Int. J. Man–Machine Studies (1991) **35**, 69–93

Hierarchical formalizations†

TOM ROUTEN

Department of Computing Science, Leicester Polytechnic, P.O. Box 143, Leicester LE1 9BH, UK

TREVOR BENCH-CAPON

Department of Computer Science, University of Liverpool, PO Box 147, Liverpool, L69 3BX, UK

This paper examines the prospects for using logic to represent legislation. This is important since it offers, via the technology of logic programming, a straightforward way of constructing knowledge-based systems in the legal domain. We suggest requirements which an ideal logical representation would satisfy and find that there is an apparent tension between two of them. Specifically, the need to produce a logically correct representation can appear to work against the need to produce a representation which is also easy to validate and maintain. This conflict, and other tensions which the use of logic is seen to create, have led some researchers to advocate the abandonment of logic. However, we argue that the tensions are created by the assumption that a logically correct representation will be one in which no meta-level features are represented. This assumption is encouraged by previous practice but is erroneous. Relaxing the assumption not only permits software engineering considerations to be respected but eases representational difficulties. The penalty is that the resulting formal representations can no longer serve as simple logic programs.

1. Formalizations of legislation

In recent years, one popular method of building knowledge-based programs in the legal domain has been to have a *formalization* of the complex definitions contained in relevant legislation at the heart of the system. The idea is to take a statute and to arrive at statements expressed in a formal language which correspond so well to the statements expressed in the statute, that anything one could determine as being a consequence of the statute for any given case, could also, given appropriate inference procedures, be derived mechanically from the formal statements. This approach is in contrast with the use of more traditional knowledge elicitation techniques. The origins of the formalization approach can be traced back through McCarty (1977) to Allen (1957), but perhaps the work on the British Nationality Act, reported in Sergot, Cory, Kowalski, Kriwacek, Hammond and Sadri (1986), is most responsible for its current popularity. The formal language which has tended to be adopted for these projects has been standard propositional or predicate logic, or a closely related formalism such as Horn clause logic or the programming language, Prolog. The use of these different formalisms means that there can be a number of different things meant by the term "formalization". In addition, there are a number of different motivations behind producing a formalization. It will serve to clarify the

† This paper is an elaborated version of Routen (1989).

0020-7373/91/010069 + 25$03.00/0

focus of this paper if we distinguish the different kinds of formalization and the different kinds of motivation behind them.

Layman Allen's original reason for formalizing legislation was to disambiguate it. The natural language in which legislation is couched is typically susceptible to a variety of interpretations, but the discipline of formalizing it forces one of these interpretations to be selected. Allen was particularly concerned with the import of such quasi-logical connectives as "unless", and the pervasive ambiguity in legislation between "if" and "if and only if". For this kind of disambiguation a formalization in propositional logic will often suffice, and no thought need be given to mechanical derivations. *Normalization,* as Allen calls this process, can be done as a first step whatever the target knowledge representation of the ultimate system, and it is a helpful first step. It is, after all, wise to be sure exactly what it is that is to be represented before attempting to represent it. This style of normalization has been used to clarify legislation before going further in formalization; for example, the ESPLEX project described in Biagioli, Mariani and Tiscornia (1987).

A second reason for formalizing legislation can be exemplified by the work done at Imperial College on the British Nationality Act, reported in Sergot *et al.* (1986) and Supplementary Benefit legislation, reported in Bench-Capon, Robinson, Routen and Sergot (1987). This work comes from within the tradition of logic programming which argues that the way to build a problem solving system is to represent the knowledge pertinent to the problem, and allow the control necessary to produce the problem solving algorithm to be supplied by a theorem prover capable of deriving the deductive consequences of that knowledge. Thus the British Nationality Act *program* is no more than a formalization of the Act in the Horn Clause subset of first-order predicate calculus. This formalization can perform as an expert system when embedded in an expert system shell such as APES (Hammond & Sergot, 1984) which supplies the necessary control component. That the resulting system appears to work so well as an expert system is due to some special factors of the British Nationality Act which are discussed in Bench-Capon (*1988a*), and since these special factors may not apply in every case, such an approach will in general only give a first approximation to be a practical system. This point is also stressed in Bench-Capon *et al.* (1987), which draws on the experience of applying this approach to very different legislation: the Law Relating to Supplementary Benefit. The prime aim of this work was not, however, to produce a legal expert system, but rather to explore legal knowledge representation and logic programming.

A third reason for formalization is illustrated by programs such as the formalization of the tax law of Canada described in Sherman (1987). Here the motivation is not theoretical but simply to produce an operationally satisfactory program. Noting the correspondence in form between the legislation and the clauses of a Prolog program, one might think that this language is peculiarly well-suited for building a program based on the legislation. Here, however, restrictions need not be placed on the use of extra-logical predicates, or the exploitation of Prolog's control strategy, as they must when the formalization is meant to be a logical one. The resulting program may not be declarative, and one might on that ground deny it the status of a formalization at all.

The final reason for formalizing legislation that we shall discuss, and one which is often overlooked, is rather more pragmatic than those discussed above, and derives from notions of software engineering. If knowledge-based systems in the legal domain are to be used in practice, they must be capable of acceptable validation, and susceptible to relatively straightforward maintenance if, or rather when, the underlying legislation changes. It may be that basing a program on a direct representation of the relevant legislation is the best means for achieving these features. This software-engineering-motivated approach is explained in detail in Bench-Capon (1988*b*), but can be illustrated here with a simple example.

The UK Social Security Act (1986) states:

> 39(1) Subject to the provisions of this Act-
> (a) a person who was over pensionable age on 5th July 1948 and satisfies such other conditions as may be prescribed shall be entitled to a Category C retirement pension at the appropriate weekly rate.

To interpret this we need also to bear in mind:

> 27(1) In this Act "pensionable age" means-
> (a) in the case of a man, the age of 65 years; and
> (b) in the case of a woman, the age of 60 years.

If we had been constructing an expert system on classical lines, relying on the ability to extract knowledge from an expert, in, say, August 1978, and we had asked our expert which people were entitled to a Category C retirement pension, he might well have said something like "men over 95 and women over 90". If we represented this information, however, the system would cease to be correct within the year. Whilst is would be plausible to pay a benefit to anyone over a certain age, in fact it is not age but date of birth that is crucial here, and whilst this is clear from a formalization of the legislation, it might well be lost in the summary of expertise extracted from an expert. Moreover, it would be dangerous in the formalization of 39(1)(a) to replace pensionable age by the date of birth indicated by the definition in 27(1)(a) and (b). For if this latter section were changed, perhaps to provide a common pensionable age, this change would not percolate through to our representation of the conditions for Category C retirement pension, and there would be no indication that the part of the program representing 39(1)(a) needed updating. Thus for maintenance purposes it is clearly advantageous to represent the legislation, and to represent it as faithfully as possible.

These points illustrate the easier maintenance of a program rooted in a formalization of the legislation, but we can make similar points with reference to validation. References to explicit dates of birth would require several pieces of legislation to be considered at once, and the performance of a calculation, whereas a more faithful formalization would permit direct and piecemeal comparison of sections to their formalization. Therefore we believe that a formalization should be put at the centre of a legal knowledge based system, not because this is the only way to produce such programs, but because programs written in this way will be better engineered, and will consequently produce more practically applicable programs. It is the best way to produce such programs not, or not only, from a theoretical perspective, but from a practical perspective.

Not all "formalizations" present us with these software engineering benefits. They do not accrue unless the formalization is disciplined in certain respects. Thus, for example, they do not attach to a program like Sherman's, since in this kind of formalization it would be perfectly permissible to substitute, in the above example, the explicit dates of birth for "pensionable age" when encoding 39(1)(a), and it would also be likely to be convenient for the sake of efficiency to do so.

It is probably worthwhile at this stage to point out that no one, to our knowledge, takes the extreme position that the deduction of logical consequences of a formalization is a correct and exhaustive model of legal reasoning, although this position has been attributed to proponents of the formalization approach by critics such as Greenleaf, Mowbray and Tyree (1987). What is proposed is not that the deduction of consequences is the whole story, but that it is useful to underpin any system based on legislation by a formalization of the relevant legislation, and that this, together with a means of deducing consequences from that formalization can form a valuable component in any AI system in the legal domain. The user will then be aware of the logical consequences, although what he chooses to do with this information is a separate matter.

2. A tension between requirements

2.1. REQUIREMENTS OF A FORMALIZATION

Let us try to make explicit the characteristics which an ideal formalization of a piece of legislation would exhibit. What we are saying above is that ideally a formalization should not only (1) be a faithful representation of what is expressed by the legislation; (2) be computationally adequate, i.e. should permit us to make all relevant derivations by machine; but also (3) be easy to validate and maintain. We might expand the third requirement into a cluster of desirable features. Perhaps the most important of these would be that the structure of the formal representation should resemble the structure of the original text.

With these points before us, we might discern a tension between 1 and the expanded 3. 1 states that a formalization should aim to capture the *content* of the legislation. Logical sophistication tells us that content is independent of the means by which it is expressed. Indeed the primary function of logic has been to help people overcome the imprecision and obfuscation of everyday means of expression in making clear what is being said. Consequently, to have a requirement which says that a formalization should respect the original means of expression would appear to work counter to the satisfaction of the most basic requirement: that we satisfactorily mine the *content* of the statute and represent it faithfully.

What is the content of a statute? The intention behind a statute is to establish the precise characteristics of properties and relationships which, were they found to hold of or between individuals, would be the signal that the individuals concerned could be afforded legal protection or could be penalized. For example, the UK's Housing Act 1985 defines what it is for a tenancy to be a secure tenancy. If a tenancy is a secure tenancy, then the tenant has greater rights than if it is not. It is natural to suppose then that the content of this statute can be expressed as a set of statements all of which *concern* elements of the housing domain. "Concern" is italicized

because we intend it to have a precise meaning. In logic, statements are conceived of in terms of objects and the properties or relationships which the statement asserts to be true of them. A statement *concerns* whatever objects it contains. By "the housing domain" we mean the set of "real-world" objects one would expect a Housing Act to be talking about. This would contain physical objects such as houses, landlords, tenants, but also non-physical objects such as tenancy agreements. "Real-world" is of course not well-defined and we use it here essentially to contrast such objects with statements contained in the statute. Here is an example of a statement in the statute which concerns elements of the housing domain:

> 79-(1) A tenancy under which a dwelling house is let as a secure separate dwelling is a secure tenancy at any time when . . . the landlord condition and the tenant condition are satisfied.

However, the Housing Act also contains statements which do not concern real-world objects but statements, or sets of statements. These kinds of statements are at said to be at the meta-level with respect to statements of the previous kind. The next subsection provides an example:

> 79-(2) Subsection (1) has effect subject to-
> (a) the exception in schedule 1 . .
> (b) sections 89(3) and (4) and 90(3) . . and
> (c) sections 91(2) and 93(2)

This is a *bone fide* statement: it has content, and that content can be judged to be true or false. What the statement expresses appears to be just as much part of the content of the statute as that which is expressed by 79-(1). Does this mean that the content of the Housing Act 1985 cannot after all be expressed in a set of statements all of which concern elements of the housing domain? Our logical intuitions tell us that although 79-(2) has content, it is inessential from the point of view of defining security of tenure. The meta-level statement is merely part of the contingent means that the drafters have chosen to express a complex definiton. The definition of security of tenure could have been made more "logical" by locating all exceptions within the general rule and thereby eliminating the need for this meta-level statement. This is precisely what we would be inclined to do in a formalization of the legislation. If we were trying to formalize 79-(1) in Prolog, our first effort might mirror precisely the content of that subsection considered on its own:

```
secure_tenancy(Tenancy) ←
  satisfies_landlord_condition(Tenancy),
  satisfies_tenant_condition(Tenancy).
```

The meta-level statement (79-(2)) states that there are exceptions to this general rule. One exception to the rule is that "a tenancy is not a secure tenancy if it is a long tenancy". We could incorporate this exception into our formalization of 79-(1) as follows:

```
secure_tenancy(Tenancy) ←
  satisfies_landlord_condition(Tenancy),
  satisfies_tenant_condition(Tenancy).
  not long_tenancy(Tenancy).
```

In the same way, we could incorporate all the exceptions and qualifications mentioned in 79-(2) into a new object-level rule, extending it to make it exception*less*. In this way, we would be able to construct a logically correct formalization which did not require explicit representation of meta-rules such as 79-(2). One would be tempted to say that this kind of formalization had captured the "logic" of the definition.

The tension between logic and software engineering has shown itself in this example, since in our quest to capture the logic of the definition we have moved from a first effort which was comprehensible and mapped well onto its textual source to a complex exceptionless rule whose content is derived from many disparate parts of the text.

Since to make the general rule into an exceptionless rule may be to create an horrendously complicated rule, there would be a natural inclination to introduce conditions within the definition which correspond to groups of exceptions rather than just one. It would be natural if the predicate names chosen for these conditions would refer to the location of the exceptions in the text.

```
secure_tenancy(Tenancy) ←
  satisfies_landlord_condition(Tenancy),
  satisfies_tenant_condition(Tenancy).
  not satisfies_exceptions_in_sched-1(Tenancy).
```

This would permit a formalization which did not require explicit representation of the meta-level rule, and also which did not suffer from over-complex rules. The penalty of doing this is that mention of the text intrudes into our definition, and it is therefore less convincingly the pure "logic" of the definition. After all, when politicians and civil servants were deciding what should or should not qualify as a secure tenancy, they did not consider the "satisfication of exceptions in schedule 1" to be significant since, schedule 1 of the Act they were in the process of writing could obviously not have existed at that time. This technique has been disparaged for this kind of reason by, for example, Gordon (1987, p. 62) states that: "it is not our purpose to model the statute, but the law represented by the statute; referring to code section numbers in rules blurs this distinction". Despite this, pragmatically speaking, it may be an attractive compromise in many cases.

3. Logical models and flat formalizations

Recently, some have argued that because of these difficulties, and others which we will examine below, classical first-order predicate calculus (hereafter, following Gordon, "standard logic") is an inappropriate formalism in which to produce formalizations (Gordon, 1987; Berman & Hafner, 1987). Their view appears to be that representing legislation requires one to have meta-level statements in the formalization, but that standard logic requires these to be eliminated in the fashion described above. "Logic does not allow explicit statements to be made about its own rules. Yet such 'meta-level' statements are frequently found in legal discourse". (Berman & Hafner (1987) p. 3) "Standard predicate logic and Prolog require that systems of general rules and exceptions be *collapsed* into logically complete formulas" (Gordon, 1987, p. 61). This leads Gordon, for example, to offer a new,

ad hoc, formalism. There are strong advantages in using standard logic for the task if possible because of its universality, naturalness, and the fact that there are well-understood mechanical inference procedures for it. We would agree that *if* the use of standard logic did not permit meta-level features in a formalization then it may well be inappropriate, but it is mistaken to think that it does not. It does not follow from this that standard logic *is* the most appropriate formal language to use, or even that it is appropriate at all. However, we believe that at this stage there is credibility gap between the arguments for abandoning the use of logic and their conclusion. While this paper will produce further considerations which may serve to cement the gap a little more, we shall argue that the gap still remains.

The origin of the view that a logic-based formalization cannot include meta-level statements may lie in the work on the British Nationality Act. Here, a strong analogy is made between statute-based reasoning and logico-mathematical reasoning. The rules contained in the statute are thought of as *axioms,* the complete set constituting an *axiomatic theory,* along the lines of Euclidian geometry (Sergot *et al.,* 1986, p. 46). It is suggested that all rules in the formalization of the British Nationality Act appear as axioms in a single theory, much as a set of premises in a very long argument. The term *logical model* has been introduced to refer to such axiomatic theories (Kowalski & Sergot, 1986; Bench-Capon, 1988*a*). "Model" in logic has a technical meaning which the phrase "logical model of legislation" is evidently appealing to. Any particular formalization of a statute in logic will be such that a number of predicate symbols will have been used and a number of constants will also have been used. Together with the usual machinery of the predicate calculus, we can consider these expressions to constitute a language, let us call the language *L.* An *interpretation* of the statements in the formalization is the association of an object with each of the constants in *L* and the association of a property or relation with each of the predicate symbols in *L.* An interpretation is a *model* of the formalization if all of the statements in the formalization are true under that interpretation. The phrase "logical model" and the Euclidean analogy strongly suggest, and indeed an ideal use of logic would so require that in order to achieve a formalization one must devise a single set of statements and an interpretation of the symbols used such that not only are the statements all true (i.e. the interpretation is a model), and not only do the statements faithfully represent what is expressed in the statute, but also that all of the statements concern real-world objects, and predicate symbols are associated with real-world relationships.

This would not be the case if the formalization utilized a meta-language. Then the "logical model" could not be of the elegant and intuitive kind described above which "says something" about the world, and it would not then be possible to liken it to the axiomatization of Euclidean geometry. Instead, the model would be arrived at by associating objects and relations with the constants and predicate symbols of the meta-language, but now those objects would be statements contained in the statute, rather than real-world objects such as houses, and the relations would be those holding between statements rather than real-world relationships. Of course, the object-level statements must also be given a meaning (an interpretation), but it does not follow from this that we would have two logical models, one at the meta-level and one Euclidean-style model at the object-level. This is because the set of object-level statements we are considering include general rules and their

exceptions, and it would be inappropriate to try to find an interpretation under which the rules and their exceptions are all true. It would be necessary to partition the set into a number of small theories, such that we can construct a model for each. The general rule 79-(1) would constitute one theory, the exceptions contained in 89-(3) another, the exceptions contained in schedule 1 another, and so on. Far from the formalization of a statute being a single logical model, it would be a complex hierarchy of models.

The important thing to note is that there is nothing logically illegitimate in using logic in this (admittedly less appealing) manner. There are simply no restrictions on the kinds of objects which can be associated with constants. One should be careful to distinguish second-order logic, which permits quantification over properties from the use of first-order logic as a meta-language whose objects are expressions of an object-language. Whilst quantification over properties would alter the nature of the logic significantly, simply altering the domain of a language from a set of objects such as houses, landlords and tenants, to a set of objects such as statements and theories does not alter the semantic conventions of the formalism employed. We should be careful too when Berman and Hafner (1987) state "Logic does not allow explicit statements to be made about its own rules". The thought underlying this remark is perhaps that unless an exclusive distinction is made between object-level and meta-level then self-referential statements are possible and some of these (such as "this statement is false") can lead to paradox. But this kind of self-reference is not likely to exist in statutes. The existence of cross-references does make a statute, when considered as a whole, self-referential, but this is quite a different, less dangerous kind of "self-reference". We are suggesting that the content of a statute can be modelled in a hierarchy of theories, none of which contain statements which refer to themselves. The self-referential nature of a statute derives that there are statements within the statute, which refer to statements which are also contained in the statute, but logically speaking they are in a different language.

What we have established so far is that there is a tension between the straightforward intuitive application of the predicate calculus and the production of a formalization which is a good one from a software engineering perspective. We have seen that this tension is in part responsible for the view that logic is not the right vehicle for such a task, but that this view depends on an erroneous view of logic. The conclusion at this stage of the argument, one might suppose, is that software engineering considerations must go by the board. The most important requirement is that the formalization be a logically correct representation of what the statute expresses. We have seen that the introduction of a meta-language entails the abandonment of the most intuitively satisfying use of logic which promised to capture the thoughts of the original drafters of the statute most satisfactorily. However, we shall now present reasons why someone may wish to think twice before making a decision to eliminate meta-level statements in order to create what Gordon calls a "flat formalization".

4. Difficulties with flat formalizations

4.1. GORDON'S POINTS

Thomas Gordon has provided arguments in favour of maintaining the separation in a formalization between general rules and their exceptions. His points are basically

twofold and they are both concerned to show that a formalization in which rules and exceptions are collapsed together would not be able to support important aspects of legal reasoning. Firstly, he says that incorporating exceptions within general rules can create, as we have remarked, incomprehensible rules in the formalization and that this will have a detrimental effect on the capability of a knowledge-based system based on such a formalization to explain itself. He states:

> 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" (Gordon (1987), p. 61).

Secondly, Gordon makes the point that this kind of formalization would not permit an expert system built upon it to answer a question regarding the applicability of a general rule without extracting from the user information concerning every exception to it mentioned in the statute. Separating rules and exceptions would allow a system to simulate legal decision-making processes more convincingly than this. For instance, not all exceptions to a rule need always be considered for it would sometimes make a decision dependent on the acquisition of too much information. In statutes, general rules can have many exceptions which are rarely if ever fulfilled but represent the plugging of loopholes found in earlier drafts. As an example, we can note that, in order to determine a person's entitlement to supplementary benefit, an advice-giving system based on a "flat" formalization of the Law Relating to Supplementary Benefit would have needed to ask of perfectly unexceptionable individuals whether they were members of a wide range of minority groups such as refugees, share fishermen, and many more, since members of those groups receive special treatment.

We can envisage further benefits in maintaining the separation. For example, should we want an expert system to be able to present relevant parts of the original legislative texts at relevant points in its interaction with the user, perhaps as part of its explanation capability, then the system would need to be able to work out which part of the text to present. If the rules it is reasoning with are composed of content from many parts of the statute this will be difficult. If instead the rules map onto the structure of the text naturally, then this kind of capability could be implemented elegantly.

4.2. KNOWLEDGE REPRESENTATION PROBLEMS

Work within the "logical model" tradition has identified knowledge representation problems: problems in formulating statements of logic which capture the meaning expressed in statutes using certain typical linguistic constructions. It may be that many of these difficulties exist only because of the implicit acceptance of the assumption that the statements of logic must always be solely at the object-level. Here we shall examine three of those problems and it will become clear how much easier the task of representing them is made once a meta-level of representation is permitted. Let us suppose then that we have two languages, an object-language and a meta-language. The meta-language can be used to construct statements which concern elements of the object language. Consequently, our meta-language requires

a way of referring to statements of the object-language. Here we shall not present the exposition of a naming convention and will build names for statements using the same characters surrounded with single quotes. Predicates of the meta-language will appear in uppercase to distinguish them from predicates of the object-language.

4.2.1. Rules and exceptions

The first of the problems is that which we have been discussing, the problem of maintaining the separation between a general rule and its exceptions. We have already looked at an example of a general rule 79(1) of the Housing Act which is subject to textually separate exceptions, and as we have seen, the next sub-section, 79(2), details where exceptions to the general rule are to be found. Instead of compiling the meta-level 79(2) down into amendments to object-level statements we could take seriously the idea that it is a statement at the meta-level. If we were to represent 79(2) naturally and directly we would need to use a meta-level predicate representing the relation it contains.

Let SUBJECT-TO be a predicate of the meta-language. Not only do we need some way of naming the single statements of the object-language, but also sets of statements (small theories) contained in subsections 89(3), 89(4) etc., so that we could describe the dependencies asserted by 79(2) in formal statements. If we use statute section names to denote these sets of statements, we can represent the 79(2) as a set of assertions at the meta-level:

SUBJECT-TO('79(1)', '89(3)')
SUBJECT-TO('79(1)', 'sched-1') ... etc.

We could then leave the general rule expressed in 79(1) as it is, with all the attendant software engineering and legal reasoning advantages mentioned above.

4.2.2. Deeming provisions

The second problem we shall look at has been called that of the "deeming provision". Deeming provisions allow things which are not true to be treated as if they were. In Bench-Capon *et al* (1987, p. 197), an instance of such a provision from the Law Relating to Supplementary Benefit is quoted:

> a person shall be treated as blind if he regained his eyesight within the previous six months...

To understand why deeming provisions are a problem, suppose we had a statute containing just the above deeming provision plus the rule which says that "a person who is blind is entitled to disability benefit". A formal version of the rule would no doubt look something like the following:

entitled(X) ← blind(X).

As it stands, this definition of entitlement does not allow a person who has regained his or her sight in the past six months to be entitled. The problem of representing the deeming provision is the problem of modifying or adding to the rule in a natural and convenient manner which at the same time achieves the consequences which the

deeming provision is intended to achieve. If achieving the right consequences were all, we could straightforwardly add to the definition of entitlement:

entitled(X) ← regained_sight_etc(X).

Comprehensibility of the formalization is already compromised since this is not what the legislation says. In addition, the formalization is no longer easy to update and maintain since the chain of reasoning which led to the addition of this alternative definition of entitled may not be easy to reconstruct. The same points would hold with more force were we to add

blind(X) ← regained_sight_etc(X).

because it would allow us to derive contradictory predications of the same individual—that he has sight and that he doesn't. A better suggestion (Bench-Capon *et al.* 1987, p. 197) would be to introduce a new predicate (let's call it blind*) which is defined in terms of blind and in terms of the deeming provision:

blind*(x) ← blind(X).
blind*(X) ← regained_sight_etc(X).

Then, by replacing blind with blind* in the definition of entitled, we would have achieved the intended meaning of entitlement. This is not entirely *ad hoc* because one can argue that when the legislation uses the word "blind", it implicitly appeals to a concept other than the real-world concept of blindness (because of the effect of the deeming provision). This technical legal concept doesn't have a name already and blind* is as good a name as any.

However, the proliferation of such peculiar concepts is not good from the point of view of legal validation. One could just as easily argue that the whole reason for the existence of deeming provisions is to allow the drafter to avoid having to introduce such concepts when defining something with great precision, and instead couch definitions in terms of ordinary concepts with patches. The above treatment of deeming provisions also creates a potential problem for the development of the software: it means that to update a formalization by adding a deeming provision could entail having to replace many occurrences of a predicate denoting a real-world concept by another denoting a related technical one. This is not always a simple matter—for example it could have profound effects on the explanation capability of an expert system based on the formalization—and is a complication to be avoided if possible.

That fact that there are problems in arriving at a satisfactory formulation should not be surprising since so far we have considered only attempts to represent at the object-level something which is more naturally interpreted as making a meta-level statement. We have said that deeming provisions are there to permit us to "treat something as true" and this is clearly a meta-level notion. Many if not all deeming provisions are more naturally interpreted as meta-level statements.

Let us introduce the following meta-level predicates, predicates whose arguments

are statements of the object-language, to help us to represent the deeming provision under consideration. To simplify exposition we shall frame the discussion as if we have a single object-level theory, rather than a number of such different theories, as has been suggested above.

DEEMED(th, S) "The statement S is treated as if it were true in the object-level theory th".

PROVE1(th, S) "The statement S is proveable in the object-level theory th"

What we want to say is that all object-level statements of the form blind(α) (where α is any object-level constant) are to be treated as if they are true if regained_sight_etc(α) is proveable. This means something like:

DEEMED(th, 'blind (x)') ←
PROVE1(th, 'regained_sight_etc(x)').†

This is not yet correct since we should be able to conclude blind(α) not only if regained_sight_etc(α) is derivable at the object-level, but also, for example, if there was a deeming provision which said that we should treat it as true, that is if it was derivable at the meta-level that DEEMED (th, 'regained_sight_etc(α)'). It would be possible to add another clause to the representation of the deeming provision

DEEMED(th, 'blind(x)') ←
DEEMED(th, 'regained_sight_etc(x)')

At some stage however, we shall want to represent the fundamental reason for having deeming provisions, that is that our peculiar overall notion of proveable (perhaps "proveable according to the statute") is going to be that some proposition, p, is true if a statement of the form PROVE1(th, p) is derivable at the meta-level, but also if a statement of the formed DEEMED(th, p) is derivable at the meta-level. Therefore, it is clear that we require a *further* meta-level of representation; one at which we can talk about derivability at the first meta-level. Predicates at the second meta-level shall be in uppercase and italicized. Let us introduce the following two:

STATUTE-PROVE(S) "statement S is proveable according to the statute"

*PROVE*2(m1, S) "statement S is proveable in the theory at the first level meta-level m1"

STATUTE-PROVE(S) ←
  *PROVE*2(m1, 'PROVE1(th, S)').
STATUTE-PROVE(S) ←
  *PROVE*2(m1, 'DEEMED(th, S)').

† Here the apparent variable, x, is not really a variable at all but merely part of each of the two unstructured names of object-level statements, and the two occurences of x do not have any relationship. To effect the definition properly we would need to have meta-level predicates which would permit us to reason about the object-level statements referred to by the names. Perlis (1985) calls this "un-naming". Suppose we had the following predicate:

EXPRESSION(P, O, S) "S is the statement referred to by the concatenation P + '('+O+')'"

The definition of the deeming provision would then look something like this:

DEEMED(th, S1) ←
EXPRESSION('blind', O, S1),
EXPRESSION('regained_sight_etc', O, S2),
PROVE1(th, S2).

With these predicates we would want to represent our deeming provision as follows.

DEEMED(th, 'blind(x)') ←
STATUTE-PROVE('regained_sight_etc(x)').

This definition mixes levels. Its conclusion is at the first meta-level and its condition is at the second meta-level. This kind of representation can be achieved in amalgamation of a meta-language with its object-language via "reflection principles" (Bowen & Kowalski, 1982), rules which "communicate" results of the meta-language to the object-language and vice versa.† With this definition, the representation of the general rule for entitlement remains unaltered. We have no peculiar predicates and the deeming provision is explicit.

4.2.3. Counterfactuals

Before looking at the third representational difficulty, how one might represent counterfactuals, we should remark that the prior question of their interpretation remains a puzzle.

The problem of counterfactuals is discussed in Ginsberg (1986), Routen and Bench-Capon (1986) and Bench-Capon (1989). Broadly, the problem centres on the fact that whilst the surface form of a counterfactual is very like that of a material implication, counterfactuals behave very differently in a number of ways.

Firstly, we cannot use contraposition with counterfactuals: the truth of

If the oven had not failed the dinner would have been on time

does not imply that

If the dinner had been late the power would have failed

since the dinner can be late for a whole host of reasons. Secondly, counterfactuals are not transitive. The following two counterfactuals may be true:

If James Bond had been born in Russia he would have been a Communist.
If James Bond had been a Communist he would have been a traitor.

We cannot, however, conclude from them that

If James Bond had been born in Russia he would have been a traitor.

On the contrary he would probably have served his country loyally as an agent of the KGB. Thirdly counterfactuals are non-monotonic; the following pair of counterfactuals can both be true:

If Bert had come the party would have been lively
If Bert and Carol had come the party would have been dreary,

since Bert is only the life and soul of a party when free from his wife Carol's restraining glares. Finally there is a pervasive problem with counterfactuals such as

If Dover were in Yorkshire, then Dover would be north of London
If Dover were in Yorkshire, then Yorkshire would be in the South of England

† This amalgamation presents the possibility of paradox mentioned earlier but, in the AI literature, see Perlis (1985) for the argument that paradox can be avoided in such a first-order system.

since we have no clear grounds of deciding which, if either, is true. The problem is quite pressing when representing legislation, since the counterfactual is quite a favourite construction of legal draftsmen.

To be able to pursue knowledge representation issues, let us look at an example of a counterfactual whose meaning is relatively transparent. A simplified definition of entitlement to supplementary benefit which was contained in the Law Relating to Supplementary Benefit can be represented as follows:

```
entitled(x, supp_ben) ←
  poor(x),
  in_uk(x),
  unemployed(x).
```

The law also allows that people can be entitled to benefit when they are away from the UK on holiday. Let us call this benefit "holiday benefit". A person is entitled to holiday benefit if he "would be entitled to supplementary benefit if he were in the UK".† The meaning of this condition appears to be clear. The conditions under which someone would be entitled to supplementary benefit if he were in the UK are precisely the conditions defining entitlement to supplementary benefit minus the condition that the person is in the UK. Therefore, the following would appear to serve to represent entitlement to holiday benefit:

```
entitled(x, holiday_ben) ←
  poor(x),
  unemployed(x).
```

We shall remark on a fundamental problem with this treatment below. For the moment, we might note a less important though still undesirable consequence for the software engineering point of view. Suppose that there were many more conditions defining entitlement to supplementary benefit, and that there were several alternative benefits defined counterfactually with respect to entitlement to supplementary benefit instead of the one. Treating the counterfactual definitions in the way suggested above would result in a formalization containing several definitions all long, and yet extremely similar. The drafter managed to avoid this duplication and so should our formalization. The general point here is that meta-level statements can be shorthand devices, the advantages of which are lost if the meta-level statements are eliminated by reduction.

The introduction of a meta-level predicate would ease our task. Let us call it WOULDBE. WOULDBE takes two statements as its arguments: WOULDBE(th, C, A) is read: "C would be derivable in the object-level theory were it the case that A was derivable" which relates the antecedent and the consequent of the counterfactual. The definition of entitlement to holiday benefit would now look like this:

```
entitled(x, holiday_ben) ←
  WOULDBE(th, 'entitled(x, supp_ben)', 'in_uk(x)')
```

† This too is slightly simplified for purposes of exposition. The full definition is given in Routen and Bench-Capon (1986).

In this way we avoid having to duplicate the definition, and the counterfactual is explicit.

We have seen three constructions which have appeared in the literature as presenting problems of knowledge representation. They have a common feature: that they are constructions used to express things at the meta-level. Relaxing the requirement that meta-level features are reduced to amendments to object-level statements facilitates the resolution of these problems. We would not want to pretend that the resolutions offered above, particularly in the case of the deeming provision, constitute an illustration of how easy it is to represent things in standard logic. From a practical point of view, the importance of the formalizations above is that the meta-level representations they employ make clearer the logical structure of the problematic constructs and show that the difficulties in constructing equivalent object-level statements are only to be expected. We shall see shortly that matters are made considerably easier when we are interested only in approximating these representations in Prolog.

4.3. FLAT FORMALIZATIONS ARE IMPOVERISHED TRANSLATIONS

Let us recap. The attempt to model the content of a statute in a theory composed of a single linear set of axioms creates representational difficulties, software engineering difficulties, leads to inflated knowledge bases, hinders the simulation of legal reasoning, and the indexing of formal rules with original source material. Yet one *still* might want to say that these considerations are secondary. One might still want to argue that requirement 1 (of section 2.1) should be our guiding light, and that this demands that all axioms in our formalization concern the law; there should be none which concern the statute. In this section we'll see that, paradoxically, even this requirement can in fact demand the contrary.

We can treat it as self-evident that the most basic constraint is that the formalization be a correct representation of the logical content of the statute. One might call a translation in which content existing in the source has been lost, an *impoverished translation* (Studnicki, 1985). If we take this constraint seriously, then it can be the case that a formalization *must not* be a single axiomatic theory. We shall describe two ways of demonstrating that such "flat" formalizations are necessarily impoverished translations.

The first way is to show the existence of irreducibly meta-level features in statutes. Counterfactuals are such *essentially* meta-linguistic features. They are not wholly reducible to object-level statements. We have argued elsewhere that they are in fact not statements at all but are devices for presenting arguments which have suppressed premises relating the counterfactual's antecedent with its consequent (Bench-Capon, 1989). If one can be said to be making a statement at all in asserting a counterfactual, one is making a statement of existence: i.e. that there exists a set of true statements from which one can derive the counterfactual's consequent once the truth of its antecedent is assumed. In compiling a counterfactual down to a single set of object-level statements then, one is inevitably failing to represent accurately its truth-conditions since the counterfactual, interpreted as an existence

statement, could be satisfied by an alternative set of object-level statements.† To illustrate the effect of this we can refer back to our example counterfactual and note that legislators may decide to alter the definition of entitlement to supplementary benefit and yet have no desire to alter the definition of entitlement to holiday benefit. Thus, while the intended meaning of the counterfactual used remains constant, its object-level representation has to change. From the software engineering point of view, this demonstrates the likely failure of modularity in a flat formalization.

The second way of showing that flat formalizations are necessarily impoverished translations is by pointing out that drafters of statutes can and often do distribute the definition of a concept across a number of fragments. This is the case when exceptions are kept distinct from a general rule, but also when concepts are defined disjunctively, and quite often when there are a number of conditions in a conjunctive definition. It is then possible to refer to and use in other contexts, single conditions, exceptions, disjuncts etc.. Common phrases such as ". . . the condition mentioned in section . . .", ". . . in virture of section . . .", ". . . within the meaning of section . . ." are all used to accomplish such reference.

For example, one might find that a statute asserts that a tenancy has a certain property if it fails to be a secure tenancy *in virtue of* 89(3) of the Housing Act 1985. If we had not stayed faithful to the text and kept the general rule of 79(1) and its exceptions distinct in our formal representation, and moreover had kept all exceptions distinct from each other, then we would be unable to represent this condition without adding to our formalization of the Housing Act an extra predicate such that the predicate is satisfied by a tenancy if the exceptions contained in 89(3) apply to it.‡ A flat formalization would not have maintained a distinct concept of "the condition contained in 89(3)" and so would have been an impoverished translation with respect to the statute. The importance of this impoverishment is perhaps dependent on the scale of one's ambitions. Just as the rigorous application of software engineering techniques appropriate to a large system is overkill for a very small program not intended for regular use, a small-scale one-off formalization in which it is possible to get to know all of the cross-references which will need to be formalized would not determine a need to worry about glueing all parts of a definition into one. On the other hand, if one were undertaking the formalization of a large piece of legislation, one which is susceptible to frequent amendment, or one which may be required to interact with formal representations of other statutes, then this point needs to be considered carefully. Statutes are hierarchical structures which hold meaning in their rafters and not only at ground level. That is, some of the essential content of a statute is embodied in the organization of the text as well as in its meaning.

5. A Prolog meta-interpreter

There are a couple of related points which remain to be addressed. Firstly, how can we use the representations offered in section 4.2 to make the relevant deductions?

† This fact also accounts for the ambiguity of some counterfactuals which is stressed in Routen and Bench-Capon (1986).

‡ This would cause duplication in the knowledge base.

Secondly, what is the relationship between the representations and the writing of a logic program which can use them.

We shall examine both by attempting to write a Prolog "meta-interpreter" which satisfactorily captures the definition of STATUTE-PROVE and which utilizes the representations given above. One should note that this is not the only conceivable way to produce a logic program which can handle features such as rules and exceptions. For example, Kowalski (1989) uses a program transformation technique to address similar concerns.

Consider a knowledge base consisting of the following three statements:

```
entitled(X) ← blind(X).
DEEMED(blind(x)) ←
  STATUTE-PROVE (regained_sight_etc(x)).
regained_sight_etc(bill).
```

It would appear that in satisfying the first and third requirements on a formalization outlined in section 2 (that a formalization be a faithful representation of what is expressed by the legislation, and that it be easy to validate and to maintain) our formalization fails to satisfy the second (that it should permit us to make all relevant derivations by machine) since, from the above knowledge base, a simple Prolog interpreter could derive nothing. What we wanted from our representation on the other hand was that it should enable us to conclude that entitled(bill) is true; to derive a statement of the form *STATUTE-PROVE*(entitled(bill)).

The specialized meta-interpreter we need to define for reasoning with statutes† can be an augmentation of a backward-reasoning meta-interpreter with a familiar top-level definition of the kind mentioned in Welham (1988, p. 293):

```
STATUTE-PROVE(KB, Goals) ←
  EMPTY(Goals).
STATUTE-PROVE(KB, Goals) ←
  SELECT(Goals, Goals, Rest),
  REDUCE(KB, Goal, SubGoals),
  COMBINE(Rest, SubGoals, NewGoals),
  STATUTE-PROVE(KB, NewGoals).
```

The standard definition of REDUCE is as follows: it is possible to reduce a goal to subgoals if there is a clause in the knowledge base whose conclusion matches the goal—the subgoals are then the suitably instantiated conditions of that rule.

```
REDUCE(KB, Goal, SubGoals) ←
  CLAUSE(KB, Goal, SubGoals).
```

5.1. DEEMING PROVISIONS

With the kind of reasoning involved in using deeming provisions, there is another way of reducing a goal to subgoals. One can reduce the problem of showing a proposition is (statute) proveable, not only if there is a clause in the knowledge base

† The one we describe here is "coarse-grained" since it exploits the underlying Prolog unification mechanism rather than making unification explicit in the definition. We are also using unification to simulate the naming and un-naming referred to above.

defining it, but also if there is a clause in the knowledge base which gives conditions under which one should treat if as being true:

```
REDUCE(KB, Goal, SubGoals) ←
  CLAUSE(KB, DEEMED(Goal), SubGoals).
```

5.2. COUNTERFACTUALS

We can similarly augment the meta-interpreter to handle counterfactuals. One of the most popular ways of interpreting counterfactuals is to follow the philosopher David Lewis (1973) and suggest that they are best understood as making statements about possible worlds. This interpretation has appeared in the A.I. literature (e.g. Ginsberg, 1986). We have given reasons for rejecting this account in Routen and Bench-Capon (1986), and an alternative view is elaborated there and in Bench-Capon (1989). The alternative view is that referred to above, that a counterfactual is properly construed as an elliptical argument, and states that there is some valid argument which has the antecedent of the counterfactual amongst its premises and the consequent as conclusion.

We can extend our definition of REDUCE to incorporate this interpretation in the following way:

```
REDUCE(KB, WOULDBE(C, A), SubGoals) ←
  CLAUSE(KB, C, Conditions),
  EXTRACT(KB, A, Conditions, SubGoals)
```

Here a counterfactual can be reduced to subgoals if there is a clause defining the antecedent of the counterfactual in terms of some conditions, and the *ex hypothesi* satisfaction of the consequent of the counterfactual is effected by removing its occurrence from the conditions to leave the remaining subgoals.† An alternative of REDUCE definition embodying the possible worlds approach to counterfactuals could be constructed, although we shall not attempt to do so here. The important thing to note is that these various attitudes to counterfactuals do not impact on the formalization, and the debate can be contained within the meta-interpreter, where its terms are at least formally stated.

5.3. RULES AND EXCEPTIONS

General rules and exceptions are often related by cross-referencing rules like 79(2) of the Housing Act. Where they are not related by an explicit cross-reference, they could be, since the rules and exceptions are invariably described in different paragraphs of the legislation. Therefore, if we define *STATUTE-PROVE* to allow us to handle cross-references like 79(2), it would provide us with a general method for formalizing rules and exceptions.

As we have remarked, representing cross-references with meta-level predicates (such as SUBJECT-TO(79(1), 89(3))) implies that one is able to refer successfully to sets of statements which represent formalizations of fragments of text (such as 89(3)). The most basic precondition for successful reference is that the object one wants to refer to exists. Since we do not know in advance what fragments we will

† Unfolding of conditions may be required to produce a set of conditions from which it is possible to extract the antecedent.

want to be able to refer to in this way, it must be possible, for any specified fragment of the statute, be it a sub-paragraph or a whole section, to extract from our formalization a set of statements which represents the content of that fragment. This constitutes a reiteration of the point we made above which said that a formalization which did not permit arbitrary cross-references to be made (or at least as many as does the original text) would be impoverished. The most convenient and perhaps the only way of developing a formalization in the light of this is by adopting the methodological principle that, as far as possible, the structure of one's formalization be faithful to the structure of the text. Ideally, there would be a one-to-one correspondance between rules in the formalization and indivisible fragments of the text. Cross-references will generally refer to fragments which *are* divisible, i.e. which contain a number of textual atoms. This means that we must tag each rule in our formalization with its source in order to be able to determine the make-up of the sets of statements referred to.† Consequently, we need to describe rules within a meta-level relation which relates source with content; perhaps:

RULE(KB, Source, Conclusion, Conditions)

For example, the general rule, 79-(1) and the exception from schedule 1 could be represented in the following two assertions:

```
RULE(housing-act-1985,
  [ha, V, 79, 1],
  secure_tenancy(Tenancy),
  [satisfies_landlord_condition(Tenancy),
  satisfies_tenant_condition(Tenancy)]).

RULE(housing-act-1985,
  [ha, sched-1],
  long_tenancy(Tenancy),
  [ ]).
```

We also require the definition of a predicate which would tell us, given two sources whether or not a particular fragment was included in another. For example, that subsection 79(2) is included within section 79.

CONTAINS(A, B) "Fragment A contains fragment B"

For our purposes it will be convenient if we consider a source to contain itself. We can now amend our definition of REDUCE as follows: one can reduce a goal to sub-goals only by finding a clause whose conclusion matches the goal *if* there does not exist a valid exception to the means by which the goal was reduced (represented by the source of the rule used).

```
REDUCE(KB, Goal, SubGoals) ←
  RULE(KB, Source, Goal, SubGoals),
  NOT EXCEPTION(KB, Source, Goal).
```

There exists a valid exception to the means by which a goal was reduced if there is a set of statements which constitute exceptions to the means by which it was reduced

† There are independent reasons for representing sources as we have remarked in 4.1.

and, after having substituted the relevant variable bindings contained in the goal to these statements, it is possible to show that any of them is (statute) true:

```
EXCEPTION(KB, Source, Goal) ←
  CONTAINS(OtherSource, Source),
  RULE(KB, AnySource, SUBJECT-TO(OtherSource, Exs), true),
  SUBSTITUTE(Goal, Exs, Exceptions),
  MEMBER(X, Exceptions),
  STATUTE-PROVE(KB, X).
```

Because *STATUTE-PROVE* is defined recursively, the meta-interpreter will check for exceptions to exceptions and so on, to the limit of exception-free exceptions. This would appear to negate one of the major benefits of maintaining the separation between rules and exceptions described above (that we would not always have to consider all exceptions to a rule). However, here we are describing just one possible meta-interpreter. The important thing is that rules and exceptions are separate at the object-level, making possible the implementation of the sophisticated inferencing features desired while still permitting one to obtain, if required, the same exhaustive behaviour characteristic of a flat formalization.

We have one final problem. The definition above, although it gives the general structure of the definition we shall require, is inadequate since there is in general no way that any SUBSTITUTE predicate of the kind we have supposed to exist can communicate the appropriate substitutions of values for variables between a goal and a set of statements which does not contain a term matching that goal. If rules and exceptions are represented entirely distinctly in the way we have described immediately above, then the rules and exceptions are not guaranteed to share terms which can unify.

Kowalski (1989) suggests that an exception C, to a rule $A \leftarrow B$ would normally be represented as a rule with a negative conclusion $-A \leftarrow C$. Clearly, if we represent exceptions in this way then our problem is solved: A and $-A$ can be unified to communicate the substitution. In doing this, we would give up one of the supposed advantages of separate rules and exceptions: that the same exception can serve as an exception to more than one rule, or even that an exception can function at one and the same time as an exception to one rule and a condition for another (as noted, drafters can achieve this by using complex cross-references such as "if x satisfies the condition mentioned in schedule 1, then . . .").

For example, let us suppose not only that $A \leftarrow B$, but also that $D \leftarrow E$. Let us suppose also that the condition C is contained in section *n*. Finally, let us suppose that both of the rules are qualified by the phrase "subject to the exception mentioned in section *n*" (i.e. C). Kowalski's method would mean that C would be represented as two rules, $-A \leftarrow C$ and $-D \leftarrow C$, must appear in the formalization, whereas it would be useful to be able to have, as in the statute, a single representation of C which the meta-interpreter, using SUBJECT-TO assertions, can know as an exception to both rules.

The best of both worlds can be achieved by compromising the representation a little and including in the formalization of a rule a list of conclusions to which it is an

exception, namely:

RULE(KB, Source, Conclusion, Conditions, Goals)

For example,

```
RULE(housing-act-1985,
  [ha, sched-1],
  long_tenancy(Tenancy),
  [ ],
  [secure_tenancy(Tenancy)]).
```

The need for such a technique which has no logical justification suggests that if we are to represent a statute in the manner this paper describes, then a distinction should be drawn between the use of logic as a representational language and its use as a programming language. That is, we give up the idea that our straight formalization in logic can be directly executable. However, the step which we have to take in order to bridge the gap between what we might now call logic specification to logic program is remarkably slight.

Having said that, one should not underestimate the practical difficulties in developing a satisfactory, domain-specific meta-interpreter of which the above was a sketch. In fact, they are formidable. One of the main problems is the variety of meta-level relations one will need to deal with. Even under the heading of "deeming provision" there is a wide variety of constructs. Let us look at one more example. The British Nationality Act 1981 (section 1) shows us that a deeming provision can have an explicit scope parameter:

> A new-born infant who . . . is found abandoned . . shall . . be deemed for the purposes of subsection (1) . . to have been born in the UK.

The striking effect of this is that a thorough definition of STATUTE-PROVE which was able to reason with deeming provisions with scope would demonstrate that statute-proveability must be scope-relative.

6. Indeterminacy and non-monotonicity

In Berman and Hafner (1987) three obstacles to the use of logic based models are advanced. The third is the objection we have looked at above (see section 3), in this section we shall examine the remaining two. They are firstly that logic permits only the truth values true and false whereas law is often indeterminate, and secondly that legal reasoning is non-monotonic, whereas logic is monotonic. We will discuss each of these in turn.

Indeterminacy is defined in Berman and Hafner (1987) p. 3 as follows:

> By indeterminacy legal scholars mean the ability to justify both sides of a legal question using accepted legal principles to reach mutually inconsistent results. This is true even in cases where there is agreement on the facts and the applicable rules of law. Thus, the law as a decision-making system is "indeterminate"—in the majority of cases the decision could go either way.

This is in clear contrast to classical logic, where a sentence is either true or false. But it is rather disingenuous to cite the fact that either side of a case could be argued

to suggest that a legal conclusion lacks a truth value, for the legal process is designed to provide a determinate answer to disputed legal questions. In an adversarial system, the system requires that both sides be argued, but that a determinate decision must be made when the two sides have been presented. (The situation is, in fact, better in law than in mathematics, which is uncontroversially accepted as an appropriate field in which to apply logic: whereas any legal question can be determinately resolved, some mathematical statements are provably unprovable.) What we need to show is not that it is possible for a logic-based approach to give an indeterminate decision, but rather to explain how disagreement as to the outcome of a case is possible. If we have a formalization of law, then any given proposition will or will not be a consequence of that formalization. There are, however, three ways in which indeterminacy can arise. Firstly there may be disagreement as to the correct formal interpretation of the law. Suppose we have a law that states that a person is entitled to a heating addition to some benefit if his house is hard to heat.

Suppose too that both parties are agreed that a particular claimant has a house which is not hard to heat. We will be entitled to conclude that he is therefore not entitled to a heating addition, only if we interpret the "if" in the legislation as "if and only if". Thus one way of arguing both sides of a legal question would be to propose two different formalizations as competing candidates for the correct interpretation of the applicable fragment of legislation. Secondly there is the irreducibly open-textured nature of many legal concepts; in the above example the two sides might be in perfect accord as to the facts of the case, that is the nature of the claimant's house, and yet disagree as to whether or not the predicate "hard to heat" correctly described the claimant's house. This is not a matter of fact, but of judgement. Thus both sides could be in agreement as to the correctness of the formalization, and so agree that if the claimant's house is not hard to heat then he is not entitled to heating addition, but argue the contrary sides on the basis of the applicability of the open-textured predicate to the facts. Thirdly, in practice, we often see legal decisions that fly in the face of the letter of law, so as to ensure that the outcome accords with the judge's idea of what is just. Such a decision may well be justified by reference to some extra-statutory legal principle. Berman and Hafner (1987) cite the case of the heir who murders his grandfather, where an Ohio court held that the heir could inherit and a New York court that he could not. Here the New York court appealed to the general principle that no wrongdoer should benefit from his wrong. We could consider this to be a case in which one formalization (that notionally used by Ohio) did not contain a representation of this principle and some other formalization (notionally used by New York) did. This would reduce it to the first case above, a disagreement as the formalization to use. Alternatively we could leave it to the user to review the consequences of our formalization in the light of general legal principles he found acceptable; this would always give the possibility of the "decision" of the system being rejected by the user. This latter solution is probably to be preferred, given the room for disagreement as to what constitutes a general legal principle, and the circumstances in which it is applicable.

Thus we have the situation where two sides of a legal decision can be argued, even though a legal model of the legislation gives a determinate answer on any of the following three grounds: that the formalization fails to embody a correct

interpretation of the law; that the conclusion depends on the applicability of some open-textured predicate which is not applicable; or that the formalization fails to embrace some general legal principle which should be operative. We submit that these three factors give rise to sufficient scope for both sides of any question to be argued, and that such legal disagreement, can, if there is agreement as to the facts and applicable rules of law, be accounted for in terms of one or more of them.

The second obstacle is that of non-monotonicity. Here we must admit that in the course of reviewing the facts of, and legislation applicable to, a given case, a legal reasoner may first be persuaded one way and then another. However, the situation need not be described as deciding first one way then another, as would be the case if true non-monotonicity were in evidence. Rather it is better to say that no decision is made until all the facts and laws have been considered. The interim views are better expressed as "on the basis of the facts and laws so far considered, the decision is this", than as "the decision is this and I have to consider more factors". When a case is decided, complete information is supposed to be available: if it is not then decision should be deferred until it is, since it would be clearly unjust to decide a case on the basis of only partial information. Where complete information cannot be made available, perhaps because there were no witnesses, as is the case with certain facts relevant to certain cases, a decision must be made on the balance of probabilities, or with reference to whatever other procedure the law provides. Thus information can be taken as being complete even in such cases. Given this (real or assumed) completeness of information the question of non-monotonicity does not arise. Against this it might be argued that jumping to conclusions is a fundamental feature of human legal reasoning; Gordon (1988) seems to be saying something of the sort when he says that a conclusion drawn on the basis of a general rule is overturned when an exception is discovered. To this there are two replies; first that the formalization approach does not attempt to model the reasoning processes of lawyers, but only to support legal decision making, and second that if non-monotonic problem solving behaviour is required, this can be accommodated within a strictly monotonic framework of inference by placing the inference component under the control of a non-monotonic problem solver and an Assumption-Based Truth Maintenance System, of the sort described in de Kleer (1986).

Apparent non-monotonicity can, in addition to coming to conclusions on incomplete evidence, arise when a decision in a particular case is reversed on appeal, and when a decision is quashed in the light of new evidence which shows that a previous decision was not soundly based. Neither of these cases presents a problem for logical formalizations. The first is simply a special case of both sides of the case being argued; the appeal will be allowed as a result of a disagreement falling under one of the three heads outlined when discussing indeterminacy. Also note that any system must provide a final level of appeal (corresponding to the House of Lords in the UK): and any decision of a lower level adjudicating body should be regarded not as absolute, but as conditional upon a superior court not overturning the verdict. The case of new evidence simply means that the situation has changed; a fact which was taken as true when deriving the original decision, should not have been so taken, and nothing in logic states that a different set of facts may not give rise to a different conclusion.

7. Conclusion

Work on the British Nationality Act, and the response to it, established a debate concerning the applicability of logic for the representation of legislation, and of law in general. The attraction of the BNA paradigm is dependent on the idea that statutes are representable as fairly simple logic programs. Difficulties in representing some things in Horn clauses were discovered and further research within this paradigm was largely an effort to discover general ways of overcoming those problems. Critics brought forward arguments to show that statutes could not be represented as logic programs and (it is important to keep these questions distinct) that logic was an inadequate tool for representing legislation.

This paper provides a synthesis of elements of the two positions. It proposes that knowledge representation problems will be inevitable unless formalizations respect the multi-layered logical structures which most statutes exhibit. This requires the introduction of meta-level features into a formalization which in turn destroys the possibility that the formalization can be interpreted as a 'logical model' of the intuitive kind which the BNA work suggests. On the other hand, once a formal representation of a statute incorporates meta-level features to enable a closer approximation of its complex structure, it will meet some of the points made by critics of the logic-based approach.

From a logical point of view the most significant and striking finding from all of the above is that, in capturing the content of self-referential bodies of text, one cannot always safely treat as irrelevant the admittedly contigent organization of that text.

References

ALLEN, L. E. (1957). Symbolic logic: a razor-edged tool for drafting and interpreting legal documents. *Yale Law Journal,* **66,** 833–879.

BENCH-CAPON, T. J. M., ROBINSON, G. O., ROUTEN, T. W. & SERGOT, M. J. (1987). Logic programming for large-scale applications in law. In *Proceedings of the First International Conference on Artificial Intelligence and Law,* pp. 190–198. New York: ACM Press.

BENCH-CAPON, T. J. M. (1988*a*). Logical models of legislation and expert systems. In H. FIEDLER, F. HAFT & T. TRAUNMULLER, Eds. *Expert Systems in Law: Impacts on Legal Theory and Computer Law,* pp. 27–41. Tubingen: Attempto Verlag.

BENCH-CAPON, T. J. M. (1988*b*). Applying legal expert systems techniques: practical considerations. In P. DUFFIN, Ed. *KBS in Government 88.* Pinner, UK: On Line Publications.

BENCH-CAPON, T. J. M. (1989). Representing counterfactual conditionals. In A. COHN, Ed. *Proceedings AISB '89.* London, UK: Pitman.

BERMAN, D. & HAFNER, C. (1987). Indeterminacy: A challenge to logic-based models of legal reasoning. In C. ARNOLD, Ed. *Yearbook of Law, Computers & Technology,* vol. 3, pp. 1–35. London: Butterworths.

BIAGOLI, C., MARIANI, P. & TISCORNIA, D. (1987). ESPLEX: a rule and conceptual model for representing statutes. In *Proceedings of the First International Conference on Artificial Intelligence and Law,* pp. 240–251. New York: ACM Press.

BOWEN, K. A. & KOWALSKI, R. (1982). Amalgamating language and meta-language in logic programming. In CLARK, K. L. & TARNLUND, S. A., Eds. *Logic Programming,* pp. 153–172. London, UK: Academic Press.

de KLEER, J. (1986). An assumption based truth maintenance system. *Artificial Intelligence,* **28,** 127–162.

GINSBERG, M. (1986). Counterfactuals, *Artificial Intelligence*, **30**, 35–79.

GORDON, T. F. (1987). Some problems with Prolog as a knowledge representation language for legal expert systems. In C. ARNOLD, Ed. *Yearbook of Law, Computers & Technology*, vol. 3 pp. 52–67. London, UK: Butterworths.

GORDON, T. F. (1988). The importance of non-monotonicity for legal reasoning. In H. FIEDLER, F. HAFT & T. TRAUNMULLER, Eds. *Expert Systems in Law: Impacts on Legal Theory and Computer Law*, pp. 111–126. Tubingen: Attempto Verlag.

GREENLEAF, G., MOWBRAY, A. & TYREE, A. L. (1987). Expert systems in law: The DataLex project. In *Proceedings of the First International Conference on Artificial Intelligence and Law*, pp. 9–17. New York: ACM Press.

HAMMOND, P. & SERGOT, M. J. (1984). *APES Reference Manual.* Richmond, UK: Logic Based Systems Ltd.

KOWALSKI, R. & SERGOT, M. (1986). *The Use of Logical Models in Legal Problem Solving*, Logic Programming Group Research Report, Dept. of Computing, Imperial College, London, UK.

KOWALKI, R. (1989). The treatment of negation in logic programs based on legislation. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pp. 11–15. New York: ACM Press.

LEWIS, D. (1973). *Counterfactuals.* Oxford, UK: Blackwell.

MCCARTY, L. T. (1977). Reflections on Taxman: an experiment in artificial intelligence and legal reasoning. *Harvard Law Review*, **90**, 837–893.

PERLIS, D. (1985). Languages with self-reference I: foundations. *Artificial Intelligence*, **25**, 301–322.

ROUTEN, T. W. (1989). Hierarchically organized formalizations. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pp. 242–250. New York: ACM Press.

ROUTEN, T. W. & BENCH-CAPON, T. J. M. (1986). *Counterfactuals in Logic Programs*, Logic Programming Group Research Report, Dept. of Computing, Imperial College, London, UK.

SERGOT, M. J., CORY, H., KOWALSKI, R. A., KRIWACEK, F., HAMMOND, P. & SADRI, F. (1986). Formalization of the British Nationality Act. In C. ARNOLD, Ed. *Yearbook of Law, Computers & Technology*, vol. 2, pp. 40–52. London, UK: Butterworths.

SHERMAN, D. M. (1987). A PROLOG Model of the Income Tax Act of Canada. In *Proceedings of the First International Conference on Artificial Intelligence and Law*, pp. 127–136. New York: ACM Press.

STUDNICKI, F. (1985). Computational aspects of legal interpretation. In C. WALTER, Ed. *Computing Power and Legal Reasoning.* St. Paul: West.

WELHAM, B. (1988). Interpreters and meta-level inference. In P. MAES & D. NARDI, Eds. *Meta-Level Architectures and Reflection*, pp. 287–299. Amsterdam: North-Holland.