain and justify its conclusions. The knowledge which the rules express moreover is that of a human expert, i.e. Ian Pickup's opinion of what entitlement to Supplementary Benefit requires. This opinion is based on extracts from enabling legislation and various supplementary regulations, on a condensation of the relevant case law, and on familiarity with the DHSS's interpretation of the law and with its application in practice.

Consequently, the rules in Hammond's system are without legal authority. They express what Ian Pickup thinks entitlement to Supplementary Benefit requires, and not necessarily what the law actually says. So the question now arises: to what extent is it possible to eliminate the human expert from this process altogether and base a system directly on the actual legislation itself?

4. REPRESENTING LEGISLATION IN LOGIC

A part of the law is expressed as written provisions or explicit regulations, but the law in force at any one time is also determined by the decisions which were taken in previous cases. In many systems of law, case law has a legally binding nature through the doctrine of precedent. In other legal systems, and in organizations of all kinds outside the law, precedent may not be legally binding, but there is a role for case law nevertheless. The decisions taken in previous cases always have to be considered, if only for the sake of consistency and fairness. This paper

concentrates on explicitly written legal provisions, although case law is mentioned also.

It is sometimes suggested that the law is also influenced by general legal principles ('no man may profit by his own wrong doing', for example). The legal status of such principles, and to what extent they must be considered in practice, remains a matter of debate for scholars of jurisprudence. In most cases, and certainly for the purposes of this paper, the practical effect of these vague principles can be ignored.

If a fragment of written legislation and its associated case law can be formalized in some mechanizable form of logic, then that formalization can function as a program which interprets and applies the law. A logical formulation of Social Security regulations, for example, can be regarded as a set of axioms. The entitlement of a claimant to a particular benefit is an example of a theorem we might try to prove from these axioms. As we shall see later, the proof of such a theorem (trivial by the standards of mathematics but useful nevertheless) is within the capabilities of even relatively unsophisticated theorem provers like PROLOG.

The advantages of this logic-based approach over the use of conventional programming languages are many. Legislation is not typically expressed as detailed algorithms: representing a complex and very high-level specification of some legal concept in a low-level algorithmic programming language is rarely a practical possibility. Symbolic logic, and at this stage we do not need to consider exactly which form of symbolic logic is appropriate, provides a precise language which resembles the draftsman's natural language closely. Symbolic logic, in the form of logical implications, can be regarded as the purest form of rule-based programming language, and the formalization inherits many of the advantages normally ascribed to expert systems. In particular, the use of a rule-based language makes the program comparatively easy to read, for lawyers and users alike. It is also easy to maintain and gives a natural way of generating explanations by constructing a trace of the rules which were used during execution.

Clearly these advantages are enhanced if the particular form of logic allows formalizations which resemble as closely as possible the style and structure of the original legislation. A close correspondence between legislation and its formalization increases confidence that the formalization is, in some sense, correct, and it makes the resulting program easier to maintain as the legislation changes.

It is possible in theory to build automated theorem provers for any system of symbolic logic; in practice, efficient proof procedures exist only for a small number of logical systems. Typically, theorem provers which are tolerably efficient are restricted to various fragments of first-order predicate logic. Extended Horn clause logic is one such fragment. It is the basis for the computational paradigm, logic programming, and for the

programming language PROLOG. Indeed, PROLOG can be regarded as a simple theorem prover which incorporates various restrictions to make it an efficient executor of logic programs. PROLOG has a fixed method of executing programs and this imposes severe limitations on its use as a general-purpose theorem prover (it sometimes goes into loops, for example). Nevertheless, PROLOG can be used to derive many useful and non-trivial logical consequences from axioms expressed as extended Horn clauses.

Whether or not PROLOG will be adequate for the needs of a particular application will depend on the kind of problem-solving tasks we wish to support and on the style of the interaction we wish to provide. But there are reasons, independent of these considerations, to suggest that extended Horn clauses are an appropriate logic for representing many kinds of legislation. Extended Horn clause logic is a fragment of first-order logic which does not allow disjunctive conclusions in rules. But we can dispense with disjunctive conclusions for many practical purposes. In particular, rules with disjunctive conclusions

A or *B* if *C*

are seldom encountered in law. Legislation is normally concerned with specifying the conditions under which some definite conclusion can be made.

Dispensing with disjunctive conclusions in rules is an important simplification (it allows us to apply relatively efficient theorem provers), but we could not realistically expect to formalize any sizeable fragment of legislation in pure Horn clauses. At the very least we shall have to allow negated conditions in rules if our formalizations are to bear any resemblance to the original texts on which they are based. Allowing negated conditions in rules takes us beyond Horn clause logic, and would seem to re-introduce the need to reason with all of first-order logic. But there is a method of treating negation without going outside the capabilities of Horn clause theorem provers. Keith Clark [8] has shown that classical negation

not *Q* is true if and only if *Q* is false

can be replaced by negation as failure

conclude not *Q* when all ways of proving *Q* fail

whenever we can make a 'Closed World' assumption

anything which is unknown to be true is assumed to be false.

Negation as failure is the easiest type of negation to deal with computationally. It is also a type of negation which is often appropriate when reasoning with law. Given the regulations which describe entitle-

ment to some benefit, it is natural to assume that the regulations cover all the possible ways of being entitled, that there is not some other way of becoming entitled which the legislators have not bothered to mention. This is precisely the assumption which justifies the use of negation as failure: if we cannot determine that a particular individual is entitled to the benefit by any of the routes explicitly mentioned, it is normal to conclude that the individual is not entitled to that benefit. There will be occasions, of course, when it is difficult or impossible to say that all the relevant legal provisions have been considered. In those circumstances the treatment of negation by failure is not justified, and we shall have to look to some other, necessarily less efficient, method for dealing with negation. The suggestion that we can often treat negation as failure is purely a practical one. Under very specific conditions, Clark's result allows us to read negation classically but compute with it by failure. Nothing in what follows turns on this, however.

I have suggested that disjunctive conclusions are seldom encountered in law, but I have to be more precise about what I mean in making such a claim. Suppose we have rules

$$A \text{ if } B \tag{1}$$

$$A \text{ if } C \tag{2}$$

which formalize some piece of legislation. Suppose we want to say further that these are the only ways of establishing conclusion '*A*' which the legislation allows. We can express this 'only if' information either in the object language by replacing the two rules (1) and (2) with the biconditional

$$A \text{ iff } B \text{ or } C \tag{3}$$

or alternatively by stating in the meta-language that rules (1) and (2) are the only ways of establishing '*A*'. In both cases it is legitimate to conclude '*B* or *C*' given '*A*'. In the first case, '*B* or *C*' is logically implied by rule (3) and the assertion '*A*'. In the second case, '*B* or *C*' follows by meta-level reasoning: if '*A*' holds and rules (1) and (2) are the only ways that '*A*' could hold, then either '*B*' holds or '*C*' holds.

Clark [8] showed that for every proof of negation by failure (using 'only if' assumptions expressed in the meta-language) there exists a structurally similar proof of the same conclusion using ordinary negation (with 'only if' assumptions expressed in the object language together with appropriate axioms of equality and inequality). This is the 'Closed World Assumption' which justifies negation as failure.

In suggesting that negation as failure is often appropriate when reasoning with law, I am in effect claiming that the law often expresses 'only if' information, either explicitly or implicitly. If we choose to write

'only if' halves for rules (1) and (2) in the object language, we are adding a rule

B or *C* if *A*

to the formalization. This is a rule with a disjunctive conclusion. I am not claiming that such rules are not encountered in law nor that we shall never want to include them in our representations.

When we apply some piece of law, we have some facts about a particular case and we use the legislation to determine what legal consequences follow from these facts. A claim that disjunctive conclusions are seldom encountered in law means that the rules which are used for this purpose (whether we write them down or not) seldom have disjunctive conclusions. These are 'if rules', and there is little difficulty in identifying them in practice. When we construct a formalization of the law, we write down these 'if rules' (and it is suggested that extended Horn clause logic is a natural choice for the formal language in which to do so). In addition to 'if rules', we shall want to include 'only if' information, in general. If we are constructing a program which will only be used to apply the law, we can, if we choose, interpret negation as failure and thus in effect include the 'only if' rules implicitly. A suggestion that the law tends to be naturally expressed as 'if and only if' rules is an argument for the use of negation as failure, as long as we limit ourselves to programs which apply the law. If we want our representation of the law to be used for some other purpose (to assert, for example, that a particular individual is entitled to Supplementary Benefit and investigate what follows from this assertion), or if we choose not to treat negation as failure for whatever reason, then we shall want to express the 'only if' part of the law explicitly, normally in the object language. These 'only if' rules will almost always have disjunctive conclusions. If the formalization includes explicit 'only if' rules, then Horn clause theorem provers will not be adequate for our purposes. In what follows, I shall be less careful: I might claim that legal rules do not have disjunctive conclusions, or that extended Horn clauses can express many legal provisions; I shall be referring only to the 'if' part of the law.

That understood, I have suggested that many legal provisions will translate naturally into Horn clauses extended to allow negated conditions. There is a certain amount of evidence to support this claim in Ronald Stamper's work on the LEGOL project (see for example [11, 12]). One aim of Stamper's project was to design a computer language for representing legislation. Stamper developed a method of analysis based on various semantic considerations, and this semantic model in turn formed the basis for a computer language in which rules could be written to simulate the effects of legal provisions. The LEGOL language (in its executable versions) was based on relational algebra. It is therefore fairly

straightforward to reinterpret LEGOL rules as Horn clauses (and it is argued elsewhere [13] that there are significant advantages in doing so). Although such a translation overlooks the emphasis placed by Stamper on LEGOL's semantic model, it does suggest an improved way of computing with LEGOL rules which is independent of what these rules might 'mean'. It can be no coincidence that LEGOL, a computer language specifically designed for representing legislation, can be reinterpreted as extended Horn clauses, and in particular that LEGOL made no provision for expressing disjunctive conclusions in its rules.

While there are many legal provisions which can be expressed in extended Horn clause logic, there are also many types of legislation which cannot. Even legislation which could be expressed in extended Horn clause logic is often written in a way that makes such a translation inconvenient or stilted. For example, many legal draftsman adopt a style in which a general rule is expressed in one place, and a number of additional conditions, particularly exceptions to it, are listed separately elsewhere. It is also common to encounter descriptions of rules in terms of other rules. An extract from section 16 of the United Kingdom's Income and Corporation Taxes Act 1970 states:

(2) Where the claimant under subsection (1) is a woman
  (a) the references in that subsection to the claimant's wife shall be construed as references to the claimant's husband, and
  (b) unless she is a married woman living with her husband, for the reference in that subsection to £320 there shall be substituted a reference to £355, and for references to £75 references to £110.

The subsection (1) referred to gives the conditions under which a person could claim a particular allowance against personal Income Tax. Clearly, subsection (2) describes a similar rule which holds only for women, not by stating its conditions explicitly, but by specifying a modification to an earlier rule instead.

These and other devices used by draftsmen indicate that we shall have to incorporate a richer variety of features into our representation language if we are to produce formalizations which keep recognizably close to the wording of the provisions they are supposed to represent. We can go a long way towards providing such a set of language features, however, by introducing a number of straightforward syntactic extensions to the basic extended Horn clause form. To take a simple example, extended Horn clauses do not allow disjunction in the conditions of rules. But a clause with a disjunctive condition

A if *B* and (*C* or *D*)

is logically equivalent to the two clauses

A if *B* and *C*
A if *B* and *D*

This equivalence can be exploited by regarding the more concise clause with the disjunctive condition as syntactic 'sugar' for the two clauses without disjunction. This particular extension is straightforward to implement, and is a feature of most existing PROLOG systems.

There is, of course more to the development of a language for representing legislation than merely providing syntactic extensions to extended Horn clause logic. These syntactic extensions can only help in formalizing legislation which could be translated into extended Horn clause logic anyway, however inconveniently. But there are many kinds of legislation which could not be expressed directly in extended Horn clause logic. We should consider, amongst other things, how we might represent what is sometimes assumed to be the characteristic feature of law, the need to reason about concepts such as obligation ('Thou shalt honour thy father and thy mother') and prohibition ('Thou shalt not kill').

The law is sometimes regarded as a set of norms which state what must or must not be done under particular circumstances. This model of the law leads naturally to consideration of the 'deontic' concepts: obligation, prohibition, permission, and their various elaborations. Such considerations in turn have lead to occasional suggestions that some form of specialized 'deontic' logic is necessary to represent the law and to analyse the processes of legal reasoning.

It should be fairly clear that there are many areas of law where we could not hope to proceed without some way of representing the deontic concepts and computing with them. Specialized deontic logics attempt to provide a general theoretical framework for reasoning about these concepts, and a whole range of deontic logics of various kinds and levels of sophistication have been proposed and investigated. Most of these investigations, however, have not been concerned with computational issues. Efficient and mechanizable proof procedures for non-classical logics in general, and for deontic logics in particular, have still to be discovered. We are unable to choose a suitable deontic logic and expect to construct a computer-based reasoning system around it. An alternative (and arguably more realistic) approach is to attempt a formalization of the deontic concepts within a classical logic. This approach would sacrifice the conciseness of a specialized logic for the greater flexibility offered by describing the concepts explicitly, and it would allow us to exploit immediately the efficient proof procedures which are known for various fragments of classical logic. The treatment of deontic concepts within a computational framework remains an important topic for current investigations, and further discussion is outside the scope of this paper.

We have to recognize that some areas of law, for example those which demand a sophisticated treatment of the deontic concepts, would seem to be beyond the reach of current automated reasoning techniques for the

present. But there are also many areas of law where existing techniques can be applied immediately, and which give rise to an enormous number of potential applications already. There are many legal provisions, particularly those of an 'administrative' nature, which can be regarded as little more than a precise definition of some legal relationship or property. To take a typical example, legislation which describes the circumstances under which an individual becomes a citizen of a country can be viewed as the legal definition of 'citizen'. If this legislation is complex enough, a system which formalizes the definition will have significant practical value. In the next section, a system which formalizes a definition of British citizenship derived from the provisions of the British Nationality Act 1981 is described.

We might argue that citizenship should be viewed as an obligation, on the Government and on those who fall under its authority, to recognize as a citizen any individual who satisfies the citizenship requirements, and to accord him all the rights and privileges associated with that status. This view of citizenship reintroduces obligations, and introduces concepts such as 'authority', 'right' and 'privilege'. If we want to reason at this level of detail there are fundamental philosophical issues to consider and, from a computational point of view, we have the attendant representational problems to contend with again. Nevertheless, the definition of citizenship, in itself, is unlikely to involve such sophisticated concepts. If the definition is complex enough, a system which restricts itself to representing the definition alone would still be of practical value, even though it does not begin to represent what the possession of citizenship actually means.

Deontic logics (whether special-purpose or not) attempt to provide a general theoretical basis for reasoning about the deontic concepts. For many practical purposes, however, it will be unnecessary to import the whole of this general mechanism into every given application. We might argue that entitlement to a Social Security benefit imposes in reality an obligation on the Government to make the appropriate payment whenever a valid claim is made. In practice, we can treat the conditions for entitlement as a definition. For we shall often want to discover whether an individual is entitled to benefit; we shall rarely want to proceed beyond that, to consider what this entitlement means, whether it really does impose an obligation on the Government, or what else we might be able to infer from such an obligation if it does exist.

It can be argued, therefore, that substantial amounts of legislation are, or can be taken to be, essentially definitional in nature. Much legislation can be viewed as a high-level specification of some legal relationship or property. This simplification does not eliminate all the difficulties, however.

An example of what can be encountered in legislation, suggested by

Trevor Bench-Capon, is provided by the regulations under which a woman could receive a 'Housewives Non-Contributory Invalidity Pension' (HNCIP), another of the United Kingdom's Social Security benefits (now defunct). A key provision in the regulations stated that a person would be entitled to HNCIP 'if she is incapable of performing her normal household duties to a substantial extent'. Like many legal provisions, the HNCIP regulations are at the same time definite, ambiguous and vague.

The regulations are definite because they state clearly what their conclusion should be: a woman who satisfies all their various conditions is entitled to the HNCIP benefit.

The regulations are ambiguous because there are two different ways of reading the condition 'incapable of performing... to a substantial extent'. The ambiguity is a syntactic one, caused by imprecision in the scope of the adverbial phrase 'to a substantial extent'. The easiest way of seeing the ambiguity is by bracketing the condition to make the scope of the phrase explicit. The condition could mean

'(incapable of performing...) to a substantial extent'

or it could mean

'incapable of (performing... to a substantial extent).

Problems arise because a person can be capable and incapable of performing a substantial amount of her normal household duties at the same time. A woman who is capable of performing only 40 per cent of her normal household duties is incapable of a substantial amount (60 per cent), and capable of a substantial amount (40 per cent). Under the first interpretation, she is entitled to the benefit. Under the second interpretation, she is excluded from the benefit because she can perform her normal household duties to a substantial extent, and she is therefore not incapable of doing so.

The ambiguity is a genuine one which came to light in the course of appeals against decisions. (Two different interpretations were adopted in two different parts of the United Kingdom. In Northern Ireland it was held that a woman who is capable of performing a substantial amount of her normal household duties is thereby excluded from the benefit. In the rest of the United Kingdom, it was enough to be incapable of a substantial amount for a woman to receive the benefit.)

The syntactic ambiguity in the HNCIP regulations is presumably an unintended one which was overlooked at the drafting stage. The example serves to illustrate an important mechanism of the law. One role of the courts, or whatever other arbitration procedures are established, is to remove accidental ambiguity when the circumstances of a particular case make it necessary (and only then). Eventually a problematic case whose outcome rests on the reading of the ambiguity comes before the

appointed adjudicator, and a decision is taken one way or the other. This decision sets a precedent for the future and establishes one particular interpretation as the 'correct' one to take. This 'debugging' mechanism also operates in organizations outside the law, although it is often very much easier to amend the regulations themselves rather than to rely on case law to specify the accepted interpretation.

The HNCIP regulations are also typical of legislation in that they are vague. Nowhere in the regulations is it specified what the phrases 'substantial extent' or 'normal household duties' are supposed to mean. Unlike the unintended syntactic ambiguity, however, this vagueness is not accidental. The drafter of the regulations communicated precisely what he wanted to say by leaving some of the conditions vague. The general intention of the regulations is clear, although the specific meaning of the conditions will be dependent on individual circumstances. Normal household duties would be different for women who have children and for women who do not, for example: no legislator could hope to foresee every eventuality and make explicit provision for it. A legal draftsman is often reluctant to specify a piece of legislation in the finest detail, preferring that every individual case be judged on its own merits. Moreover, social attitudes shift, and superfluous detail in regulations can make the law unnecessarily inflexible and resistant to change.

In spite of the vagueness in the regulations, most cases of entitlement to HNCIP would be decided straightforwardly in practice. Assuming that the regulations had been suitably disambiguated, it would normally be clear whether a person should be regarded as incapable of performing her normal household duties to a substantial extent or not (at least, this is what the draftsman has assumed). There may be occasions, however, when some doubt remains. There may also be cases in which an appeal is made against a particular decision. In those circumstances, an adjudicating body at the appropriate level of authority will have to intervene. When such a case is brought before it, the adjudicating body will be forced to make a decision: whether, for example, a particular individual is or is not incapable of performing her normal household duties to a substantial extent. This decision sets a precedent. Similar cases in the future will have to be decided in a similar way, either because the precedent is a legally binding one, or because there is a social or moral obligation that persons in the same circumstances be treated in the same way. Through this mechanism, case law emerges which supplements the written regulations, and which gradually closes the definition of the legal concept which the draftsman had chosen to leave unspecified.

In practice, of course, no two cases brought before a court will be identical in every respect. The court may decide that a new case resembles some previous one so closely that the previous decision should apply in the new case also. Alternatively, the court may be persuaded

that the new case, although similar in many respects to previous ones, is nevertheless different enough to require a different decision.

The decisions of a court hold only for those specific individual cases which have been tried. There is (usually) no provision to decide hypothetical cases. It follows that in any new case we cannot say with certainty what its outcome will be, although legislation and case law may provide good guidelines. Until the case has been tried, there is no fact of the matter. Legal concepts are thus often said to exhibit open texture: a legal concept is only precisely defined for those individual cases which have been decided; there is no precise definition for cases which have still to be tried.

Ambiguity, imprecision, and vagueness are always present in legislation and it is the resolution of problems which arise as a consequence that gives legal reasoning much of its own special character. In view of this it is sometimes supposed that symbolic logic has no place in legal reasoning because, it is suggested, logic is a language of such precision that it can only be applied to concepts which are sharply defined. Yet it is exactly the precision of logic which makes it an indispensable tool for analysing and reasoning with law. We cannot avoid ambiguity and vagueness in law, and we must not pretend that these problems do not exist when it comes to constructing computer representations of law. But we cannot even describe ambiguity and vagueness unless we have a language of sufficient precision. Similar arguments for the use of logic are often made by legal scholars. Layman Allen, for example, has advocated the use of symbolic logic for many years, as a practical tool for analysing and potentially simplifying the content of legal documents (see for example [14]).

The stylized form of natural language used by legal draftsmen is a notorious source of imprecision and unintended ambiguity. Recently there have been several attempts to prescribe language standards which will improve the precision of legal documents. Most notably, Layman Allen has proposed the use of a 'normalized' form in which to draft legislation [15]. Not surprisingly, this 'normalized' form is based on a disciplined use of logical connectives, with some suggestions for syntactic conventions to avoid the inadvertent ambiguities to which legal draftsmen are prone. None of this is to argue that draftsmen should be forced, or even encouraged, to eliminate all imprecision from legislation. Many regulations are effective precisely because they are vague and under-specified, and we have seen an example of this in the HNCIP regulations. Rather, it is unintended ambiguities which need to be eliminated from legal documents.

In effect, Allen's normalized form is an attempt to provide the draftsman with a semi-formal language with some degree of precision. Such recommendations are beginning to be adopted in the drafting of

new legislation. As we begin to construct computer representations of the law, we must be aware of these recommendations also. We must eliminate unintended imprecision from our representations of the law, and we cannot possibly do so if the representation language which we use is already imprecise.

The key contention is that legislation can be represented in a mechanizable form of logic, and that extended Horn clauses are a natural form of logic to take for many kinds of simple legislation. In order to test this hypothesis, and to identify candidate features for any syntactically richer representation language, we formalized a sizeable fragment of real legislation, the British Nationality Act 1981. The formalization of the British Nationality Act also tests to what extent the simple execution strategy of PROLOG would be adequate to apply the law (a specific but typical use), and it serves to illustrate why the use of a formal logic is not incompatible with the open texture of legal concepts.

5. LEGAL EXPERT SYSTEM 3

Much of the British Nationality Act 1981 was formalized in extended Horn clauses by a student, Fariba Sadri, in the months of July and August 1983 with no expert legal assistance. A self-contained part of the Act runs in APES on a 128K micro-computer as a program which can apply the law in individual cases. The formalization of the Act is described in more detail in ref. [16].

The British Nationality Act was introduced to provide a new definition of British citizenship in terms of four new categories. The Act itself consists of five Parts and some supplementary Schedules. The first four Parts define the four categories of citizenship. The fifth Part and the Schedules include various definitions which are needed to understand the other four Parts. The first Part of the Act deals with British Citizenship proper: rules for its automatic acquisition (at birth, for example), conditions for entitlement to register or naturalize, and provisions for the renunciation and resumption of citizenship. Each of the next three Parts treats one of the other categories of citizenship (British Dependent Territories Citizenship, British Overseas Citizenship, British Subjects) in a similar way.

At the time it was introduced, the Act was a very controversial piece of legislation and we hoped that its formalization might clarify some of the issues which caused the controversy. More importantly for the experiment, we chose the British Nationality Act because it exhibits all of the characteristics of legislation, while at the same time allowing a number of considerable simplifications. I shall consider these various simplifications in the course of the next few sections. It is enough to remark here that the Act contains so much detail, and its various provisions interact to

such an extent, that even experienced lawyers find it difficult to assess the implications of the Act for particular individuals. Nevertheless, most individual clauses in the Act are easy enough to understand in isolation: useful and non-trivial conclusions can be reached by a straightforward, mechanical application of the rules, so that a formalization of the Act does have some practical value.

The first clause of the Act deals with the acquisition of British Citizenship at birth:

> 1-(1) A person born in the United Kingdom after commencement shall be a British Citizen if at the time of birth his father or mother is
> (a) a British citizen; or
> (b) settled in the United Kingdom.

According to the Act, 'after commencement' means after or on the date on which the Act comes into force. This clause of the Act can obviously be formalized as Horn clauses, although it is less obvious what specific predicates should be chosen. Since elsewhere in the Act it is important to know the date on which an individual acquires citizenship, and the section of the Act by which he does so, we eventually decided on:

> *x* acquires British citizenship on date *y* by section 1.1
> if *x* was born in the UK
> and *x* was born on date *y*
> and *y* is after or on commencement
> and *z* is a parent of *x*
> and (*z* is a British citizen on date *y* by section *y*1
> or
> *z* is settled in the UK on date *y*).

Here I have used the syntactic extension referred to earlier, which allows disjunctions in the conditions of rules. Notice that the formalization makes the assumption, not stated explicitly in the Act, that a person who acquires citizenship by section 1-(1) does so at birth.

The possession of British Citizenship can be related to its acquisition by the rule:

> *x* is a British citizen on date *y* by section *z*
> if *x* is alive on date *y*
> and *x* acquires British citizenship on date *y*1 by section *z*
> and *y* is after or on *y*1
> and not (*x* ceased to be a British citizen on date *y*2
> and *y*2 is between *y*1 and *y*).

A person can cease to be a British citizen by renouncing it or by being deprived of it. The condition

> *x* is alive on date *y*

ensures that a person ceases to be a British citizen when he dies.

Notice, too, that the definition of British citizenship is recursive: whether one person becomes a British citizen by section 1-(1) can depend on the citizenship of another. Elsewhere in the Act, acquisition of citizenship by one section can be blocked by the acquisition of citizenship by some other section (although in general an individual can be a British citizen by more than one section of the Act). It has sometimes been suggested that a propositional logic is adequate for representing legislation. The recursive definition of British citizenship suggests that, in the case of the British Nationality Act at least, a propositional language is not adequate.

Section 2-(1) of the Act states that:

> 2-(1) A person born outside the United Kingdom after commencement shall be a British citizen if at the time of the birth his father or mother
>
> (a) is a British citizen otherwise than by descent; or . . .

Since section 2-(1) is labelled 'Acquisition by descent' in the original text, it was natural to assume that the section could be formalized thus:

> *x* acquires British citizenship on date *y* by section 2.1
> if *x* was born outside the UK
> and *x* was born on date *y*
> and *y* is after or on commencement
> and *z* is a parent of *x*
> and (*z* is a British citizen on date *y* by section *y*1
> and *y*1 is different from 2.1)
> or
> (. . .))

In fact, it turns out that 'British citizen by descent' does not mean British citizen by section 2-(1) ('Acquisition by descent'). 'British citizenship by descent' is defined explicitly, and differently, in section 14 of the Act. The rule for section 2-(1) is corrected by replacing the condition

> (*x* is a British citizen on date *y* by section *y*1
> and *y*1 is different from 2.1)

by the condition

> (*z* is a British citizen on date *y* by section *y*1
> and not *z* is a British citizen by descent on date *y*)

and formalizing

> *z* is a British citizen by descent on date *y*

to reflect the definition in section 14.

We assumed at the beginning of the experiment, rather naïvely with hindsight, that it would be possible to formalize the Act by considering

every individual clause in isolation. This would have meant that different persons could work on the various parts of the Act independently, requiring only that they agree a common vocabulary of predicate names. It turned out that this was impossible in practice. The formalization was forced to proceed by a process of trial and error. Sections encountered later in the Act would often indicate that our formalization of some earlier section was inaccurate or over-simplified. The modification of the rule for section 2-(1) in the light of section 14 is a simple example. Other examples are given in ref. [16]. The need for these revisions raises questions about what is the intelligible unit of law: is it a clause, a section, an Act, or 'the law' in its entirety? This is important if we are ever to say that a formalization represents a fragment of law, since there will be some minimum size of fragment which must be considered. I shall return to this question in later sections of the paper.

In spite of these various complications, sections 1-(1) and 2-(1) of the British Nationality Act translate fairly naturally into extended Horn clause form. Although not all of the Act is quite so straightforward (occasional logical complexities did have to be clarified, some of which are described in ref. [16]), the British Nationality Act is an unusually well written piece of legal prose with relatively few of the linguistic contortions which make so much legislation obscure. Most representation problems arose simply from the scale of the exercise. The Act comprises approximately 70 pages of densely written legal prose. This is big enough to cause severe difficulties in the formalization; it is fairly small in comparison with other areas of law, the Social Security laws of the United Kingdom for example.

The most obvious application of the formalization is to use it as a program which can apply the law in specific cases, to determine whether or not a given individual is a British citizen of some particular category, or to determine whether he is entitled to register or nationalize if it turns out he is not. The easiest way of implementing this is to execute the program in an augmented PROLOG system like APES, so that missing person-specific information is requested from the user, and so that explanatory proofs of conclusions are available if required.

Questions from the system appear in the stylized form which APES uses if no natural language alternative has been specified. I have omitted various complications to make the dialogue easier to follow, and I have included in the program the information that the Act came into force on 1 January 1983. In order to begin a consultation with the system we ask, for example, whether a particular individual is a British citizen by any section of the Act on a given date:

confirm (Peter is a British citizen on date (16 Jan 1984) by section *z*)
Which *X*: Peter was born on date *X*? **(3 May 1983)**

Is it true that Peter died before (16 Jan 1984)? **no**
Is it true that Peter was born in the UK? **yes**
Which X: X is the father of Peter? **William**
Which X: William was born on date X? **(1 March 1952)**
Is it true that William died before (16 Jan 1984)? **no**
Is it true that William was born in the UK? **yes**
Is it true that William was found as a new-born infant abandoned in the UK? **no**
Is it true that William was adopted? **no**
Is it true that William was a citizen of the UK and Colonies on date (31 Dec 1982)? **yes**
Is it true that on date (31 Dec 1982), William had the right of abode in the UK under the Immigration Act 1971 as then in force? **yes**

.
.
.

Is it true that Peter has ever renounced British citizenship? **no**
Is it true that Peter has ever been deprived of British citizenship? **no**

Yes, I can confirm that
Peter is a British citizen on date (16 Jan 1984) by section 1.1 **how**

I deduced
Peter is a British citizen on date (16 Jan 1984) by section 1.1
from the rule

x is a British citizen on date y by section z
if x is alive on date y
and x acquires British citizenship on date $y1$ by section z
and y is after or on $y1$
and not (x ceased to be a British citizen on date $y2$
and $y2$ is between $y1$ and y)

I can show
1. Peter is alive on date (16 Jan 1984)
2. Peter acquires British citizenship on date (3 May 1983) by section 1.1
3. (16 Jan 1984) is after or on (3 May 1983)
4. not (Peter ceased to be a British citizen on date $y2$ and $y2$ is between (3 May 1983) and (16 Jan 1984))

Explain? **2**

I deduced
Peter acquires British citizenship on date (3 May 1983) by section 1.1

from the rule

x acquires British citizenship on date *y* by section 1.1
if *x* was born in the UK
and *x* was born on date *y*
and *y* is after or on commencement
and *z* is a parent of *x*
and (*z* is a British citizen on date *y* by section *y*1
or
z is a settled in the UK on date *y*)

I can show
1. Peter was born in the UK
2. Peter was born on date (3 May 1983)
3. (3 May 1983) is after or on commencement
4. William is a parent of Peter
5. William is a British citizen on date (3 May 1983) by section 11.1

Explain?

In examining a proof such as this, the user has a number of options at every stage. He can choose to examine the proof of any individual subcondition to any required depth. In the more recent implementations of APES, he can in addition request a justification for the particular rule used. This justification would normally take the form of supplementary documentation which explains where the rule comes from, how it relates to the original legislation, and what hidden assumptions have been made in its construction. A justification for the rule representing section 1-(1) of the Act, for example, would include the original text, and should point out the assumption, not explicitly stated in the Act, that a person who acquires British citizenship by section 1-(1) does so at birth. In this way, the user can examine the proof in as much or as little detail as he requires.

The user may also, if he wishes, ask whether an alternative proof exists for the same conclusion. It may be, for example, that Peter is also a British citizen by section 1-(1) because his father was settled in the United Kingdom at the time of his birth. He may be a British citizen by virtue of his mother's citizenship, or for any number of other reasons. Each of the alternative proofs can be examined systematically in turn.

If the citizenship of an individual can not be confirmed, the proof of the negative answer can be examined in similar fashion. Because APES treats negation as failure, proving a negative conclusion involves explaining why every possible way in which citizenship could have been established does not in fact succeed.

The output computed by the British Nationality Act program is not simply the answer 'yes' or 'no', therefore. More important is the set of proofs which can be constructed from the axioms in the formalization and the information supplied by the user. Normally in expert systems, explanations are included to increase the user's confidence in the system's conclusions. In legal expert systems, the ability to generate proofs is more fundamental: the construction of proofs is usually the principal aim of consulting the system. I shall return to the importance of proofs in the next section.

I should remark on the state of the implementation of the British Nationality Act program. At the time the formalization was completed only a small fragment of the Act could be loaded, together with APES, onto the small micro-computers on which micro-PROLOG was then available. In order to demonstrate the system we were forced to identify a self-contained portion of the Act. We chose those sections in Part 1 of the Act which deal with the acquisition of British citizenship proper, and the relevant definitions from Part 5 and the Schedules. We also had to add some general rules (for example, to define parents in terms of fathers and mothers), and some rules for handling calculations on dates. As of December 1983, this system (containing approximately 150 rules) ran in APES on a micro-computer with 128K bytes of memory. We have estimated that a micro-PROLOG system which could address 512K bytes would be sufficient to run the complete Act (containing about 500 rules). Such micro-PROLOG systems have since become available, although at the time of writing we have not transferred the whole formalization to these larger systems.

The secondary objective of the experiment was to test whether PROLOG's simple execution strategy would be adequate for applying the formalization as a program. Since we attempted to keep the formalization as close as possible to the original text, and we were not concerned at all with its efficiency, it is a little surprising that the formalization actually runs remarkably well as a PROLOG program. There are occasional inefficiencies, when PROLOG recomputes something which has already been established earlier in the computation. These inefficiencies could be eliminated straightforwardly by storing selected portions of the computation as 'lemmas'. There are also isolated parts of the Act which cause the program to loop when executed by PROLOG. Such loops, of which there are two in the fragment of the Act which we run, are not problematic and both can be eliminated by program transformation methods.

In the meantime, we have consolidated our experience with the British Nationality Act by applying the same techniques to a number of other examples in the legal domain. Some of these are mentioned in a later section. For now, it is more important to make a number of general remarks about the whole approach.

6. AXIOMATIC MODELS OF LAW

It might be tempting to assume that our formalization of the British Nationality Act is a precise representation of what the legislation tries to express. Arguably, it does represent exactly what the draftsman intended. Nevertheless, the law concerning British citizenship is not what the draftsman tried to express, but what the relevant authorities decide that he did express, and these two things might not coincide. To take a simple example, recall that our formalization assumes that a person who becomes a British citizen by section 1-(1) does so at birth. This is a very reasonable assumption, and it may well be what the draftsman intended. Nevertheless, it is not what he wrote, and it is always possible that a case will arise in the future to cast doubt on this interpretation. If such a case does arise, and if the court decides that an individual became a British citizen by section 1-(1) at some time other than at birth, then our formalization will be an incorrect representation of the law, whether it represents what the draftsman intended or not. It is a key rule in the interpretation of legal documents that the exact wording is critical. It follows that paraphrasing is forbidden. Yet this is precisely what we have done in formalizing the British Nationality Act: we have paraphrased the original text, albeit in a formal language.

The British Nationality Act program is not so very different from Legal Expert System 2 after all. Hammond's system describes Ian Pickup's opinion of what the law requires for entitlement to Supplementary Benefit. The British Nationalty Act program formalizes a particular interpretation, mostly Fariba Sadri's, of the definition of British citizenship as given in the British Nationality Act.

There is a difference between the two systems, however, and it is an important one. We might, for some given application, need to claim that our formalization of the British Nationality Act represents accurately the current state of the law. To substantiate this claim we would have to show the rules in the formalization to an expert lawyer, and even to the draftsman himself. Each individual rule would have to be examined and compared against the original text on which it is based, together with any additional assumptions which were made in its construction. It is because the rules in the formalization are explicit that such an assessment is possible at all; the assessment becomes a practical possibility because the rules are expressed in a form which resembles the structure of the original legislation.

It would be very much harder to assess the accuracy of the model in Legal Expert System 2. We can take it that the model is an accurate one because Ian Pickup is an expert on Supplementary Benefit. Nevertheless, although the rules in Legal Expert System 2 are also stored explicitly, they are far removed from the original sources on which they are based.

It would be difficult or impossible to estimate how well Ian Pickup's interpretation of the law corresponds to the actual legislation. The difference between the two representations is one of granularity. Pickup viewed the law as a whole; Sadri viewed it section by section. The difference is similar to that between a precis and a paraphrase: accuracy is always easier to determine in the latter case.

Given that some assessment of the rules is possible, an assessment of the system's conclusions follows immediately. Both the British Nationality Act program and Legal Expert System 2 are axiomatic systems which attempt to model some fragment of the law. In both cases, answers computed by the program are theorems, logical consequences of the rules in the formalization and the information supplied by the user. It does not follow of course that the conclusions produced by such a system are accurate. It does mean, however, that a conclusion of the system is guaranteed to be accurate, if the axioms in the system are accurate.

This is a fundamental point. It explains why such emphasis has been placed on the examination of proofs and on the importance of documenting the source of the rules in the formalization. Since proofs are guaranteed to be valid, the only remaining doubt concerns the accuracy of the rules and the accuracy of the information which the user has supplied. The proof serves to identify the assumptions on which the conclusion is based; the accuracy of these assumptions can then be examined in detail.

This is in contrast to a model of the law which is expressed in a conventional programming language or in a rule-based programming language with no formal properties. A conventional program does not represent explicitly the assumptions on which it is based. It cannot produce proofs of its conclusions for critical examination, and it is practically impossible to state how the answers it computes relate to the assumptions in its model. A rule-based program does make its assumptions explicit, but unless the computation proceeds by the application of sound rules of inference, there is no sense in which the accuracy of the conclusions can be said to follow from the accuracy of its rules.

We are attempting to construct computer programs which purport to represent legislation in computer intelligible form. These programs are executed mechanically, and it would be of little practical value if the answers they produce could not be said to mean anything precise. A claimant who is advised by computer that he is entitled to Supplementary Benefit must know what this advice is worth: whether it would stand scrutiny in a court of law, for example. There is a real danger that the advice of computer-based legal systems will be overestimated. We can guard against misunderstandings only by explaining to users what the computer-generated advice means in reality, and this explanation cannot be in terms of detailed computations which the program invokes.

It is sometimes assumed that the test of a legal expert system is how well it can predict judicial decisions in its own particular domain. To suggest this is to miss the point entirely. For it must be recognized that legislation is often imprecise, and that it is essentially open textured. The role of the judiciary is to resolve questions of law as they arise. And the resolution of these questions is fundamentally a matter of choice. A judge will decide one way or the other, not because there is a right answer, but because he is forced to make a decision, and because he will find the arguments presented for one decision more persuasive than the arguments for another. This does not mean that we can never predict judicial decisions. The majority of cases will be decided straightforwardly (for otherwise that fragment of law would be so intolerably inefficient that it could not survive long its present form). If a legal expert system could not anticipate decisions in the routine cases, we would be right to dismiss it. But decisions in cases which are not routine can not be predicted, because it is impossible to anticipate all the arguments which could be presented, and because we cannot be sure which arguments the judge will prefer (that is what stops a case being decided routinely). If we accept that the law is open textured, then there really is no right answer until the judge has pronounced his verdict.

This is not to say that a legal expert system which is capable of predicting judicial decisions would not be an extremely useful tool in legal practice, or even that it would be impossible to build. But such a system would have to manipulate statistics and likelihood estimates of various kinds, and it is not the type of system which is being considered in this paper. It would resemble rather the imaginary Legal Expert System 1 which was described in this paper by way of introduction. This paper is concentrating instead on how a set of rules (axioms) might be constructed to model some fragment of the law (or more precisely, given the essentially indeterminate nature of the law, some particular interpretation of the law).

The ability to generate proofs takes on additional practical significance if we consider the nature of the computer systems we are attempting to build. In the first instance we might want to construct a system which is intended to take legal decisions. Although we would be reluctant to have the British Nationality Act program decide whether or not individuals have British citizenship, and although we would not want Legal Expert System 2 to process routinely the claims which are made for Supplementary Benefit, it would be perfectly acceptable to have many para-legal decisions taken mechanically by machine in applications outside the law. A company might want to process by machine the expense claims of its employees, or their requests to take leave, or any number of other similar day-to-day decisions. Indeed, the vast majority of 'legal' decisions taken every day are of precisely this trivial (though not necessarily

straightforward) kind. We are most of us willing to accept the legal decisions taken by the payroll system which calculates our salaries, and the tax and other deductions which it makes, even though it is normally impossible to question the system about the assumptions on which these calculations are based.

When decision-taking systems are able to generate proofs they are able to account for their decisions, and they become applicable in less trivial decision-making tasks. At the very least a proof will demonstrate that any decision reached was not an irrational one. A proof will also form the basis for an appeal should any particular decision be unpopular or considered unfair for some reason. Moreover, there are bound to be decisions which will be unacceptable for any number of reasons, whether we apply the law by computer or not. The examination of proofs for such decisions would identify areas where the existing legislation could be revised; it may even suggest ways of implementing these revisions so that unwanted decisions are less likely to occur in the future.

For most applications within the law, however, it will never be acceptable to have decisions taken routinely by machine, whether such a machine can explain its reasoning or not. Legal expert systems are more widely applicable if they can be used, not as legal decision takers, but as legal decision-taking aids. In this kind of system the construction of proofs is more than a convenient by-product: in many applications of legal systems it will be the proofs and not the conclusions which will be of primary interest. An individual claiming for some additional benefit will want to know what reasons he can adduce to show that he is entitled; an officer writing a submission to an appeal tribunal will want to know which rules he can cite to support the decision that he made; and an advocate who is preparing his client's case will want to anticipate the opposing arguments he is likely to encounter.

The proofs which are generated as a result of consulting the system will provide a framework for a legal argument, although this framework will have to be developed further before we have a convincing legal argument. A proof starts from some assumptions or premisses and moves by rules of inference to a conclusion. If we accept the assumptions then we cannot deny the conclusion. However, we can properly refuse to accept a proof by casting doubt on the assumptions on which it is based. The assumptions which appear in our system-generated proofs are either rules from the formalization itself, or information which is supplied by the user in response to questions from the system. Let us consider in turn how doubt might arise over these two types of assumptions.

In the first instance we might have included rules in the formalization which are inaccurate because we have misread or misunderstood completely the provisions of the legislation. More likely (and more difficult to detect) is the possibility that we have not taken into account the effect of

(meta-level) rules of interpretation. Most legal systems prescribe rules which govern the interpretation of legal documents, and which constrain how legal documents are to be read. We could presumably guard against this type of inaccuracy by involving an expert lawyer in the formalization process, to advise on matters of interpretation and to identify the laws of interpretation as they apply. Even more problematic, especially if we adopt the computational expedient of treating negation as failure, is the very real possibility of overlooking some legal provision which is not mentioned explicitly in the fragment we are formalizing but which is considered as part of the same law nevertheless. (The British Nationality Act 1981, for example, amends various provisions in the Immigration Act 1971, although we would have no way of knowing this from reading the text of the Immigration Act alone.) The law concerning some particular matter is usually derived from a great variety of sources and we must be aware of the dangers of treating an individual fragment in isolation. Such considerations again raise the problem of identifying an intelligible unit of law which I alluded to when discussing our formalization of the British Nationality Act. Just as we could not treat an individual section of that Act independently of the rest, we could not safely represent any one Act, or any other fragment of the law, in isolation. Yet we cannot realistically expect to formalize the whole of 'the law'. We could only hope to eliminate these problems, or at least not to overlook them, by taking expert legal advice as the formalization proceeds.

A more interesting source of inaccuracy perhaps is the case of legislation which is ambiguous, particularly since this was identified earlier as a characteristic of all legislation.

7. AMBIGUITY

When we began the formalization of the British Nationality Act, we expected that the process of formalization itself would identify cases of imprecision and ambiguity which had been overlooked at the drafting stage. I have already mentioned an example of imprecision in the British Nationality Act: the Act does not state clearly the time at which an individual becomes a British citizen by section 1-(1) of the Act. If the formalization is to run as a program then we have to make some additional assumptions. We decided to assume that such an individual would acquire citizenship at birth. It could be argued, however, that citizenship imposes duties as well as granting rights; that duties cannot be imposed on a minor; and therefore that an individual acquires citizenship by section 1-(1), not at birth, but on the day he reaches full age. This might be a tenable argument (in fact, it is easily defeated by considering other sections of the Act), and, if it is, we have two distinct interpretations of the Act. If we cannot choose between them then we really have

no choice: we must incorporate both interpretations of the Act together, or risk giving a distorted view of the law. We have two different, similar, but distinct, formalizations of the law. Now if we ask whether a given individual is a British citizen on a given date we may receive two different and conflicting answers: 'yes' by one interpretation (and an associated proof); 'no' by the other interpretation (and its associated proof). All else being equal, the proofs will differ where the rules for section 1-(1) of the Act appear. We are left with choosing whether we prefer one interpretation of the law to the other. If we genuinely cannot decide, we must wait for a suitable case to come before the appropriate adjudicating authorities, for they will be forced to make a decision which in turn will help us decide which of the conflicting interpretations to adopt. Of course, if an individual's citizenship is not determined by the application of section 1-(1), then the two formalizations will agree (although in general there will be other formalizations which disagree about the reading of some other section or some other ambiguous phrase in the Act).

Where there is genuine and obvious ambiguity in legislation, we have to formalize both possible interpretations or risk misrepresenting the law. Once a case has been tried and a suitable disambiguating precedent has been set, it may be possible to discard one or more of the formalizations. Until this happens we are left with several, potentially inconsistent, versions of the law. There remains the technical problem of maintaining large formalizations which are distinct but which contain an immense amount of duplication. I do not propose to discuss such (implementational) details in this paper.

It may happen of course that a piece of legislation is ambiguous, without this being at all obvious. The Housewive's Invalidity Pension (HNCIP) regulations which were mentioned earlier are an example. It is not clear on first reading that the regulations are ambiguous. Indeed, even though the ambiguity is known to exist (two different interpretations were adopted in practice), it is not easy to demonstrate where the ambiguity lies. The process of formalization does not in itself identify ambiguity or imprecision. The person who is responsible for constructing the formalization must detect the ambiguity before deciding how it should be treated. A representation language which is precise enough may allow him to describe as many different, unambiguous interpretations as he sees fit, but it does not in itself make the detection of the ambiguity automatic. It follows that we can expect cases to arise which will identify an ambiguity which was overlooked in the formalization. When such cases come to court, and as they are decided, we will be forced to amend our formalization of the law to reflect these decisions.

This raises a question concerning the formalization of regulations which quite obviously do not express what the draftsman intended. Bill Sharpe's account [17] of his attempt to formalize the United Kingdom's

Statutory Sick Pay legislation provides such an example. The Sick Pay regulations refer repeatedly to 'periods of incapacity for work' which are defined, in essence, as 'periods of four or more consecutive days, each of which is a day of incapacity for work'. Quoting now directly from Sharpe,

> we realize that . . . the rule does not say what it means. It is quite clear from the use made of the definition that a period of incapacity for work is not just a consecutive period of sickness but the longest such period. It begins when someone falls sick and ends when they are better; a subset of a period is not properly speaking a period in the sense meant there.

It may sometimes be impossible to devise a formalization which is at the same time an accurate model of what the draftsman intended and a faithful representation of what he actually wrote. I shall have a little more to say about Sharpe's example later in the paper.

8. VAGUENESS AND CASE LAW

Leaving aside inaccuracy in the formalization (although more could be said) let us consider now the accuracy of information supplied by the user. If the formalization is executed by a system such as APES, then, roughly speaking, the user will be required to answer questions about concepts which are mentioned in the legislation but which have not been defined explicitly by the draftsman. We can distinguish immediately two different types. There will be concepts which are undefined but which have a precise legal meaning nevertheless: although they are not defined in the fragment of legislation we have formalized, they are defined explicitly elsewhere. And there will be concepts which are not defined explicitly anywhere in the law, and which have no precise legal meaning.

Let us dismiss the less interesting type, the concepts which are defined precisely elsewhere. In the case of the British Nationality Act, there are many references to concepts from the Immigration Act 1971 and to concepts from previous Nationality Acts superseded by the 1981 Act. Presumably when the draftsman of the 1981 Act refers to 'a person who immediately before commencement was a citizen of the United Kingdom and Colonies', he means something very precise (although we would need to read the earlier Nationality Acts to find out exactly what it is). In formalizing the provisions of the 1981 Act there are several ways of proceeding. In addition to formalizing the 1981 Act, we can formalize the provisions of the earlier Acts too, to define precisely what it means to be a 'citizen of the United Kingdom and Colonies'. If this turns out to be impractical, we can adopt the technique of Legal Expert System 2 and build a definition of 'citizen of the United Kingdom and Colonies' from the opinion of some appropriate expert (although then we would get a

system of less 'authority'). And if this is impractical too, we can always choose to leave the concept undefined, and treat it as if it were a concept of the second type, one with no precise meaning.

The more interesting concepts, perhaps, are those which are not defined explicitly anywhere in law. Into this category fall concepts which have a common-world meaning and which need no special legal definition (in the British Nationality Act, what it means to be 'born' for example), and vague concepts which have been left unspecified by the draftsman on purpose (like 'normal household duties' in the HNCIP regulations I mentioned earlier). The distinction is an artificial one of course, if we accept that the law is incurably open textured. With a little imagination, it is easy to see how a case could arise which would throw considerable doubt onto the meaning of 'born' (or, more likely, onto the meaning of phrases like 'the time of birth' or 'the place of birth'). Conversely, we shall sometimes be able to say with great confidence what 'normal household duties' are for a given individual. Nevertheless, we can take a pragmatic point of view, and distinguish informally what we might call 'concrete' data (like a person's date of birth) from concepts which are inherently vague (like 'normal household duties'). In treating these concepts we shall have to take into account the existence of any relevant case law, for this will constrain the meaning of the vague phrases. And we must always remember that a case in the future might force us to revise our classification of which concepts we consider 'concrete'.

So far we have been treating concepts which are not defined in the legislation by omitting them from the formalization altogether and by asking the user to supply information about them as it is required. But there is an alternative. There is a property which holds for any logical system (we might even take it to be the definition of a logical system) that

> when conclusion '*P*'
> follows logically from assumptions $\{A1, \ldots, An, B\}$
> then conclusion '*P* if *B*'
> follows logically from assumptions $\{A1, \ldots, An\}$.

If we are trying to derive conclusion '*P*' from assumptions $\{A1, \ldots, An\}$ (which are rules representing some piece of legislation) and cannot continue because some concept '*B*' appears in the conditions of a rule but is not defined, then we can add '*B*' to the assumptions $\{A1, \ldots, An\}$ and continue. If the computation now succeeds (that is to say if we manage to prove '*P*' from the augmented set of assumptions) then we will have proved '*P* if *B*' from the original assumptions $\{A1, \ldots, An\}$. In other words, we are able to report an answer which is qualified with this extra condition '*B*'. To take an example, instead of reporting 'Yes, Peter is a British Citizen', the system might respond with the qualified answer 'Yes, Peter is a British Citizen, if (it can be established that) he was a

citizen of the United Kingdom and Colonies on 1 October 1982.' Let us call such a qualifying condition a 'tentative assumption'. It should be clear that we can qualify answers with as many tentative assumptions as we like.

Computation with tentative assumptions is not quite as straightforward as I have made out, particularly in the presence of negation as failure. There are some similarities to an extension to PROLOG proposed by Gabbay and Reyle [18] which they called 'hypothetical implication'. More detail is beyond the scope of this paper, but let us suppose that computation with tentative assumptions is available, because it gives us enormous flexibility.

In the first place we might try to implement a system by treating all undefined concepts as tentative assumptions. This does not give a system of much practical value unfortunately. For we would get from the system little more than a paraphrase of the original formalization, although without the original structure. (Consider what happens if we ask 'Is Peter a British citizen' from the formalization of the British Nationality Act. All we get in reply is the answer 'Yes', qualified with all the conditions which must be satisfied before Peter's citizenship can be established.)

Computation with tentative assumptions works very much better in conjunction with querying the user. In the case of the British Nationality Act, for example, it would be reasonable to ask users about the 'concrete' undefined concepts (a person's date and place of birth, the identity of his parents, and so on). The availability of this concrete information means that fewer sections of the Act will apply in a particular case. The system will produce answers which are more specific and which, if qualified, are qualified by low-level, detailed tentative assumptions.

Tentative assumptions can also be exploited to improve the treatment of queries to the user. We can allow users the option of refusing to answer a question which they do not understand or to which they do not know the answer, since the treatment of these 'don't know' answers clearly reduces to adding some appropriate tentative assumption. Once we allow users the possibility of answering questions with 'don't know', we can go further, and we can reasonably revert to the scheme in which users are asked to supply information about all undefined concepts, the vague as well as the concrete. For a user will sometimes be able to decide whether a vague concept applies in the particular case under consideration. If he cannot, there is always the option of answering 'don't know'.

I am not recommending any one of these possible treatments as the right one to adopt in practice. Which concepts to treat as tentative assumptions and which to treat by querying the user will depend on the particular application, and will probably reflect personal taste. The British Nationality Act program (or at least that fragment of it which we

have been running on the small micro-computers) works reasonably enough by querying the user about all undefined concepts.

Whichever method we choose for treating undefined concepts, we must still help users to discover what these concepts mean. If the user is expected to answer questions in the course of the consultation then this help will have to be available when the question is asked. If we are using the mechanism of tentative assumptions then help can be delayed until after the consultation is complete. In either case, we have to give the user some indication of what the concept is taken to mean in practice, especially if there is existing case law which must be brought to his attention.

The help we provide can take a variety of forms, and the appropriate choice will depend on the nature of the application we are attempting to build. In the simplest cases, it might be enough to accompany every question to the user with a fragment of text which is intended to describe the concept in fairly general terms, and to summarize any relevant case law which might exist. Incidentally, this suggestion also allows me to make the point that it is not necessary to formalize every detail of written legislation. Section 50 of the British Nationality Act includes the information that 'ship includes a hovercraft'. It would be more sensible to include this as a footnote to questions about ships instead of going to the extreme lengths of including such information in the formalization. Similarly, one of the key conditions for acquiring British citizenship is that a person be 'settled in the United Kingdom'. Section 50 of the Act defines this concept as 'ordinarily resident in the United Kingdom without being subject under the immigration laws to any restriction on the period for which he may remain' (with some minor exceptions). Since 'ordinarily resident' is not defined further in the Act, there may be little point in formalizing the definition of 'settled in the United Kingdom'. Doing so would only replace questions about 'settled' with questions about 'ordinarily resident'. (Although it may be that 'ordinarily resident' is one of these concepts which are defined elsewhere in English law.)

Attaching explanatory text to questions will only provide adequate help in the simplest of applications. In general we shall have to give more detailed help. One possibility is to allow the user access to the extensive legal document retrieval systems which are now becoming increasingly important in everyday legal practice. We could also seek to advise on matters of case law by allowing the user to consult a data base of decisions which were taken in previous cases. We could construct a supplementary expert system (constructed by any suitable method) which would allow users to invoke a supplementary consultation whose sole aim is to help with determining what some vague concept has been held to mean in the past. Trevor Bench-Capon and I have proposed elsewhere [19] that vagueness in law could be captured by constructing a system of

conflicting rules, arranged to generate arguments for and against a particular conclusion. These various methods of providing help are not mutually exclusive, of course.

In general, a system which represents some piece of written legislation will have a component which is a formalization of the written legislation, and a component which is designed to help with deciding the lowest level concepts whose definitions have been omitted from the formalization. It is critical that these components should be kept as distinct and separate as possible. For otherwise, by coupling the two components together, we get a system whose conclusions are based on one single set of assumptions. It becomes difficult then to disentangle those assumptions which are fairly certain (because they are derived from written legislation) from those whose accuracy is practically impossible to assess (since they express implicitly stated rules of law derived from fragments of case law). We must recognize that in questions about open-textured concepts there is no fact of the matter. Often, the best way of seeing why a concept is vague and discovering what kind of criteria are important for establishing its truth is by executing parts of the system several times over and varying the assumptions for each consultation. We can more readily encourage users to experiment in this way if we keep separate the different components in the system. There is a great deal of flexibility in all of this. We can allow supplementary experiments in the course of the main consultation. We can leave experimentation to the end of the main consultation, until we have an answer with its various qualifications. Or we can let the user decide which of these options he prefers.

A final remark is in order. If we are interested in building a system which is to take legal decisions autonomously (and I argued above that we might) then we *must* couple a formalization of written legislation with rules which define precisely how vague concepts are to be treated in practice. There are some applications where this could be done. But if we can write rules to eliminate vague concepts, then so could a draftsman. It is because we lose flexibility that autonomous and automatic decision takers are necessarily limited in scope.

It should be clear why our formalization of the British Nationality Act works reasonably well, both as a program for applying the law, and as a system for helping lawyers to solve legal problems related to British citizenship. The provisions of the Act are complex and interact to a great extent; the Act contains relatively few references to other legislation; and although vague phrases have been used by the draftsman, they are at the lowest level of detail and they only have to be taken into account once the 'concrete' items of data have been determined for a given individual. Not all legislation is like this however. If we take instead some fragment of legislation where the rules are very shallow, or where there are very few concepts which can be readily decided, then formalizing the

legislation will give us very little help. We will be more exposed to the problems of vagueness and open texture, and we will have to rely to a greater extent on the supplementary advice giving systems which I sketched out above.

9. GOAL-DIRECTED FORMALIZATION

I have stressed several times that our formalization of the British Nationality Act should be regarded as an axiomatic system, and I have claimed that this is where all of its power is derived. However, the process by which the formalization was developed contrasts directly with the normal method of constructing an axiomatic system. Usually in mathematics an axiomatic system is constructed from the bottom up, by choosing first a set of primitive, undefined concepts, and then devising axioms which define new concepts in terms of those which are primitive or already defined.

This bottom-up approach is also found in attempts to represent legislation as computer programs, most notably in Ronald Stamper's work with the LEGOL project [11, 12], mentioned earlier. In LEGOL, representation of some piece of legislation proceeded in two distinct steps: first an analysis based on LEGOL's 'semantic model' attempted to identify the various entities, concepts and relationships which appear in the legislation; then LEGOL rules were written to simulate the effects of the legislation on the concepts identified in the analysis phase. Although a LEGOL program could not be described as an axiomatic system, the method of construction did correspond closely to the bottom-up construction of axiomatic theories in mathematics.

There are severe practical problems in attempting a bottom-up formalization of any realistically sized piece of legislation, however. For how are we to choose the bottom level primitives in the first place? In the case of the British Nationality Act, for example, we would be forced to decide at the outset whether a concept such as 'the Secretary of State's discretion', which occurs several times in the Act, is a primitive concept or one which should be defined in terms of something more primitive instead. And there are many other such phrases which appear in the Act.

I should stress that this is precisely the problem which the LEGOL project was attempting to address. Its primary objective was to develop a methodology for identifying the relevant concepts in a piece of legislation; the implementation of practical systems was a secondary objective. It might be argued that such an ambition is not a realistic one, but leaving that aside, it is certainly the case that no such complete methodology exists at present. We must consequently look to some other approach if we are to construct a formalization of some fragment of the law in practice.

The obvious alternative, and the way in which the formalization of the

British Nationality Act was developed, is to construct the formalization in a goal-directed, top-down manner instead. Thus, if our aim is to formalize how British citizenship is acquired, there is a natural place to begin. The conditions for acquiring British citizenship are given explicitly in the Act: formalizing them introduces lower level concepts which may or may not be defined elsewhere in the Act. We proceed accordingly, formalizing those concepts which are defined in the Act, and leaving out of the formalization those that are not (although more about this later). At every stage we have an executable version of the formalization, since top-level concepts are defined and systems like APES can query the user for information which is missing. More importantly, we only need to address the problem of representing a concept like 'the Secretary of State's discretion' when we know in what sense it is relevant to the top-most goals we are considering.

To see why this is important, note that the British Nationality Act does more than specify the circumstances under which an individual becomes a British citizen. From the Secretary of State's point of view, for example, the Act describes some circumstances under which he is permitted to treat one individual differently from another. If we are interested in representing this aspect of the Act also, the formalization will need to treat the Secretary of State's discretion in some detail, for the Secretary of State will be interested in establishing whether he is permitted to exercise his discretion in a particular case, and he will be interested in the conditions under which this permission is granted. For the purpose of establishing some given individual's British citizenship, however, this kind of detail is superfluous. If we want to establish an individual's citizenship, the Secretary of State's discretion can be treated as if it were a primitive concept (either he does exercise his discretion or he does not). The representation of discretion in all its generality is difficult. Top-down, goal-directed, formalization removes the need to address difficult problems of representation unnecessarily, until it becomes unavoidable to do so. This is a practical point. I am not suggesting that the representation of discretion, or the representation of any other general legal concept, is not a topic worth investigating in its own right.

Legislation is seldom written with a particular goal or method of application in mind, and we might argue therefore that its computer representation should not be tailored to a particular usage either. To some extent, a formalization in logic reflects this requirement. The British Nationality Act program is not limited to establishing the citizenship of a given individual. Any number of different queries will invoke a consultation of the system, and in this respect it resembles a deductive data base more than it does a traditional expert system (although if we are to use it as a deductive component to a data base we shall have PROLOG's limited execution strategy to contend with). And we

could take the same formalization, make explicit the 'only if' rules we chose to treat implicitly in our program, and employ more powerful theorem provers to exploit the formalization for any number of different purposes.

Nevertheless, it is quite unrealistic to imagine that we can devise a formalization of some part of the law without taking any account of how we shall use the formalization once it is constructed. A single piece of legislation can be used for many different purposes: even to read its provisions we need some idea of what we want to know and the kinds of things we want to discover. In the same way, the formalization will be influenced by what we expect to do with the rules and the kinds of inferences we shall want to support, although we might have several goals in mind, and there might be several aspects of the legislation that we want to represent simultaneously. Written legislation is expressed in natural language which assumes (reasonably enough) an enormous amount of common-sense knowledge before any sense of it can be made at all. At the very least we shall need to ground the formalization in some minimal representation of the relevant common-sense knowledge. It seems inevitable that we shall have to make some assumption about intended usage in doing so. After all, we would not consider representing in computer-intelligible form some arbitrary piece of natural language text without having some idea of what this representation is for. Bill Sharpe makes a similar point, although in a slightly different way, in his account of the Statutory Sick Pay formalization [17] which I mentioned earlier. Sharpe found he could not represent the Sick Pay regulations without considering to some extent how his program would be used, and the kind of common-sense knowledge (he called it 'problem solving rules') which this would require.

In formalizing a piece of legislation, we shall sooner or later have to choose predicates, and we shall have to represent the legislation in terms of these predicates. In advocating top-down, goal-directed formalization, all I am suggesting is that we should not choose the bottom-level predicates at the outset, partly because we cannot know at that stage which common-world concepts will turn out to be relevant, and partly because we cannot realistically expect to represent the bottom-level concepts (common-world or not) without some idea of purpose.

The penalty for this goal-directed approach is the phenomenon of what Thorne McCarty has called 'the hyphenated predicate'. Thus in our formalization of the British Nationality Act there appear such monolithic 'primitive' predicates as 'had the right of abode in the United Kingdom under the Immigration Act 1971 as then in force', or 'was found abandoned in the United Kingdom as a new-born infant'. In his own work on the TAXMAN project (see for example ref. [20]), McCarty has stressed the need for some 'deep' underlying conceptual model without

which, he argues, it is impossible to reason more than superficially about the law. This opinion is surely right, but it does not conflict at all with the goal-directed approach which I am advocating for the formalization of written legislation.

McCarty's TAXMAN work has been primarily concerned with representing and reasoning with case law. The TAXMAN experiments are based on cases covering various aspects of Corporate Tax Law in the United States, in which the central concepts are Corporations and Shareholders, stocks and shares, events and transactions, and, to some extent, permission and obligation too. A major concern of the TAXMAN work has therefore been the invention of an appropriate representation for these concepts, for without one it would be impossible to reason about these cases in any meaningful way. If we consider now the British Nationality Act, we see that high-level concepts like the acquisition of British citizenship are defined in terms of primitive concepts such as the time and place of a person's birth and the identity of his parents. It is these primitive concepts which form the underlying conceptual model for our formalization of the British Nationality Act (although it is such a trivial model that it hardly deserves the name). Between these two extremes, the TAXMAN programs with their rich conceptual model and the British Nationality Act program with its simple model, we might place Sharpe's formalization of Statutory Sick Pay legislation. The primitive concepts underlying the Sick Pay regulations are periods of sickness and incapacity for work. Although Sharpe did not have to represent the concept of 'sickness', he did have to consider how to represent periods of sickness. It seems almost superfluous to add that these three conceptual models, for the TAXMAN programs, for our British Nationality Act program, and for the Statutory Sick Pay program, have no primitive concepts in common.

How is it then that our formalization of the British Nationality Act contains primitive bottom-level predicates like 'was found abandoned in the United Kingdom as a new-born infant', for these are surely not part of our underlying conceptual model? We can distinguish, however, concepts which involve points of law from concepts which involve only points of fact. In our formalization, primitive 'hyphenated' predicates are used to represent facts which are certainly complex but which are devoid of any points of law. 'Hyphenated' predicates whose application depends on points of law are broken down further in terms of more primitive concepts. In a sense, the draftsman of the British Nationality Act forced us to introduce the predicate 'was found abandoned in the United Kingdom as a new-born infant' (or something like it) by introducing the notion in the wording of the Act. We are less likely to need such predicates when reasoning with case law because we would have no reason for introducing the corresponding concepts into our conceptual model.

Some 'hyphenated' predicates which involve only points of fact will sometimes have to be broken down to make their application more readily apparent. It must be recognized, however, that being found abandoned as a new-born infant is hardly a major consideration for establishing the British citizenship of a given individual. There are many such concepts in the British Nationality Act: they are not central to understanding the provisions of the Act, although they have to be considered eventually when we attempt to apply the Act in some specific instance. We do need a conceptual model if we are to represent some piece of law, that much is clear. But we only need a model which treats in detail the concepts which are central to understanding the particular fragment of law we are considering. It is unnecessary to insist that the model should be rich enough to treat all the peripheral concepts, too. We must be realistic, moreover. A deep, conceptual representation of what it means to be found abandoned as a new-born infant is staggeringly difficult, perhaps even impossible, and there are numerous similar concepts which appear in the British Nationality Act. It is impractical, and unnecessary, to insist on a conceptual model which is sophisticated enough to handle them all.

Of course there is more to constructing a conceptual model than simply compiling a list of suitable predicate names. We must capture also the various relationships and constraints which obtain between our primitive concepts (that fathers are always male, for example). If we are reasoning with representations of case law then it is these various constraints which will be of primary importance. In systems which represent written legislation, we shall be more concerned with rules which define the high-level concepts, but we cannot neglect the primitive concepts entirely. At the very least, the absence of a suitably rich conceptual model will lead to stilted dialogues; and there is always the danger of generating questions which will undermine the user's confidence in the system (when, for example, the system asks whether John, who is known to be male, is the mother of Mary). How to make use of constraints to eliminate such questions is outside the scope of this paper.

The concept of being found abandoned in the United Kingdom as a new-born infant can be treated superficially in our formalization of the British Nationality Act, not only because it is not central to understanding and reasoning about the conditions for acquiring British citizenship, but also because the concept has no special meaning in law other than the way we would naturally understand it as laymen (as far as I know). The application of the corresponding 'hyphenated' predicate does not turn on a point of law. Suppose however that a case should arise in the future concerning a child who was found in the street somewhere in the United Kingdom. Suppose moreover that there is considerable doubt whether this child should be regarded as a new-born infant, and whether or not it

was actually found 'abandoned'. To decide the case, a court at some appropriate level of authority will eventually pronounce its verdict, and as soon as it does the whole situation will change. For it will no longer be the case that 'being found abandoned as a new-born infant' has only its common-sense meaning. The concept will have acquired a specific legal meaning too, constrained by the precedent which the court would have set. In these circumstances, we would be right to attempt a representation of what 'found abandoned as a new-born infant' means, and the underlying conceptual model would reflect the considerations which the court took into account when reaching its decision (the child's apparent age amongst other things perhaps). A similar situation, although a much less likely one, would arise if the Government ever decided to pass new legislation for the care of abandoned infants. A concept such as 'being found abandoned as a new-born infant', peripheral to understanding the British Nationality Act, would become central for the purposes of formalizing this piece of new legislation. But many of the problems with representing this concept would also have disappeared. In drafting the new provisions, the legislators would presumably have spelled out a precise legal meaning for this concept. This definition we could reasonably hope to formalize with some degree of confidence.

We must beware of assuming that a conceptual model which has allowed us to formalize one piece of legislation will allow us to formalize some other fragment also. Nevertheless, there may be some concepts which are fundamental to understanding the law and which are common to many types of legislation. Permission, obligation and their various elaborations are obvious candidates. Detailed investigation of such concepts and their representation in a computational framework are active areas of current research—for a recent (model-theoretic) treatment see McCarty's proposal [21].

In these last sections I have dwelt on some of the problems involved in representing legislation as computer programs. But we must not assume that these problems will always arise. There are many applications of legal systems where ambiguity, vagueness, and other types of indeterminacy in the law simply do not arise or where they can be discounted for all practical purposes. This is particularly true for applications in organizations outside the law. In the next section a brief account of a simple example of this kind is given.

10. LEGAL EXPERT SYSTEM 4

Following our formalization of the British Nationality Act, the same techniques have been applied at Imperial College to a number of other examples from the legal domain, mostly as student projects. Those which incorporate realistically sized fragments of legislation include a subset of

the Immigration Act 1971 [22], regulations for Government grants to industry [23], and a large company's pension scheme with associated tax legislation [24].

The last example, dealing with pension regulations, is interesting because it incorporates an explicit representation of legislation within a system for decision support. It also illustrates the kind of application which can be implemented with relatively little effort.

The project was undertaken by David Chan as part of his M.Sc. course in the Department of Computing at Imperial College. The problem was supplied by a large chemical manufacturing company.

The company had recently introduced a scheme in which their employees could make an additional voluntary contribution (AVC) to the company's pension fund with a view to increasing their income after retirement. A problem arises because the Government in the United Kingdom imposes a limit on the total pension an individual can receive. This limit is determined for an individual by the salary received in the last few years before retirement, and by economic factors such as the rate of inflation prevalent at the time of retirement. Any additional voluntary contributions which raise an individual's pension above this limit are lost and cannot be refunded retrospectively.

Word had spread within the company that the AVC scheme was an attractive one, although in fact most of the company's employees were in such a position that AVC payments could not benefit them. Nevertheless, there were individuals who could benefit substantially from making some appropriate contributions under the scheme. The company was naturally concerned that those employees nearing retirement should receive the best possible advice concerning AVC. Local personnel officers seldom felt competent to give this kind of advice, however, so that the majority of queries were being referred to the AVC specialist in the company's Pensions Department. The classic need for an expert system arose as a consequence. AVC advice requires very specialized knowledge, yet the company AVC expert was so overwhelmed by requests that he was finding it increasingly difficult to cope.

At first Chan did consider implementing a classical expert system as a solution to the problem. He interviewed the specialist several times, and managed to formulate a number of suitable rules as a result. However, AVC advice is extremely difficult to give. It involves predicting, amongst other things, the likely date of retirement, the expected salary at retirement, the prospects for promotion or for exceptional bonuses in the meantime, and an estimate of what the rate of inflation will be like at the time of retirement, five or ten years in the future.

The expert was, naturally enough, very reluctant to propose possible rules. This reluctance was due partly to the difficulty of making the necessary predictions, but also because the expert was still himself in the

process of building up expertise. Moreover, both the company and the expert were particularly anxious that no decision should ever be made on behalf of an employee. The decision, they insisted, must remain with the individual; the role of the expert was to explain the scheme as well as he could, and to help the client assess for himself the possible benefits of joining the scheme.

For these reasons, Chan rejected the classical expert system solution, and decided instead to base a system directly on a representation of the relevant regulations. This involved formalizing the company's internal pension regulations, the additional AVC regulations, and the relevant fragments of income tax legislation. Supplied with an individual employee's data, and an estimate for the rate of inflation and whatever other factors are relevant, the system calculates the total pension received under these assumptions. The employee is now encouraged to experiment with the various parameters. He can vary the date of retirement, his salary at that time, his monthly AVC contributions, the rate of inflation, and so on, until he acquires sufficient understanding of the scheme and how it relates to his own circumstances to make an informed decision. Explanations (proofs) generated by the system in support of its calculations also help in conveying to the employee what are the critical conditions for him.

Chan's system illustrates many of the points that have been made in this paper. The regulations on which the system is based are certainly complicated (the company has several separate pension schemes, for example), yet they introduce no particular logical complexities. The system incorporates one particular interpretation of the regulations: for all practical purposes there is little question that this interpretation is a reasonable one to take. The system itself is a combination of rules which formalize the written regulations, and supplementary rules which are derived from the opinion of an expert. There is no attempt to provide a system which will supply a particular recommendation. Instead, the user is expected to experiment with the system's various assumptions until he arrives at a conclusion of his own. And it may be that several mutually exclusive conclusions are equally likely. Deciding between them is then a matter of choice.

In this particular example the assumptions which the user is expected to vary are those relating to his own particular circumstances, and a value for the rate of inflation. In another example the state of the relevant legislation may be open to question instead. In principle, there is no additional technical difficulty in allowing users to vary also the rules which represent the legislation, to estimate the effects of taking different, mutually inconsistent, interpretations of the law. The law is ambiguous, imprecise and vague. But there is nothing logically inconsistent in contradictory conclusions which are derived from different assumptions.

And when these assumptions are equally plausible, deciding between them is at bottom a matter of choice.

11. CONCLUSION

In this paper I have discussed how it is possible to build an axiomatic model of some fragment of written legislation, and some of the ways we might use such a model. In particular, there are many areas of the law which we can reasonably regard as a set of definitions. Such legal definitions can be formalized fairly naturally as extended Horn clauses, although we will need to include a number of syntactic extensions to this language if the formalizations we produce are to resemble the style of the original legislation on which they are based.

How we proceed to apply such formalizations will depend on the nature of the application and on the way we plan to employ the system. In the first instance, we may be interested in constructing a system which is to take legal decisions autonomously in areas of the law which are not particularly sensitive. In most applications of legal systems, however, we shall be more concerned in providing legal decision support, either to help an adjudicator whose job it is to take decisions, or to aid an advocate in preparing and presenting his client's case in court. In all applications, I have argued, it is not so much the conclusions of the system which shall be of interest, but the proofs which such a system can construct.

Two contrasting and complementary methods for producing such a model of the law have been discussed. Hammond's description of entitlement to Supplementary Benefit was built by formalizing the opinion of an expert. Our formalization of the conditions under which an individual acquires British citizenship was based directly on the legislation itself. Recognizing that the law is often imprecise and ambiguous, and that it is necessarily vague, the difference between the two approaches is a methodological one rather than something more fundamental. There are practical differences between the two methods, however. We shall always have to justify our conclusions in terms of the assumptions on which these conclusions are based, and it is very much easier to assess the accuracy of assumptions when they can be traced to some authoritative legal source.

For the formalization of any reasonably large piece of legislation, for the purpose of building a practical application, I have advocated the use of a top-down, goal-directed approach. Any alternative would force us to identify the relevant bottom-level primitive concepts at the outset, before the formulation of rules can begin. Given the number of concepts which appear in any piece of legislation, common-sense as well as legal, I cannot believe that this would be a practical possibility in any but the

simplest of applications. This is not to suggest that we should not look out for concepts which are common to many types of legislation, or to begin with concepts which we have formalized before and which we are confident of representing already. But there will always be concepts we have not encountered before, and which we shall not know how to treat. Goal-directed formalization allows us to delay addressing the most difficult representational problems until it becomes unavoidable to do so. And we always have the option of refining the formalization later by defining what was once a primitive concept in terms of something more primitive instead. We shall also have to ground our formalization by representing the common-sense knowledge required to understand and apply the legislation. It is very much easier to construct such representations when we know what kind of common-sense knowledge is required, and the ways in which this knowledge will be used.

Like all formalization, representing legislation in computer programs is necessarily a process of trial and error. This holds true whether the formalization proceeds from the bottom-up or whether we attempt to write rules from the outset. If we isolate an individual paragraph, or section, or page of legislation, trial and error creates no real difficulties. Difficulties arise as the scale of the representation increases, especially when we try to formalize a piece of legislation in its entirety. As the number of rules in the formalization increases, even the most trivial of adjustments becomes a major exercise, sometimes requiring large-scale revision and restructuring of the rules. If we are ever to make the formalization of legislation relatively routine we need techniques to manage these problems of scale. Modules and other methods of structuring provide only a partial solution. For it is often the case that the presence of a later section in some Act indicates that we have misunderstood or misrepresented an earlier section completely. I gave an example of this kind. Although section 2-(1) of the British Nationality Act is labelled 'Acquisition by descent', it is only when we encounter section 14 that we realize that 'citizen by descent' in earlier sections did not mean 'citizen by section 2-(1)' (as we might have assumed), but rather 'citizen by descent' as described in section 14 of the Act.

Since there is no alternative to trial and error formalization and since there seems to be no way of avoiding revisions as later sections are encountered, we need to provide a programming environment to make the incorporation of these revisions as painless as possible. I would suggest that in practice it is the absence of a suitable environment, rather than any shortcomings in representational languages or techniques, that imposes a limit on the kind of applications we can approach with reasonable confidence of success.

It is important to distinguish the use of extended Horn clause logic for representing legislation from the use of PROLOG to execute these

representations. The example programs I have described in this paper are all executed by PROLOG (more precisely, by the augmented PROLOG system APES). In particular, these programs are executed top-down, reasoning backwards from conclusion to conditions. There will be many applications, however, where we shall want to execute programs with a mixture of backward and forward reasoning, or where we shall want to place the computation more in control of the user. One thing we shall never want to do is to execute these programs entirely bottom-up, supplying all relevant data in advance. This is a consequence of the very large number of detailed data items which are required for the solution of any given legal problem. To confirm my own British citizenship bottom-up, for example, I would have to anticipate the need for all kinds of data just in case they are required, ranging from the information that I was not found abandoned as a new-born infant, to the information that I have a reasonable command of English, that my father was never employed overseas in the service of the Crown, that I was never resident in the United Kingdom while in breach of the immigration laws as then in force, and many, many more such items of data. I might be willing to supply the date and place of my birth, the identity of my parents, and their date and place of birth, but I will not want supply much more than that in advance.

Vagueness and ambiguity are characteristic features of all legislation, but neither of them is an argument against the use of symbolic logic. Indeed I cannot see how we could begin to approach these problems without the precision of a formal language. It is no solution to invent a representational language which avoids these problems by ignoring them altogether. If a particular legal provision is genuinely ambiguous, and we manage to detect this ambiguity, we cannot ignore it. The law will have two distinct interpretations (or more) and, unless there are very exceptional circumstances, we shall have to make users aware of this fact and give them the opportunity of assessing the consequences of both interpretations. There may be reasons to prefer one interpretation over another, of course. We might prefer one particular interpretation because it is the more likely to be accepted in court; if we are given the job of preparing a client's case we might prefer one interpretation, not because it is the more likely to be accepted, but because it is in the interests of our client that our interpretation be accepted as the right one to take. How hard we shall have to work in persuading a court to accept our interpretation will depend the nature of the court itself, the ability of opponents in arguing the opposite, and on how many strong arguments we can adduce in support of our claim. In particular an interpretation which is far-fetched and which requires an elaborate and extremely involved argument to support it is unlikely to be accepted (although this does not mean it will not be accepted).

The resolution of vagueness too is fundamentally a matter of choice, constrained by the various rules of argumentation which a particular court will allow. In the long term, we shall need systems whose function is to advise on matters of interpretation, on the likelihood of getting some particular interpretation accepted, and on possible lines of argumentation to achieve this end. In the short term, I have mentioned a range of methods we could provide to help users arrive at some sort of decision. This help ranges from the simplest devices, like attaching some explanatory footnotes to questions about open-textured concepts, to sophisticated methods for accessing large data bases of decisions which were taken in previous similar cases. Trevor Bench-Capon and I are pursuing an approach which simulates the legal decision-taking process itself, by arranging for sets of inconsistent rules to generate arguments both for and against a particular conclusion. If all the generated arguments point in the same way then the decision will be relatively clear-cut. If not, it will be up to the individual user to assess the various arguments, and decide for himself which is the more likely to succeed in court, or which is in the best interests of his client. I have also suggested that we can go a long way towards conveying why a particular concept is vague by encouraging users to experiment with the various assumptions in the system. How much of this process we could mechanize, and what kind of advice we could provide about the best kind of experiments to perform, are the subjects of current investigation.

The use of a formal representation language gives the resulting program a meaning which is independent of its behaviour inside a machine. This property increases the usefulness of the formalization. Theorem provers no more sophisticated than PROLOG are adequate for executing the formalization as a program which applies the law. More sophisticated theorem provers can exploit the formalization for other purposes, particularly at the drafting stage and in the formulation of policy. The ability to prove logical consequences of a proposed piece of legislation can speed the drafting process and can improve the clarity of expression. The derivation of logical consequences helps to make clear whether proposed legislation is necessary or desirable. It can also suggest ways of simplifying or otherwise restructuring the legislation.

Let us imagine that at some point in the drafting of the British Nationality Act, concern was voiced that the proposed provisions were unfair because a person who was entitled to register as a citizen, but who died before he did register, could thereby deprive his children of British citizenship: but for his death, his children would be citizens. Suppose it is generally agreed that this state of affairs, if indeed it exists, is undesirable. Such concerns might be countered by suggesting that the current draft already has the property that the conditions under which a person is entitled to register as a citizen guarantee that his children are automati-

cally citizens in their own right. If this is true, problems with persons who die before registering as citizens do not arise and no amendment of the current draft is necessary. Now this hypothesis might or might not be true. It could be tested straightforwardly however. Either it is a consequence of the current draft that entitlement of a parent to register as a British citizen implies the citizenship of his children, or it is not. Such a theorem, if it is a theorem, could be proved mechanically. If it is not a theorem, then the device which I called tentative assumptions would give the conditions under which it would be a theorem, and these qualifications in turn would suggest how the legislation could be amended to eliminate the anomaly of children being deprived of citizenship by the death of their parents. In principle there is nothing difficult about proving such theorems. In practice, the tools will only be used by a draftsman if they are convenient to use. This would be another motivation (if one were needed) for pursuing the development of logic programming environments.

As long ago as 1957, Layman Allen [14] advocated the use of symbolic logic as a tool for analysing, and potentially simplifying, the structure of complex legal documents. He suggested, in particular, the use of automated tools for this task, and automated theorem-proving techniques have advanced dramatically since that time. Mathematics is concerned with axiomatic systems in which a small number of very general axioms give rise to a large number of powerful and very general theorems. The kind of theorem proving which will be required for help in drafting the law will be in complete contrast: we shall want to supply a large number of very detailed axioms (representing some fragment of law), and derive very specific theorems from these axioms.

I have labelled the examples in this paper 'legal expert systems'. Whether such programs should properly be called expert systems is a matter of terminology. They certainly have many of the features associated with expert systems. They derive their conclusions by inference from knowledge expressed explicitly; they can justify their conclusions, and they can generate requests for missing information as it is required. But these programs also resemble the executable specifications common in software engineering, particularly if they are to be used at the drafting stage. This resemblance is more than superficial. I have mentioned that in formalizing the British Nationality Act we were not concerned with efficiency, but with keeping the formalization as close as possible to the structure of the original legislation. And I have also remarked that it may be impossible to achieve this, while at the same time producing a program which behaves well computationally. In the case of the British Nationality Act formalization, it is possible to employ program transformation methods to yield a program which is computationally better behaved, but which is less readable and further removed

from the legislation as a consequence. What we shall want in such circumstances is a system which will execute the efficient but obscure program, but explain its conclusions in terms of the original, inefficient, formalization.

There are many areas in the law, and in regulation-based organizations more generally, for which it is possible to build applications in the short term. In this paper I have stressed the practical aspects, and I have mentioned only briefly the very many technical problems which remain to be solved. We need to discover natural and computationally tractable representations of the deontic concepts, for example, because they occur in many areas of law, although not nearly as frequently as might be supposed. Much more important, whether we treat the deontic concepts or whether we restrict attention to 'definitional' law, are the problems of legal reasoning which stem from conflicting beliefs and the need to reason with incomplete information. Experiments with varying assumptions are a crude attempt to deal with conflicting beliefs. Reasoning with incomplete information introduces default reasoning and reasoning with logical systems which are non-monotonic. The simplest way of providing a default, non-monotonic reasoning system is to adopt the treatment of negation as failure. This treatment of negation is only justified under very specific conditions however. In many practical applications these conditions will be satisfied. In general they will not, and we shall be unable to proceed without some more sophisticated treatment for making default inferences on the basis of incomplete information.

There is hardly an area of research in Artificial Intelligence which could not find in the law an immensely rich source of challenging problems. Unlike most other experimental domains, however, any advance finds immediate practical application, in a domain moreover where applications will have the profoundest social implications.

Acknowledgements

I am indebted to Robert Kowalski for his enthusiasm, encouragement and support, and for his many valuable suggestions. I am particularly grateful to Trevor Bench-Capon for his comments on this paper and for many stimulating and informative discussions, to Peter Hammond to whom much of the credit for APES belongs, and to Fariba Sadri for undertaking the tedious job of formalizing the British Nationality Act. I am also grateful to Thorne McCarty and Ronald Stamper for discussions which have helped me enormously. This work was supported by the Science and Engineering Research Council.

REFERENCES

1. Shortliffe, E. H. (1978) *Computer-based medical consultations: MYCIN.* North-Holland/Elsevier.
2. Duda, R., Gashing, J., and Hart, P. (1979) Model design in the PROSPECTOR consultant system for mineral exploitation. In *Expert systems in the micro-electronic age* (ed. D. Michie). Edinburgh University Press, Edinburgh.

3. Waterman, D. A. and Peterson, M. A. (1980) Rule-based models of legal expertise. *Proc. First National Conf. on Artificial Intelligence,* pp 272–275. Stanford University.
4. Waterman, D. A. and Peterson, M. A. (1981) Models of legal decision making. *Technical Report R-2717-1CJ.* The Rand Corporation, Santa Monica.
5. Hammond, P. (1983) Representation of DHSS regulations as a logic program. *Proc. BCS Expert Systems '83,* Cambridge. British Computer Society.
6. Hammond, P. and Sergot, M. J. (1983) A PROLOG shell for logic based expert systems. *Proc. BCS Expert Systems '83,* Cambridge. British Computer Society.
7. Hammond, P. and Sergot, M. J. (1984) *APES 1.1 reference manual.* Logic Based Systems Ltd, Richmond, UK.
8. Clark, K. L. (1978) Negation as failure. *Logic and data bases* (eds H. Gallaire and J. Minker). Plenum Press, London.
9. Sergot, M. J. (1983) A Query-the-user facility for logic programming. In *Integrated interactive computer systems* (eds P. Degano and E. Sandwell). North-Holland, Amsterdam.
10. Clark, K. L. and McCabe, F. G. (1984) *micro-PROLOG programming in logic.* Prentice-Hall, London.
11. Stamper, R. K. (1979) LEGOL: Modelling legal rules by computer. *Computer science and law* (ed. B. Niblett). Cambridge University Press, Cambridge.
12. Jones, S., Mason, P. J., and Stamper, R. K. (1979) Legol-2.0: a relational specification language for complex rules. *Information Systems* **4**(4), 293–305.
13. Sergot, M. J. (1980) Programming law: LEGOL as a logic programming language. Department of Computing, Imperial College of Science and Technology, London.
14. Allen, L. E. (1957) Symbolic logic: a razor-edged tool for drafting and interpreting legal documents. *Yale Law Journal,* **66,** 833–79.
15. Allen, L. E. (1979) Language, law and logic. plain legal drafting for the electronic age. In *Computer science and law* (ed. B. Niblett). Cambridge University Press, Cambridge.
16. Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P., and Cory, H. T. (1985) The British Nationality Act as a logic program. Research report, Department of Computing, Imperial College of Science and Technology, London (December 1983, revised May 1985). To appear in *Commun. ACM* **29**(5), 370–86.
17. Sharpe, W. P. (1984) Logic programming for the law. *Proc. BCS Expert Systems '84.* British Computer Society.
18. Gabbay, D. M. and Reyle, U. (1984) N-PROLOG: An extension of PROLOG with hypothetical implications. I. *J. Logic Programming* **4,** 319–55.
19. Bench-Capon, T. J. M. and Sergot, M. J. (1985) Towards a rule-based representation of open texture in law. Research report, Department of Computing, Imperial College of Science and Technology, London.
20. McCarty, L. T. (1979) The TAXMAN project: towards a cognitive theory of legal argument. In *Computer science and law* (ed. B. Niblett). Cambridge University Press, Cambridge.
21. McCarty, L. T. (1983) Permissions and obligations. *Proc. IJCAI-8,* Karlsruhe.
22. Suphamongkhon, K. (1984) Towards an expert system on immigration legislation. M.Sc. thesis, Department of Computing, Imperial College of Science and Technology, London.
23. Lowes, D. (1984) Assistance to industry: a logical approach. M.Sc. thesis, Department of Computing, Imperial College of Science and Technology, London.
24. Chan, D. (1984) A logic based legal expert system. M.Sc. thesis, Department of Computing, Imperial College of Science and Technology, London.
25. Sergot, M. J. (1982) Prospects for representing the law as logic programs. In *Logic programming* (eds R. Clark and S. Tarnlund). Academic Press, London.