YEARBOOK OF LAW COMPUTERS
AND TECHNOLOGY
VOLUME THREE, 1987

Some Problems with Prolog as a Knowledge Representation Language for Legal Expert Systems

Thomas F. Gordon

Introduction

In vol 2 of this Yearbook, Marek Sergot and his collegues at Imperial College discussed their formalisation in Prolog of the British Nationality Act[1]. The idea of using Prolog as a *"knowledge representation language"* for legal expert systems has been a popular one in the last few years,[2] but Prolog suffers from a variety of problems which make it unsuitable for this purpose. Some of these problems were mentioned in the British Nationality Act article. I would like to expand upon these difficulties here. I begin by outlining a view of the process of deciding legal cases. Next, the architecture of knowledge-based systems is described. I then describe some properties these systems must possess to be useful tools for supporting the legal decision-making process, and I measure Prolog against this list of properties. Finally, I briefly discuss Oblog, the language we have been developing to overcome the shortcomings of Prolog for legal expert systems.

Deciding Legal Cases

The view of legal decision-making as *merely* a form of deduction has been thoroughly criticised. We are all familiar with the Realist school and Holmes' dictum concerning the life of the law not being logic, but experience[3]. Susskind recently reviewed these jurisprudential issues in the context of legal expert systems[4] and argued that researchers eager to apply artificial intelligence (AI) methods to legal problems would be well advised to familiarise themselves with the relevant jurisprudence literature. I do not intend to address all the theoretical issues here but, sensitive to Philip Leith's misinformed criticism that artificial intelligence researchers have a simplistic notion of the role of logic and legal rules[5], would like to summarise at the outset our position on these matters.

The decision in a concrete legal case is a process of theory or *"model" construction*[6]. The theory consists of two components, a theory of the facts of

the case, and a theory of the applicable law. The theory of law need not be, and cannot be, all-encompassing. That is, the theory is not intended to cover all areas of law in some jurisdiction, and is not intended to be adequate for deciding all cases, past and future. Rather, it is intended to be adequate for deciding the case before the court.

Although there may be a wide range of acceptable theories, a judge's power to construct a theory along the lines he desires, perhaps to achieve some particular result, is not unconstrained. Rather, a judge's power is limited by the properties which a theory must satisfy. The first of these constraints concerns the relationship of the theory of law to the legal *sources*. These sources are of many different kinds, with various degrees of *authority*. *Primary sources* include statutes and published decisions of prior cases. *Secondary sources* may include commentaries and law review articles. Not to be forgotten, however, is the background information concerning such things as our culture, history, current events, and moral values.

Note that Statutes and prior decisions are not themselves the law, but sources of law. This distinction is of critical importance. Statutes and case decisions are merely texts of special authoritative value. It is the *meaning* of these texts which is the law. Their meaning can be acquired only through a process of *interpretation*, which implies an interpreting *agent* in some *context*. The relevant context is provided by the background information. This observation about the status of Statutes and cases would remain true even if they were to be represented in some less ambiguous form than natural language, such as the "normalised" language suggested by Layman Allen[7], or some formal language (which we are not suggesting). To the extent that such formalisation efforts make the intended meaning of legislation clearer, they further constrain the power of judges; but they cannot relieve judges of their duty to discover the intended meaning.

The interpretation of Statutes and cases surely includes a deductive component, but as legal AI researchers have recognised[8], other reasoning processes, such as reasoning by analogy and by hypothetical cases, play a role here as well.

The second constraint on a judge's power to decide is the relationship between the theory of the facts and the evidence. Facts, by which I mean the atomic sentences of the theory, such as "John is 28 years old", are not given but must be found. This task is not trivial. Although some facts may be deduced from other facts using logic, there must always be some facts that arise from essentially nonlogical processes. The problem is one of interpretation, of associating terms and predicates of the language of the theory of law with perceptual events. This process is poorly understood; its difficulty has been the principal weapon in the arsenal of those disputing the value of logic for legal reasoning.

Facts can be divided into two kinds: *primary* facts, which describe the events of a case using conventional, nonlegal terminology; and *secondary* facts, which use the technical, legal terminology of a theory of law. The problem of determining whether some secondary fact exists given a set of primary facts has been called the "problem of classification".[9] In civil law countries this task is called *subsumption*. In the legal context, the classification

problem is complicated by the dynamic quality of the theory of law from which the terms used to describe the facts are drawn. The construction of the theories of law and facts is an iterative, quasi-parallel process. With the aim of characterising the facts in a certain way, a theory of the law may be, provisionally, constructed. Using this theory, the facts may then be filled out somewhat; but, should difficulties occur, the theory of the law may be modified. This is related to, but different from, the problem of "open textured" legal concepts. As the term was originally used by H L A Hart, "open texture" refers to the evolving meaning of legal terms over a series or line of related cases[10].

As may be clear from this description, theory construction is a *purposeful* activity. It cannot be an entirely objective act, as the iterative process of shifting and modifying the theories of law and fact continues until, in some subjective sense, the judge is satisfied with the result.

An additional requirement of the theory is logical consistency, which may be reasonably viewed as an ideal to strive for, rather than an absolute requirement, due to the computational difficulties of showing consistency given any problems of realistic proportion. Moreover, it must of course be shown that the legal decision is a logical consequence of the theory constructed. This is one point where deduction clearly plays a role. Some consider this to be a relatively trivial process once a satisfactory theory has been constructed. In general, however, because of the undecidability of logics as expressive as first-order predicate logic, and granted the usual computational complexity of proof procedures for less expressive, but decidable logics, this task is not so trivial as it may at first seem.

Finally, and most important, a judge must persuasively argue that the decision is *just*. This calls for rhetorical skills and appeals to our emotions. It is not enough that the theory constructed bears an acceptable relation to the authoritative legal sources and that the decision is a logical consequence of the theory; we must also be persuaded that the decision is right and just. Should a judge be so constrained by the authoritative sources as to be unable, despite reasonable effort, to construct a theory allowing a just decision, he should at least explain his dissatisfaction with the decision he considers himself bound to make, and appeal to the appropriate authority to make the necessary changes in the law.

Do judges, then, make law? Strictly speaking, no. The theory of law a judge constructs, like the sources of law upon which the theory is founded, such as Statutes, is not the law, but another representation of the law. Reasonable persons can disagree as to whether or not the theory is a faithful representation. The law itself remains an amorphous object. All representations of the law, including statutes, however carefully drafted, are by their nature simplifications and inherently subject to multiple interpretations. This is so as each person interpreting the symbols of the representation provides a context based on his own unique history and experience. Linguists refer to the *conventional* meaning of terms,[11] which arises from our common experience and culture. Although, in the abstract, we can accept that terms do have a conventional meaning, this meaning is impossible to pin down in any given instance. Nevertheless, it is the existence

of conventional meanings which permit our laws to have their intended normative effects.

Knowledge-based Computer Systems

At no point in my description of our view of legal decision-making did I mention *formal* reasoning. The theories of law and fact may be (and have always been) stated in natural language, and the reasoning used to show that the decision or judgement is a logical consequence may be informal. Logical consequence is, after all, a *semantic* notion which is presupposed by formal systems. The purpose of formal systems is to substitute or supplement reasoning at the semantic level by purely syntactical manipulations which can be, at least partially, automated. In the spirit of the industrial age, the hope is that such systems will increase the efficiency and reliability of deductive reasoning.

In the case of legal systems, the goal is to allow us more accurately and efficiently to forsee the legal consequences of our actions, which would allow us to plan our affairs with more confidence and promote the normative purposes of law. Moreover, as we are presumed to know the law, which given the complexity of modern law is a legal fiction if there ever was one, this capability of legal expert systems could make this presumption more realistic and just. It is also hoped that, when legal disputes do arise, that expert systems can help reduce the costs of resolving them, thus opening up the legal system to those who have been denied access for economic reasons. There are of course also risks: the risk that such systems would unduly upset the balance of power between the legislative and judicial branches, reducing the ability of judges to reach just decisions in individual cases, and the risk of making the law still more complex and cryptic, to name just two.

An expert system is a *knowledge-based* computer system containing a representation of expert knowledge. Whether or not some knowledge-based system is an expert system depends on the intended users. A system whose knowledge-base is common knowledge to lawyers would not be an expert system for the legal community, but may very well be if adapted to the needs of the layman.

Following Brachman and Levesque[12], we consider a knowledge-base system to be a component, or module, of some larger software system. Roughly speaking, this module consists of three components: a knowledge base, a logic and an *inference machine*. Responsibilities of the module include managing the knowledge base and answering queries about what is logically implied by the knowledge base. Whether or not something is a logical consequence of the knowledge base is determined by the logic of the knowledge-based subsystem. This logic is usually not explicitly represented in software, but is used to design and measure the inference engine, which is. Note that "classical" first-order predicate logic is not the only alternative here. In colloquial speech, "logical" and "reasonable" are synonyms, which seems to imply that there is only one logic. Technically speaking, however, there are arbitrarily many logics, of which first-order predicate logic is the most familiar. A logic is not a discovery, but a human invention.

The inference engine consists of two components, a grammar and a control structure. The grammar consists of a set of transformation rules which allow us to *generate* new sentences from a set of sentences, and possibly to *test* whether some sentence is derivable from a set of sentences by a sequence of transformation rule applications. I say possibly, as it is in general an intractably hard problem to find a suitable sequence of rules. It is the responsibility of the control structure to determine which rules to apply and when. The control structure need not be (and if the logic is expressive enough, *cannot* be) an algorithm. That is, it need not be guaranteed to find a solution within a finite amount of time. Rather, it may use *heuristic* methods and may even resort to asking the user for help.

If, with respect to the logic of the system, the transformation rules are *truth preserving*, we say the transformation rules are *correct*. If there is always some sequence of transformation rules for a sentence which is a logical consequence of some set of sentences, where the meaning of "logical consequence" is determined by the logic of the system, then we say the rules are *complete*. Given a complete set of transformation rules, if the control structure is clever enough always to find a solution if one exists, or to terminate in finite time with failure if one does not exist, then we say the control structure is *complete*. (A complete control structure is algorithmic, and is possible only if the logic of the system is decidable.) Note that "completeness" in the context of inference machines has two different meanings.

The distinction between the grammar and control portions of an inference engine is abstract. In a particular inference engine, these components may not be so clearly distinguished.

Given our view of legal decision-making as a process of theory construction, in what ways could knowledge-based computer systems support this task, particularly given the limited role of deduction in this view? Obviously the requirement that a legal decision be a logical consequence of the theory constructed, would be assisted by such a facility. This is so, even though the theories of law and facts are subject to change during the construction process. The requirement that the decision be a logical consequence of the theory is a *constraint* which can be satisfied by modifying the theory, the decision or both. A knowledge-based system might allow a lawyer to play "what-if" games to discover quickly the effects of various alternative representations of the law, facts and decision.

In other application areas for expert systems, a distinction is usually made between the "knowledge engineer", who constructs the knowledge base for some domain, and the "user" who then applies the expert system to some specific problem. In the case of legal expert systems, the lawyer using the system fills both roles simultaneously. This is not meant to suggest that a legal expert in some domain could not prepare a theory of his specialty for later use by a non-specialist lawyer in analysing some case. It only means that the theory constructed by the specialist must be open to revision by the lawyer user, who is the person ultimately responsible for the quality of the theory of law applied.

Some Problems with Prolog

Prolog fits into this conception of knowledge-based systems as follows: the logic component of Prolog is *extended Horn clause logic*, which is Horn clause logic extended with the *closed-world assumption* and negation as failure. Unlike Horn clause logic, extended Horn clause logic is not a subset of standard first-order predicate logic; it has a non-standard semantics. Prolog is, for example, *non-monotonic*: a formula which is a logical consequence of some set of axioms may not be a logical consequence of the same set extended by additional formulas. (Such non-monotonicity may be a desirable property, but non-montonicity can arise in various ways). Prolog has only one transformation rule, called resolution[13], which is especially well-suited for mechanisation. The control structure of Prolog is very simple: depth-first "blind" search with backtracking. It does not apply heuristics to limit the search. This control strategy is incomplete and is usually augmented in the expert system shells based on Prolog, such as APES [14], the system used to model the British Nationality Act. In APES, for example, the system may question the user for information when it is unable to prove some subgoal. In addition, Prolog programmers can influence the control strategy with the *cut* construct, which can be "dangerous", as careless use can affect correctness.

In the criticism of Prolog that follows, it should be remembered that I will be focusing on Prolog's usefulness as a knowledge representation system for theories of legal domains. Prolog was not designed for this purpose, but as a programming language. Some people have referred to Prolog as an *assembler* for knowledge representation systems, as it is relatively easy to implement in Prolog languages with characteristics lacking in Prolog itself.

What properties should a knowledge-based system for legal expert systems have, and how does Prolog fare with respect to these features? Rather than try comprehensively to list all requirements, I will focus on those which have come to our attention during our efforts to use Prolog for building "toy" theories of various legal domains.

First, the system should support the definition of hierarchies or taxonomies of legal concepts. Arguably, extended Horn clause logic is expressive enough for this purpose. The problem with Prolog here is its naive control strategy. If concepts are defined by Prolog clauses in a natural way, with *genera* and *differentia*, Prolog quickly goes into an infinite loop when trying to apply the clauses. Perhaps an example here will help. Suppose we want to define contracts as enforceable agreements. This could be expressed as:

(r1) contract(X) if agreement(X) and enforceable(X).

Here we have stated that enforceable agreements are contracts. If we want to complete this definition by saying in addition that all contracts are enforceable agreements (i.e. that something is a contract *if and only if* it is an enforceable agreement), then we must assert as well:

(r2) enforceable(X) if contract(X).
(r3) agreement(X) if contract(X).

Now suppose we have asserted in addition

(r4) agreement(c1).
(r5) enforceable(c1).

and want to show contract(c1). Prolog will find r1 and reduce this top-level goal to two subgoals.

agreement(c1), enforceable(c1)

Prolog will then try to show agreement(c1) by applying r3. The list of remaining goals now is

contract(c1), enforceable(c1)

so we are back to trying to show contract(c1), which leads to another application of r1, and so on. Note that this really is just a limitation of Prolog's control strategy; the resolution inference rule is complete. We just need to apply the clauses in another order. (In this example, we should have used the two "facts", r4 and r5, after the first application of the rule r1).

It is usually recommended that facts be placed before rules, so it might be argued that our example is somewhat unfair. But putting the facts at the beginning does not solve the problem, it just substitutes one kind of incompleteness for another. Suppose that the facts r4 and r5 appear before the rules r1 to r3. Now suppose that there is an additional fact

(r6) contract(c2).

which is placed at the end of the other facts, i.e. after r5, and before the rules. If we again pose the question contract(c1), Prolog will find the solution this time. But if we generalise the enquiry somewhat and ask Prolog to find all contracts, contract(X), Prolog will repeatedly find contract(c1), but will fail to find contract(c2), even though this fact has been asserted!

A second problem with Prolog, which it inherits from standard logic, is that it assumes we have perfect information. It is taken for granted that we have already built a complete theory of the legally relevant facts before asking Prolog to prove some goal. In interactive systems, such as APES, the theory of the facts need not be complete before putting a question to the system, but it is nonetheless assumed that the *user* is always able to answer the questions posed by the system. This is a very serious problem for legal expert systems; not only do lawyers not have perfect information, they usually have *no* first-hand experience of the relevant events. Discovering evidence is a time-consuming and costly task.

Justice has its price, but legal systems use *presumptions* and *burden of proof rules* which allow decisions to be made in the face of incomplete information

and, just as important, allow the interested parties to tailor somewhat the cost of reaching a decision to the value of the dispute. Moreover, judges are bound to reach a decision in a case. They may not simply say "I don't know"! Our computational theories of legal domains should preserve this ability to reach decisions with less than perfect information. Prolog fails at this. Let us look at one of the examples used from the British Nationality Act.[15] Section 11-(1) and 11-(2) of the Act state:

> 11-(1) Subject to subsection (2), a person who immediately before commencement—
>
> (a) was a citizen of the United Kingdom and Colonies; and
> (b) had the right of abode in the United Kingdom under the Immigration Act 1971 as then in force,
>
> shall at commencement become a British citizen.
>
> (2) A person who ... [P] ... shall not become a British citizen under subsection (1) unless ... [Q] ...

These sections were represented in Prolog as:

> *x* acquires British citizenship on date *y* by sect. 11.1
> if commencement is on *y*
> and *y1* is immediately before *y*
> and *x* was a citizen of the United Kingdom and Colonies on *y1*
> and on date *y1*,*x* had the right of abode in the UK under the Immigration Act 1971
> and not [*x* is prevented by 11.2 from acquiring British citizenship at commencement]

Sections 11-(1) and 11-(2) consist of three levels of rules which are collapsed into one-level by the Prolog formulation. The first level rule states

> a person who immediately before commencement—
> (a) was a citizen of the United Kingdom and Colonies; and
> (b) had the right of abode in the United Kingdom under the Immigration Act 1971 as then in force
>
> shall at commencement become British citizen.

This rule is an intentional simplification of the complete law. In the original text, this general rule is subject to the exception stated in 11-(2), namely that

> A person who ... [P] ... shall not become a British citizen under subsection (1)
> where P is some set of conditions that the state must prove to cancel the effectiveness of the general rule in subsection 1. This exception is subject to a further exception, "unless ... [Q] ...", which the person desiring citizenship may resort to if the state has succeeded in showing P.

The Prolog formulation wrongly places the entire burden of proof on the party desiring citizenship! In particular, the party must show (not P or Q). In

all fairness, I should mention that the Imperial College group recognises that statutes have this structure even though their representation in Prolog does not, and cannot, preserve it: "*Legislation is often drafted by a general rule, followed separately by a list of exceptions to it. Such rules are common in the British Nationality Act*".[15]

Sergot's group has suggested that Prolog's negation as failure rule can often be used to handle exceptions, as negation as failure assumes that something is *not* true unless it can be shown explicitly to be the case. For example, if p is to be assumed true if q is shown, subject to an exception r, we might try to represent this in Prolog as

(r7) p if q and not r.

The party desiring to show p must, of course, prove q; but arguably he could prove (not r) simply by expending no effort trying to prove r; (not r) would then be true by negation as failure. This appears to be a solution at first glance, but suppose there is another rule regarding r which states

(r8) r if not s.

If nothing is done to prove (not r), in particular if nothing is done to prove s, then (not s) may be true by another application of the negation as failure rule, r would be true by r8 and (not r) would fail in our attempt to apply r7. Thus our formulation of r7 fails to allow p to be presumed by just showing q.

The negation as failure rule causes everything not *known* to be true to be assumed false. This is too simplistic. A knowledge representation system for legal applications must provide a more sophisticated mechanism for controlling what is assumed when.

As can be seen by the example from the British Nationality Act above, burden of proof is often distributed implicitly by a system of general rules and exceptions, rather than by adding explicit control rules to a set of ordinary first-order sentences. In the context of logic programming, for which Prolog was designed, Robert Kowalski's famous formula

program = logic + control

suggests that, ideally, logic and control aspects should be cleanly separated. General rules and textually separate exceptions mix these two aspects. I should like to argue that this tradition well serves an important function of the law, and that Prolog's failure to preserve this tradition is an additional reason for its inappropriateness as a language for legal expert systems.

The core of my argument concerns the normative goals of the law. The law's primary purpose is not to settle disputes after they have arisen, but to prevent disputes and guide human conduct. This function can only be fulfilled to the extent that we are able to learn the law, and to apply it as we plan our affairs. The traditional general rule/exception structure of the law provides a mechanism by which persons can *incrementally* learn the law.

Knowing a general rule, which is an incomplete statement of the law, is better than knowing no rule at all. We can plan our actions using general rules and, in some sense, *probably* do the right thing. Knowledge about some legal domain can be *deepened* by learning the next level of exceptions, without having to relearn or modify our understanding of the more general principles.

The law is, or at least should be, structured so as to be easily remembered and applied. This cognitive quality of traditional legislation is one reason why "normalised drafting" is not yet a mature idea. Normalisation of the type proposed by Allen[16] clarifies the intent of the legislature, but makes the legislation impossible to remember. For a related reason, Prolog (and standard predicate logic) are inadequate as knowledge representation languages for legal expert systems. It is not enough for our models of the law to be suitable for symbolic manipulation by a computer, they must first of all be comprehensible to their human users. An expert system, especially a legal expert system, must be capable of explaining its reasoning in terms users can appreciate. To achieve this transparency, it is helpful if the computer model and the legal sources upon which the model is based have a similar structure.

Standard predicate logic, and Prolog, require that systems of general rules and exceptions be *collapsed* into logically complete formulas. By complete here, I mean that no rule may be subject to implicit exceptions. Suppose some stature includes a set of general rules and exceptions with the following form:

```
(r9)   p if q and r.
(r10)  not p if s and t.   /* exception to r1 */
(r11)  p if u and v.   /* exception to r2 */
```

I have used a Prolog-like syntax here; but, careful, this is not Prolog. Prolog requires us to collapse these three rules into *one*:

```
(r12)  p if q and r and [not[s and t] or [u and v]].
```

Even this short example demonstrates how incomprehensible the collapsed version of a system of general rules and exceptions can be. Moreover, if a further exception were to be introduced, this rule would have to be modified and would grow in complexity, violating our principle that new knowledge should require little change to existing knowledge.

Large, complicated rules in Prolog *can* be broken down into a set of shorter rules, by introducing new concepts (i.e. predicate symbols). The example above, r12, could be rewritten as

```
(r13)  p if q and r and p1.
(r14)  p1 if p2 or p3.
(r15)  p2 if not [s and t].
(r16)  p3 if u and v.
```

But what symbols should we choose for p1, p2, and p3? In an abstract

example like the one here, this is no problem; we just choose arbitrary names. But when trying to model an actual law, we like our symbols to be mnemonic. Unfortunately, mnemonic names do not suggest themselves in practice, as these predicates have nothing to do with the legal concepts of the domain being modelled; their sole purpose is artificially to decompose large rules. Some authors attempt to avoid this problem by referring to code section numbers, as in:

(r17) p if q and r and 13.b
(r18) 13.b if . . .

Here, 13.b would be the section where the exception to the general rule of r17 is to be found. Although lawyers tend to be capable of this, remembering arbitrary code section numbers is not easy. Moreover, this technique does not help us to model non-statutory law. Finally, it is not our purpose to model the statute, not the law represented by the statute; Refering to code section numbers in rules blurs this dinstiction.

Oblog

Oblog, for object-oriented logic, is an experimental hybrid knowledge representation and reasoning system developed by our group, for Forschungsstelle fur Informationsrecht (FS-INFRE) of the Gesellschaft fur Mathematik und Datenverarbeitung (GMD), under the leadership of Prof. Herbert Fiedler. Oblog is our attempt to correct those shortcomings of Prolog discussed above.

The version of Oblog to be discussed here is the second version of the system. The first version[17], was an extension to the logic programming system MRS [Genesereth to support structured objects. That is, it was an attempt to combine the features of the logic and frame approaches to knowledge representation. There we tried to retain the truth semantics of first-order logic by interpreting inheritance hierarchies as *similarity* networks. Type hierarches were used only as a construction aid for knowledge bases; assertions about a type were not inherited by its subtypes. Rather, the type hierarchy was used merely to remind the knowledge engineer that assertions attached to a supertype may also be applicable to subtypes. It was the responsibility of the knowledge engineer to decide whether some assertion was indeed applicable to some subtype, in which case he was required to make an additional assertion to that effect. This approach suffered from several inadequacies. There was no inference rule that an instance of some type is also an instance of all supertypes of the type, which is clearly counter-intuitive. (Leo the lion could not be shown to be animal.) Another shortcoming of the previous version of Oblog was its adequate support for plausible reasoning. It did not address the problem of reaching decisions with less than complete information.

The current version of Oblog also supports structured knowledge representation, although in a somewhat different manner. There is now a terminological component for defining taxonomies of *types* and *attributes*. These are similar to the *concepts* and *roles* of KL-ONE[18] *Entities* in Oblog are

the structured, frame-like objects. They are instances of types and have attributed values. An entity which is an instance of some type is now also an instance of all supertypes of that type. From a frame language perspective, attributes are similar to *slots*.

Oblog *rules* are sets of Prolog-like definite clause procedures for computing attribute values. Each rule is associated with a type and an attribute. There may be multiple rules concerning some attribute for various types. The type hierarchy is used to order the rules. Rules attached to types higher in the hierarchy are considered to be more general than other rules concerning the same attribute attached to lower-level types. This approach avoids the problem of determining relative generality by a direct comparison of the bodies of clauses. In effect, attaching a rule to a type is an *assertion* as to the rule's generality and is a responsibility left to the user. Thus, in Oblog we have the ability to define general rules subject to implicit exceptions. To give a feel for the language, here is a short legal example, drawn from a "toy" model of the German law concerning the validity of contracts with minors. Essentially, a contract is presumed to be valid, i.e. enforceable. (Under German law, the question of the *existence* of a contract is separate from that of its validity.) A contract with a minor, however, is not enforceable, *unless* it has been ratified by the guardian of the minor. There are further exceptions, but we can ignore them for our purposes here. One way of representing this law in Oblog would be:

```
(1)  type contract
(2)  type minor
(3)  relation party
(4)  type contract-with-minor (contract)
       contract-with-minor(X):-
          party(X,Y),
          minor(Y).
(5)  Property ratified-by-guardian
(6)  type ratified-contract-with-minor (contract-with-minor)
       ratified-contract-with-minor(X):-
          agreement-with-minor(X),
          ratified-by-guardian(X).
(7)  property enforceable
(8)  rule contract enforceable
       enforceable(X). ;Horn clause without body
(9)  rule contact-with-minor enforceable ;null-rule
(10) rule ratified-contract-with-minor enforceable
       enforceable(X).
```

Some of the concept types, such as "contract" and "minor", have been left completely undefined. They are thus primitives of the model. A "contract-with-minor" has been defined as a subtype of "contract" for which a party of the contract is a minor. These are not definitions in the normal sense, as they state sufficient but not necessary conditions for showing that some entity is an instance of the type. Also, although the example does

not demonstrate this, a type can be defined to be a subtype of a *set* of types. Thus Oblog supports a kind of *multiple inheritance.*

Properties and relations are unitary and binary attributes, respectively, of entities. Attributes, like types, can be arranged in a hierarchy. This feature, too, is not demonstrated in the example. As all goals in Oblog concern the types and attribute values of entities, goals are restricted to unary or binary predicates. It can be shown that this requirement does not restrict the expressiveness of the language and offers several advantages. It is not at all obvious how to represent structured objects in predicate logic. There are several alternatives, each with its advantages and disadvantages. But, whatever alternative is chosen, it is important to apply it consistently throughout the knowledge base. Oblog relieves the user of the burden of addressing this issue, and applies a strategy that is generally agreed to be the most flexible and modular, if somewhat verbose.

Rules in Oblog are indexed by type and attribute. A rule indexed by some type is more general than rules concerning the same attribute indexed to subtypes of that type. Rule 9 is a *null rule.* The purpose of such rules is to cancel the applicability of the general rule for instances of the subtype, without having to provide a set of clauses for determining values of the attribute for the subtype.

Let us add to this knowledge base some case specific information:

(11) entity c1(contract)
party: Jeffrey
ratified-by-guardian
(12) entity c2(contract-with-minor)
(13) entity Jeffrey (minor)

These "facts" state that there is a contract, cl, which has a party, Jeffrey, and has been ratified by a guardian. c2 is known to be a contract with a minor. Jeffrey is known to be a minor.

With this knowledge base, what kinds of queries can Oblog answer? It can show, of course, that c1 is a contract which has Jeffrey as a party and that Jeffrey is a minor. That is, Oblog can retrieve the asserted facts. But Oblog can also derive that c2 is a contract, using the type hierarchy. That is, we have augmented resolution with special-purpose inference procedures for answering questions about type (and attribute) relationships. Unlike Prolog, this special purpose inference machinery avoids infinite loops. Concepts can be defined by *genera* and *differentia* in a natural way. Most interesting, however, is Oblog's control strategy for handling hierarchies (partial orders) of rules. Suppose we wish to know which contracts are enforceable by asking, in effect:

?- contract(X) and enforceable(X).

Oblog finds the first contract, c1, and then uses rule 8, which states the general rule that contracts are enforceable, to conclude that c1 is enforceable. Not that this conclusion is only plausible or approximate. An explanation

component for Oblog would tell us that the general rule was applied and allow us to examine the exceptions if desired. But, even without an explanation component, the user may ask for a *better* solution. Oblog searches the rule hierarchy, top-down, for an exception, and would next find rule 10, the rule for ratified contracts with minors. (Rule 9, the null rule for contracts with minors, is skipped as it cannot be used to determine enforceability. It would, however, have prevented rule 8 from being used if we had *asserted* that c1 is a contract with a minor.) To apply rule 10, Oblog must first determine whether or not c1 is a ratified contract with a minor. It uses the "definition" of ratified contracts with minors for this purpose, statement 6. In applying the clause of the definition, the same control strategy is applied as described here. Thus, plausible reasoning is used to determine whether or not an entity is an instance of some type, even though the clauses of definitions are not directly subject to exceptions.

The description above of Oblog's control strategy is simplified somewhat. Oblog must also consider the structure of the *attribute* hierarchy. When we ask whether or not a contract is enforceable, for example, the relationship of the enforceability attribute to other attributes must be considered. In addition to the rules directly asserted for some attribute, rules for subattributes of the attribute may be applicable as well. As I said, Oblog is an experimental system under development. It is unlikely that the system will remain the way I have described it here. Problems we have recognised include the lack of negation, the inability to infer the *existence* of entities (they must all be asserted), and the fact that a new, often artificial type must be introduced to create an exception to some rule. These difficulties are minor, however, and the next implementation of the system may have corrected them. Much more serious is Oblog's lack of a declarative semantics! Like so many A1 knowledge representation systems, Oblog is an entirely syntactical system. As there is no known *logic* for which Oblog is a calculus, it does not make sense to talk about its correctness or completeness.

Conclusion

Legal decision-making is a process of theory construction, in which deduction plays a subordinate but important role. We have taken no position on the issue of whether or not true artificial intelligence is possible, given the difficulties of encoding the contextual information needed to interpret the meaning and intended domain of application of some theory. Although we have not discussed these issues here, for ethical, moral and political reasons, if not necessarily technical ones, legal decisions should remain the responsibility of living, breathing persons. Artificial intelligence methods nonetheless may have a great deal to offer lawyers in the way of powerful *tools* for increasing the efficiency of the theory construction process. Neither standard first-order logic nor Prolog, however, appear to be well suited for the task of representing legal domains. We leave the safe, familiar ground of standard logic only reluctantly; but fail to see an alternative to looking for more suitable, non-standard logics. Our system, Oblog, has been helpful for clarifying our intuitions about how to reason with general rules and

exceptions, but it is at present an *ad hoc*, purely syntactic system. It may be possible to discover a logic for which the Oblog inference machine is a correct calculus; but traditionally calculi are developed for logics, and not vice versa. Actually, a logic and calculus suitable for legal expert systems must be developed together, as the range of suitable logics is limited by the computational and cognitive properties of their possible calculi.

Thomas F. Gordon
Gesellschaft fur Mathematik und Datenverarbeitung MBH
Bonn
W Germany

Notes

1 M Sergot et al.; "Formalisation of the British Nationality Act"; 1986; Yearbook of Law, Computers & Technology; vol. 2;40-52.
2 S Grundmann; "Voruberlegungen zur Ausarbeitung eines computergestutzten juristichen Expertensystems"; 1986; Datenver arbeitung im Recht; J Schweitzer Verlag; Munich; 175-190.D Schlobohm; "TA—A Prolog Program which Analyses Income Tax Issues under Section 318(a) of the Internal Revenue Code"; 1985; Computing Power and Legal Reasoning; West Publishing Company; 765-815.
3 O W Holmes; "The Common Law"; 1881.
4 R E Susskind; "Expert Systems in Law: A Jurisprudential Approach to Artificial Intelligence and Legal Reasoning"; 1986; The Modern Law Review; vol. 49; 168-194.
5 P Leith; "Artificial Intelligence and the Legal Rule"; 1986; Technology in Legal Education, University of Warwick, April 10th- 11th.
6 H Fiedler; "Expert Systems as a Tool for Drafting Legal Decisions"; 1985; Logica, Informatica, Diritto; Florence; 265- 274.
7 L Allen, C Saxon; "Computer-Aided Normalising and Unpacking: Some Interesting Machine-Processable Transformations of Legal Rules"; 1985; Computing Power and Legal reasoning; West Publishing Company; 495-572.
8 E L Rissland; "A1 and Legal Reasoning"; 1985; International Joint Conference on Artificial Intelligence; Los Angeles; 1254-1260.
9 N MacCormick; "Legal Reasoning and Legal Theory"; 1978; Oxford University Press.
10 H L A Hart; "The Concept of Law"; 1961; Oxford University Press.
11 E A Nida; "Componential Analysis of Meaning"; 1975; Mouton Publishers.
12 H J Levesque, R J Brachman; "A Fundamental Tradeoff in Knowledge Representation and Reasoning"; 1985; Readings in Knowledge Representation; Morgan Kaufmann Publishers; 41-70.
13 J A Robinson; "A Machine-Oriented Logic Based on the Resolution Principle"; 1965; Journal of the Association for Computing Machinery; vol.12; 23-41.
14 P Hammond, M Sergot; "A Prolog Shell for Logic Based Expert Systems"; 1983; Imperial College of Science and Technology; internal report.
15 M Sergot, et al.; "The British Nationality Act as a Logic Program"; 1986; Communications of the ACM; vol. 29; no. 5; 370- 386.
16 *Op cit*
17 T F Gordon; "Object-Oriented Predicate Logic and its Role in Representing

Legal Knowledge"; 1985; Computing Power and Legal Reasoning; West Publishing Company; 163-203.

18 *Op cit*

19 T F Gordon; "Oblog Reference Manual"; 1986; Gesellschaft fur Mathematik und Datenverarbeitung; forthcoming.