The Argument Construction Set — A Constructive Approach to Legal Expert Systems

Thomas F. Gordon
German Research Center for Computer Science
Sankt Augustin, Federal Republic of Germany

Abstract

Usually, legal expert systems are conceived as rule-based systems for subsuming the facts of a case under the general rules of some substantive area of law. In such systems, the role of the lawyer-user is reduced to that of answering questions about the facts of the case. The Imperial College expert system for the British Nationality Act is perhaps the most familiar system of this kind. The guiding view of jurisprudence in such systems is that the substantive law of some field can be adequately represented as a set of rules and that deduction is the central task of legal decision making. There is an opposing view which asserts that deduction, although important, plays only a secondary role and that the principle task of a lawyer when analyzing a case is the construction of the theory of the law and facts from which the legal conclusions deductively follow. In this paper, we describe the architecture of a software system, called the Argument Construction Set, which is a first attempt to support this second, theory construction view of legal decision making. The system does not depend on the existence of clear legal rules and applies Artificial Intelligence ideas concerning reason maintenance and nonmonontonic reasoning.

1 Introduction

A popular view of the law is that it is a system of rules and that the decision of legal cases is to a large extent a question of applying the rules of law to the facts of the case, in a deductive fashion. This view stresses the role of deduction in legal decision making, and ignores a variety of issues which have great practical and philosophical importance, such as determining whether or not the general terms of the law are satisfied by the events of the case, or deciding how to interpret the variety of sources of law, such as statutes and cases, in order to arrive at a representation which is amenable to some form of deduction. It also neglects the fact that logics are human inventions; there is not one logic, but arbitrarily many, and it is not at all obvious which, if any, of the currently known systems of logic are suitable for this task.

The old issue about just what role logic plays, or should play, in legal decision making is of current interest because of the new technologies arising out of the field of Artificial Intelligence (AI), especially expert systems. So far as I am aware, there is still no commercially successful expert system (deserving the name) in the field of law, so this discussion is still very much academic. There have been a number of attempts to build expert systems in the law, such as [McCarty 77, Sergot 86, Susskind 87], and not surprisingly, the whole enterprise has also begun to attract its critics [Leith 85]. Most programming environments for constructing expert systems allow knowledge about a field to be represented as a set of rules, where the exact nature of these rules varies a great deal. For those who consider the law to be a

system of rules, the idea of building expert systems in law using such environments is an obvious one.

The controversy regarding the suitability of rule-based systems for building legal expert systems centers around the feasability of representing the law a set of rules, as a naive view of the law would suggest. A representative of the camp arguing that the rule-based approach is appropriate, at least for some useful subset of the tasks facing lawyers, is Richard Susskind, whose dissertation on expert systems in law addresses this issue at length. One of Susskind's central arguments, following Hart, is that although the law is open-textured, there is a class of clear cases which can be described adequately by rules [Susskind 87]. The principal opponent of the rule-based approach for the legal expert systems is Philip Leith, a rule skeptic who disagrees with Hart, and therefore Susskind, by arguing, essentially, that there is no method for identifying the clear cases [Leith 85].

The debate about the whether or not there are clear rules will not be of further interest to us in this paper. Rather than taking sides with one camp or the other, we will sidestep the issue by describing an alternative architecture for legal expert systems which does not depend on the existence of clear rules or cases. The basic idea is due to Fiedler [Fiedler 85], who views legal decision-making as being not principally a deductive task of applying rules, clear or otherwise, to the facts of a case, but as a constructive task of *designing* the facts and rules, in an iterative fashion, so as to justify the decision. In this conception, the lawyer or judge creates an argument from the sources of law, such as statutes and reported cases, and the information available regarding the facts of the case. This argument must satisfy certain constraints, only one of which is that the decision must deductively follow from the interpretation of the law adopted.

Fiedler described his architecture in general terms. The main goal of this paper is to refine these ideas by presenting a software system in which some of them are realized, the Argument Construction Set (ACS).

It may be thought that a *proof checker* of some kind for standard logic would be adequate as a basis for such a system. However, any earnest attempt to formulate the relevant law in first-order predicate logic would often, if not usually, make it impossible to derive the intended decision of the case because of the lack of information about the facts of the case sufficient to rule out all known exceptions to the rules being applied. In the last ten years or so, a great deal of effort in AI has gone into solving the problem of reasoning with incomplete information. For technical reasons, the subject has come to be called "nonmonotonic reasoning". The problem has by no means been solved, but the issues have become much clearer and a variety of approaches, some of them promising, have appeared. The Argument Construction Set is based on one of these approaches, called "reason maintenance".

In the rest of this paper, first the deductive view of legal reasoning is described in greater detail; then, Fiedler's constructive conception of legal decision making is constrasted with the deductive view; next, the arguments for nonstandard logics supporting nonmonotonic reasoning in the context of legal decision making are outlined, and the principal AI approaches to nonmonotonic reasoning are introduced; finally, I describe the design of the Argument Construction Set and the state of the current prototype.

2 Deductive View of Legal Reasoning

Before describing the constructive view in detail, for the purpose of comparison let me first outline more precisely the more conventional deductive conception of legal decision making and its realization in legal expert systems. The situation of interest, according to this deductive view, involves a lawyer, who may be a judge,

faced with

1. a body of law,
2. the facts of a particular case, and
3. an issue to be decided.

The legitimate question as to where these elements come from is not addressed, at least the issues raised by this question are not considered to be of central importance to an explication of the nature of legal decision making. In any case, the deductive view concerns itself only with those issues which arise after these elements have been identified and stated.

Given these three components, the task of the lawyer, according to this view, is to decide the issue by formulating a proposition for the issue and then determining in some fashion whether the proposition or its negation deductively follows from the law and facts.

Critics of the deductive view sometimes claim that this task is a simple one and that the truly difficult tasks involved in deciding legal cases are elsewhere. While there are indeed difficult tasks ignored by the deductive view, some of which will be discussed below, the claim that deduction itself is simple is unsupported. One relatively simple logic is propositional logic. It is of limited expressiveness, but offers the advantage of *decidability*, that is, procedures exist for deciding whether some proposition is entailed by some set of other propositions. But decidability alone does not guarantee that the procedures available are efficient enough to be useful in practice. This is unfortunately the case for propositional logic: it has been proven that no efficient procedure for propositional entailment exists. (Technically speaking, propositional entailment is *NP complete*.) The situation just gets worse with more expressive logics, such as first-order predicate logic, which is not even decidable. That is, no procedure guaranteed to terminate, of any complexity, exists for determining whether one formula is logically entailed by a set of other first-order formulas. These considerations are not intended to suggest that these logics are not useful, just that the task of finding logical proofs, even after the the law and facts have been expressed as a set of formulas in the logic, is not a trivial one.

There are other problems, of course. If the law and facts are *incomplete*, in both formal and informal senses of the word, then it may be that neither the proposition nor its negation is entailed by them. For example, if the law states that vehicles are not allowed in the park, and the facts are that a bicycle was in the park, it does not follow *logically* that the bicycle was not allowed in the park, as there is no explicit statement in the law, as it has been stated, to the effect that all bicycles are vehicles. This is Hart's example of *open-textured* concepts and illustrates the problem of *subsumption*, i.e. whether a specific term, in this case "bicycle" is subsumed by a more general one, here "vehicle".

Despite such difficulties, there have been a variety of attempts to build prototype legal expert systems based on this deductive view of legal decision making [McCarty 77, Sergot 86, Susskind 87]. Usually, the designers of these systems are aware of such problems, but claim that the systems are potentially useful tools for lawyers despite such qualifying limitations. No serious researchers today claim that such deductive systems can be used for autonomously deciding legal issues. Rather, it is only claimed that such systems can be useful for a preliminary analysis of cases by sophisticated users who are aware of the dangers and limitations of the method.

At an abstract level of description, these tools for constructing legal expert systems share a common architecture. With such systems, there are two types of users: the expert lawyer who builds a "knowledge base" for some specific and technical area of law, and the non-expert lawyer, who applies the knowledge base

to analyze particular legal cases. The software system includes components for building and testing knowledge bases, to be used by the expert lawyer, and tools for the nonexpert, such as a *dialog manager* which prompts the non-expert for the facts of the case, an *inference engine* which selects and applies rules, and an *explanation module* which explains how conclusions were reached, usually by displaying the rules which were applied in some more or less digestible fashion.

Notice that this architecture does not exactly mirror the description of deductive legal reasoning outlined above. The facts of the case need not have been illicted before the system attempts to find a "proof" of the proposition in question. Questions regarding the facts are posed to the user during the attempt to find a proof. There are also differences between the meaning of "rule" in the expert systems context and "inference rule", as logicians use the term. In logic, inference rules are domain-independent. In expert systems, the domain expert describes his knowledge by writing rules tailored to his subject. From a logical perspective, there is no difference between what are called "rules" and "facts" in the expert systems context: they are both "formulas" representing sentences. This does not mean that expert systems do not use logical inference (although in fact most expert systems use *ad hoc* methods with no clear relationship to existing formal systems of logic); rather, in the expert systems community a terminology has developed which is somewhat at odds with the terminology used by logicians.

3 Constructive View of Legal Reasoning

The expert systems architecture described in the previous section was not developed for the explicit purpose of building expert systems in law. Initially, expert systems were developed for a specific field of application. Later, the basic methodology and software tools were abstracted from these complete applications with the goal of developing domain-independent programming environments, usually called "shells", for building applications in arbitrary domains. Most attempts to build legal expert systems used one of these off-the-shelf programming environments, e.g. [McCarty 77, Sergot 86]. At the same time, however, there have been attempts to develop *knowledge representation languages*, i.e. programming languages for expert systems, tailored specifically to the problem of representing the law [Stamper 80, deBessonet 84, Gordon 87]. Although these languages are intended for representing the law, with each aimed at some particular problem of legal knowledge representation, the overall architecture of deductive expert systems described remained the same.

To my knowledge, Fiedler [Fiedler 85] was the first to propose a completely different architecture for legal expert systems, an architecture designed to approximate more closely the way lawyers actually construct legal arguments. Fiedler describes legal decision making as a "modelling" process:

> In effect, the task of the judge essentially includes the choice, shaping and logical construction of the appropriate legal rules as well as the pertinent statements of facts in mutual interdependence. It is true that the resulting fabric of the judgment and its reasons has the function of deductively connecting the decision to the rules of law and the facts of the case. Nevertheless the process of decision-making is not reduced to the application of deductive logic to given premises, but essentially consists in constructing a logical fabric, which at the same time has the qualities of an adequate model and a stringent deduction. In terms of modern methodology, judicial decision-making will have to be qualified as a process of model-construction or "modelling".

Fiedler then goes on to describe the features legal expert systems should have in order to support this "modelling" process, and draws an analogy between the type of system he proposes and Computer-Aided Design (CAD) systems. His requirements list is quite comprehensive, if abstract, and includes components for assisting the creation of natural language texts documenting the arguments constructed, such as briefs and judgments.

The role of deduction in Fiedler's conception of legal reasoning is principally one of *justifying* decisions, after they have been made, at least tentatively, rather than to provide a method for reaching or discovering decisions. That is, the decision must follow logically from the rules of law and the facts of the case, but the rules and facts are not present from the beginning; they are composed by the lawyer during his construction of an argument justifying the decision. The rules are formed by interpreting, and reinterpreting, the various sources of law, such as reported cases, and the facts arise out of the interpretation and discovery of evidence. The choice of terminology for stating the facts and the rules are mutually dependent on one another. The law cannot be formulated without considering the facts, and the facts cannot be stated without considering the applicable law. Logical deduction is principally a *constraint* on the structure of the resulting argument.

This does not suggest that deduction cannot play an active role in the creation of arguments. Once the law has been tentatively formulated as a set of rules, these rules can be used by a deductive reasoner to help illicit the facts necessary to prove some goal proposition. But if the proposition cannot be proved, it may be that it is the rules which need to be reformulated, by reinterpreting the legal sources. When rules are reformulated, the expert system should be able to assist the lawyer in withdrawing just those conclusions which depend on the previous formulation.

That legal arguments must be logical is not controversial; but this basic constraint of rationality is of course not sufficient justification for a legal decision. Justice has other demands as well, such as the principle that like cases should be treated alike, which guides the treatment of judicial precedents. There might be software tools that one can imagine which would assist lawyers in making decisions which are justifable not only on logical grounds, but on broader grounds as well. Fiedler's proposed architecture for legal expert systems does not eliminate the possibility of such tools, but does not explictly require them either.

In the usual conception of legal expert systems, there are two classes of users: the expert who builds the knowledge base and the lay user who applies the knowledge base to some particular problem. What is the role of the expert lawyer in Fiedler's constructive view of legal decision making? The question concerns the role of secondary sources of law, such as case books, treatises and commentaries, in legal decision making. These secondary sources are compilations of the knowledge of experts in legal specialities. In contrast to the knowledge bases constructed by experts for use in conventional expert systems, however, theses secondary sources of law are not self applying; the lawyer analysing some case must interpret these secondary sources just as he interprets the primary sources, the statutes and published case reports. Secondary sources provide an additional source of material to aid the process of interpretation, but do not relieve the decision-making lawyer from his responsibility of interpretation.

In legal expert systems based on this constructive model of legal decision making, knowledge bases could be prepared in advance by legal experts, just as is done in conventional expert systems based on the deductive model, but one must be careful to design the system so as to prevent the experts from usurping the responsibility of the lawyer deciding the case for the ultimate content of the model of the law used. The non-expert lawyer using the constructive system to decide some case must be held responsible for the content of the knowledge base, for the formulation of the rules used to reach the decision. The non-expert should not be able to delegate

this responsibility to an expert. To the extent that the expert system encourages or permits the non-expert lawyer to delegate responsibility for interpreting and understanding the original legal sources to an expert, the difference between the constructive and deductive views is diminished.

There are two interests to be balanced here:

1. The desire to compile knowledge bases in advance which may be applied repeatedly to many different cases. Experts can spend their careers developing and refining such knowledge bases, saving end users time and effort and providing them with the benefit of the special insight and knowledge which can only be obtained after intensive study over a period of years; and

2. The importance of reinterpreting the law in light of the events of new cases, events which may not have been forseen by either the judicial or legislative bodies, or by the experts who interpret primary sources of law.

In view of the first of these goals, it would be unreasonable to simply forbid precompiled knowledge bases from being used in expert systems. There is a middle ground, however, between an "expert system" which does not support the use of knowledge bases constructed by experts, and expert systems in which such knowledge bases are self applying: the system could be designed so as to allow the lawyer deciding the case to easily modify knowledge bases prepared by experts. In addition, the explanation module should be constructed so as to not merely explain which rules were applied, but to explain as well why the rule was formulated the way it was. That is, information about the primary sources of law relied on in formulating the rules should be included, together with whatever other information is necessary to understand just why the expert chose this representation of the law. As an additional safety guard, the system could require the end user to review and explicitly confirm that he is in agreement with the formulation of a rule, before it is applied.

Most expert systems perform *diagnostic* tasks; given information about some object, it is their job to classify the object given some hierarchy of terms or to identify certain of the objects properties. For example, expert systems for medical diagnosis are provided with information about the symptoms of a patient and then attempt to identify the particular illness or illnesses causing the symptoms. Legal expert systems based on the deductive model of the law can be considered to be of this type: given information about the facts of the case, they apply a representation of the law in order to classify the operative facts of the case according to the general terms of the law. Not all expert systems are diagnostic systems, however. Another class of expert systems perform *configuration* tasks. For example, one of the most commercially successful expert systems, XCON, routinely configures Digital Equipment Corporation VAX minicomputers so as to meet the requirements of DEC's customers. Legal expert systems of the type proposed here, based on the constructive view of legal decision-making, belong to this configuration class; they would help "configure" arguments so as to meet, at least, the formal requirements of justice.

As mentioned above, Fiedler drew an analogy between the type of legal expert system he is proposing and Computer-Aided Design (CAD). This analogy is helpful only to a certain extent. CAD systems are not usually knowledge-based systems. They do not usually apply Artificial Intelligence methods of knowledge representation and inference. If the system proposed were to be limited to a kind of proof checking function, if it would not support the use of knowledge bases prepared by experts in advance, then the CAD analogy would be more accurate. But such a system could not properly be called an *expert* system, as it would not contain expert knowledge, except in the limited sense that it would implicitly contain knowlege

about logic and deduction. A CAD system, however, can be a knowledge-based or expert system, and there does seem to be a tendency for CAD to be developing in this direction, incorporating more and more AI technology. The two technologies, CAD and expert systems, are not incompatible.

4 Defeasible Reasoning

Fiedler's theory of legal reasoning as a process of argument construction, as I have described it above, says nothing about the form such arguments have, or should have. In his proposed architecture for legal expert systems, however, he does state that:

> Knowledge representation and inference mechanisms should provide for specific customs of legal interpretation and specific strategies of legal reasoning. (e.g., modularization of the rule base functionally similar to law, which would also enhance the understandability and modifiability of the system ... inference mechanisms incorporating methods of legal heuristics.)

This goal speaks against the use of standard propositional or predicate logic for expressing legal arguments. Although the bulk of the jurisprudence literature about the role of logic in the law seems to assume that, when logic plays a role at all in legal reasoning, this role is adequately filled by a standard logic, AI has some new ideas to contribute to this theme, arising out of attempts to understand common sense reasoning.

Standard logic is *monotonic*, in the following sense: if a proposition is entailed by a set of propositions, then it is also entailed by every superset of the initial set. That is, no additional information will allow previous conclusions to be retracted. There is a flip side to this: only those propositions which are necessarily true given that the premises are true can be deduced. That is, standard logic offers no mechanism for prefering one proposition to another if neither proposition is necessarily true given the premises.

Contrast the monotonicity of standard logic with common sense reasoning. In the context of common sense reasoning, truth is not the principal goal, but rational, considered decisions. Decision-making occurs in time and space and is subject to limited resources, in particular limited information. When faced with a decision to make, if a relevant proposition cannot be deduced from available information, then other forms of reasoning must be applied. If the proposition is usually true in contexts similar to the current one, then an agent may assume it to be true in this case as well. That is, some form of probabilistic reasoning may take place. Even if the proposition is probably true, however, an agent may choose to believe that it is not true, as he must consider the consequences of making a wrong decision. The relative risks of error must be weighed.

Legal reasoning is nonmonotonic, and the traditional forms of expressing legal rules is designed to support this kind of reasoning. Laws are not usually stated as logical propositions, although they bear a surface resemblance to propositions because of their use of such key words as 'if', 'and', 'or', etc. Rather, they are expressed as general principles, which are then qualified and refined in other statements. If these principles can be viewed as rules, such rules are subject to implicit exceptions which are made explicit, if at all, in other sections of the legal text.

For example, according to BGB §108 of the German civil code, contracts are valid. But then §108 goes on to say that contracts with minors are not valid. Later we read that contracts with minors which have been ratified by a guardian of the minor are valid. Hart's famous example for the open texture of legal concepts also

demonstrates this aspect of legal reasoning. The general rule is that vehicles are not allowed in the park. When a court decides that baby carriages, however, are not excluded from the park by this law, it need not argue that baby carriages are not vehicles. In this case the rule is subject to an implicit exception; indeed an exception that is nowhere made explicit in the text of the statute.

In [Gordon 88] I discuss the importance of nonmonotonicity for legal reasoning. Its importance stems from the normative and conflict resolution goals of law. Briefly,

1. The normative goal of law requires that average persons, in particular non-lawyers, be able to learn and apply the law. General rules, subject to exceptions, are shorter, simpler and easier to learn and apply than complex, detailed statements of the law. Exceptions can be learned, incrementally, as the need arises.

2. Procedures for resolving civil disputes according to law must be sensitive to the economic value of resolution to the parties involved. There is no simple correlation between the complexity of the legal issues raised by a case and the value of resolution. Burden of proof rules provide a means of reducing the cost of judicial decision-making by permitting facts to be assumed without proof. It is left to the parties to decide which issues to raise, based on their own evaluation of their interests and taking into consideration the potential costs of proving some point. Burden of proof rules are nonmontonic inference procedures, in the sense that they permit propositions to be accepted without proof. The use of general rules with exceptions, is one way of allocating burden of proof.

I have been arguing that nonmonotonicity is required for legal expert systems; but what relationship does nonmonotonic reasoning have to logic, and how can one go about adding nonmonotonicity to expert systems? It could be argued that nonmonotonic reasoning has little to do with logic, by restricting the conception of consequence to the standard notion in which a proposition is entailed by a set of axioms only if it is necessarily true when the axioms are true. From this point of view, logical inference is just one reasoning process used by decison-making agents, among others. But a principled conception of rationality and, in the legal context, of justification, requires that these "extra-logical" processes be specified clearly enough so as to be able to determine, in some way, whether a decision is rational (or a judgment justified) when nonmonotonic inferences have been made. If this can be done, then the usual notion of consequence can be modified so as to include these additional factors and considerations. There are many nonstandard logics, each with its own nonstandard notion of consequence or entailment, so there is nothing especially unusal about this idea. Not all AI approaches to nonmonotonic reasoning take this semantic approach, however. In fact, most of the approaches have been ad hoc procedural approaches in which it is unclear what relation they have to standard logical notions such as entailment.

There have been a great many approaches to nonmonotonic reasoning proposed in AI. Nonmonotonic reasoning is a central requirement for many AI tasks. Unfortunately, this is one area where theory and practice are still quite far apart. Great progress has recently been made on both fronts, but the practical approaches need to be cleaned up by the theorists, and the promising theories still need efficient proof procedures and implementations. For our purposes here, a brief description of the most significant work in the field will have to suffice. See [Genesereth 87] or [Brewka 87] for introductions to the field, and [Ginsberg 87] for a collection of historically important papers on the subject.

4.1 Formal Approaches to Nonmonotonic Reasoning

On the theoretical side, the most discussed approaches presently include *circumscription*, *default logic* and *autoepistemic logic*. None of these approaches is easy to explain or to understand unless one has a solid foundation in mathematical logic; but let me try at to convey a feel for each of these approaches.

4.1.1 Circumscription

Circumscription was first proposed by John McCarthy in 1980 [McCarthy 80]. In the meantime, a variety of forms of circumscription have been developed, so it no longer makes sense to talk about *the* circumscription method for nonmonotonic reasoning. The central idea of all versions is the same: given a set of first-order predicate logic formulas representing what is known about the domain of interest, add a second-order formula to *minimize* the extensions of certain selected predicates. Unfortunately, to explain this idea adequately would require an excursion into model theory, which would be too envolved for our purposes here. Perhaps an example will help. According to German law, contracts are presumed to be valid. Contracts with minors, however, are presumed not to be valid. These two rules could be represented in first-order logic as:

$$\forall x\,.\,contract(x) \wedge \neg exceptional(x) \rightarrow valid(x)$$

$$\forall x\,.\,contract\ with\ minor(x) \rightarrow exceptional(x)$$

Given only these two formulas, it is not possible to show that a contract is valid if it cannot be explicitly shown that the contract is not exceptional, which is not what is wanted. How can we arrange things so that a contract can be presumed to be valid if there is no information as to whether or not it is exceptional? Circumscription achieves this with a second-order axiom stating, in effect, that the only things which are exceptional are those which *must* be given the axioms.

A proof procedure for full circumscription is not possible because of the second-order axiom; but special cases of circumscription have been discovered in which the second-order axiom can be reduced to a first-order one, permitting existing theorem proving techniques to be applied. Another serious problem with circumscription is the difficulty in selecting the second-order axiom. The axiom to chose depends on the intentions of the user, and methods need to be discovered for assisting users in designing the axiom, depending on their goals.

Circumscription is the most general of the minimization approaches to nonmonotonic reasoning. The *closed-world assumption* is another, more specialized, approach to nonmonotonic reasoning based on the minimization idea. In contrast to circumscription, however, the closed-world assumption is widely applied, in data bases and in the logic programming language Prolog, for example. Unfortunately, the closed-world assumption does not offer enough control over the allocation of burden of proof to be useful for legal expert systems.

4.1.2 Default Logic

Default logic was created by Raymond Reiter, also in 1980 [Reiter 80]. His approach is proof theoretical; general rules subject to exceptions are not represented as formulas within the logic, but as *inference rules* tailored to the particular theory of interest. These inference rules for default reasoning have the form:

$$\frac{\alpha : \beta}{\gamma}$$

which is intended to mean

If α is true and β is *consistent* with everything else that is true, then γ is true.

Using this approach, the contracts example could be represented

$$\frac{contract(x) : \neg contract\ with\ minor(x)}{valid(x)}$$

$$\frac{contract\ with\ minor(x) : \neg ratied(x)}{\neg valid(x)}$$

These are the default inference rules. The known facts of the case would be represented in the usual way as a set of logical formulas. If all we know is that a is a contract, then the first default rule could be used to deduce that a is valid, as $\neg contract\ with\ minor(x)$ is consistent with what is known.

Default Logic has attracted just about as much attention as circumscription; there are now large number of theoretical results concerning it. Reiter's formulation of Default Logic was proof theoretical, but Etherington gave it a model theoretic semantics [Etherington 87]. Konolige, in an award winning paper [Konolige 87] proved the equivalence of Default Logic and Autoepistemic Logic, which will be discussed next.

An issue arises in the context of Default Logic which is of general importance for nonmonotonic reasoning, the *multiple extension* problem. An extension is, roughly speaking, the original set of logical formulas completed with all other formulas which can be derived by the ordinary inference rules of predicate logic together with applications of the theory specific default rules. However, as default rules can interact with one another, it is possible for a default theory to have more than one extension. For example, there could be extensions in which we conclude that some contract is valid, and other extensions in which just the opposite is concluded. In such cases, the usual approach is to say that any of these extensions is an acceptable set of beliefs. An agent is free to use extralogical means to prefer one such extension to another.

Given that default theories may have multiple extensions, would Default Logic be appropriate for legal reasoning? Put another way, does the law, or should the law, uniquely determine a judge's decision? Or does the law merely constrain judicial decisions to some set of justifiable alternatives, among which judges are free to exercise their discretion? If the first view is adhered to, Default Logic would not be appropriate for legal reasoning. An approach to default reasoning for which there is always exactly one permissable extension would be necessary. It is not clear to me at the moment which of these alternatives is preferable.

4.1.3 Autoepistemic Logic

There have been a variety of *modal* logic approaches to nonmonotonic reasoning. One of the first was NML I, by McDermott and Doyle [McDermott 80]. NML I and its successor, NML II [McDermott 82] are both said to have suffered from a variety of sematical difficulties; they did not quite capture the intended intuitions about nonmonotonic reasoing. The first modal approach to gain wide recognition is Moore's Autoepistemic Logic [Moore 85].

In modal logic, there are 'operators' which make statements *about* statements. For example, the modal operator M is usually interpreted to mean *possible*. If p is a sentence meaning I will inherit a million dollors, then Mp is intended to mean that it is possible that I will inherit a million dollars. Modal operators can be applied to sentences containing modal operators, so e.g. sentences of the type MMp, meaning it is possible that it is possible p are allowed. Note that in modal logic, just as

in standard predicate logic but unlike Default Logic, knowledge about a domain is represented entirely with logical formulas.

Autoepistemic logic uses primarily another modal operator, L, which is usually interpreted to mean *necessary*, but is reinterpreted in the context of nonmonotonic reasoning to mean *known*, or *believed*. To represent the contracts example using Autoepistemic Logic, one could write:

$$\forall x \,.\, contract(x) \wedge \neg L \; contract \; with \; minor(x) \rightarrow valid(x).$$

This formula is intended to mean that all contracts which are not known to be contracts with minors are valid.

Autoepistemic logic is so named because it is intended to model the reasoning of an agent about his own beliefs. Moore's standard example of such reasoning involves whether or not he has a brother. He concludes he does not have a brother, because if he did he would know it. Moore contrasts this kind of default reasoning with *typicality* default reasoning. The classic AI example for nonmonotonic reasoning concerns whether some bird flies. Knowing nothing else about the bird, it is assumed to be able to fly because birds typically fly. Moore argues that these two kinds of nonmonotonic reasoning may need to be handled differently. This raises the question whether still other types of nonmonotonic reasoning are required in other contexts. In the law, for example, defendants in criminal cases are presumed innocent until proven quilty, even though defendants are typically guilty. Thus, typicality is not the basis for nonmonotonicity here. Neither does autoepistemic reasoning seem to being playing a role. We do not presume the defendant to be innocent on the grounds that if he were guilty, we would know it. Rather, the consequences of a wrong decision are being balanced here. It is considered worse to punish an innocent person than not to punish a guilty one.

4.2 Pragmatic Approaches to Nonmonotonic Reasoning

With few exceptions, such as the use of the closed-world assumption in Prolog, the approaches to nonmonotonic reasoning which are theoretically well understood have yet to be applied in AI systems. This lack of mature theory, of course, has not hindered AI engineers from building systems which appear to adequately perform tasks requiring nonmonotonic reasoning. Without well developed theories of nonmonotonic reasoning, however, it is not possible to understand fully just what it is these systems are doing. Two classes of *ad hoc* approaches will concern us here: inheritance hierarchies in frame-based knowledge representation systems, and reason maintenance systems.

4.2.1 Inheritance Hierarchies

Many, if not most, expert system shells today support knowledge representation using a combination of rules and so-called *objects*. Objects of the type meant here have had a variety of names, such as 'frames', 'units', and 'schema', and each realization of the basic idea has been somewhat different than the others. In many systems, for example, a distinction is made between generic objects or *classes*, and other objects which are *instances* of these classes. One feature all these systems have in common, however, is support for a kind of nonmonotonic reasoning called *default* reasoning.

The original motivation for using frames to represent knowledge is due to Minsky [Minsky 75]. An early implementation of the idea is FRL, by Roberts and Goldstein [Roberts 77].

Rather than describe concrete systems in detail, let me describe their common approach to default reasoning in an abstract, system independent fashion. Objects

have internal structure; each object has a number of properties. Each of these properties of an object has a value, which is another object. The properties of an object are sometimes called *slots*, and their values *fillers*. Given a set of objects, the task of the reasoner is to determine the values of their properties. Some of the properties are provided by the user; these properties play a role similar to that of axioms in logic. (Note that one simplification made in these systems is that the reasoner need not infer the existence of objects; all relevant objects are assumed to have been asserted by the user.) The values of the remaining properties are determined in one of two ways: by *property inheritance* or by *procedural attachment*. Only inheritance is of interest to us here, as it is the method which supports a kind of nonmonotonic reasoning.

To use property inheritance, the objects of the system must be arranged into a *hierarchy*, that is the objects must be linked together into a kind of net structure. The links are directed; they point from one object to another, but not in both directions. Cyclic links are prohibited; there is never a path along the links from object to object which leads back to the first object in the path.

Now, given such a network of objects, inheritance can be performed as follows: the value of some property of an object is determined by following links until an object is found for which the property has a value. This value is then *inherited* by the object for which the value was previously undetermined. This procedure is nondeterministic, i.e. it doesn't necessarily result in a unique value for the property, as there may be a number of different paths through the net from the object to other objects with values for the property of interest. Each system has its own strategy for choosing one of the competing values in such cases.

This discussion has been very abstract so far. Let us represent the contracts example using an inheritance hierarchy. We will distinguish between objects which represent classes and objects which are instances of classes. There are two classes to represent: contracts and contracts with minors. The only property we are concerned with is validity. Let us suppose there are objects representing truth and falsity. By installing a link from the contracts with minors object to the contracts object, we can represent that the first is a subclass of the other. Now, the presumption that contracts are valid can be represented by setting the value of the validity property of the contracts object to true. If nothing else is done, the contracts with minors object will inherit this value because of the link from this object to the contracts object. To prevent this, which we want to do because contracts with minors are presumed not to be valid, the value of the validity property of this subclass of contract must be explicity set to false.

To use this system to determine whether some particular contract is valid, an object representing the contract must be created and installed into the net by linking it to other objects. If it is known that the object is a contract with a minor, then it would be linked to the contracts with minors object. The contract would then inherit the value false for its validity property, as desired. If it is not known whether one of the parties of the contract is a minor, then the object would be linked directly to the contracts object, in which case it would inherit the value true for its validity property. Here we see another limitation of this approach to knowledge representation; there is no standard method for modifying links. If it later becomes known that the contract is a contract with a minor, its direct link with the contracts object would have to be moved.

Default reasoning with inheritance hierarchies can be implemented very efficiently. For this reason principally, perhaps, there have been a number of attempts to give inheritance hierarchies a formal semantics [Etherington 83, Touretzky 86].

This brief introduction to structured objects and property inheritance will have to suffice. For more detail about this approach to knowledge representation and default reasoning see, e.g., Nilsson's introduction to AI [Nilsson 80]. Winston and

Horn's AI programming book [Winston 84] includes a chapter on implementing frames in Lisp.

4.2.2 Reason Maintenance Systems

The subject of nonmonotonic logic is concerned with discovering logics which capture our intuitions about the nature of defeasible reasoning and which have desirable computational properties. Despite this interest in computation, however, researchers who have taken a logical approach to the problem of nonmonotonic reasoning have generally not shown interest in the *process* of reasoning with incomplete and changing information over time. Rather they have generally focused their attention on the traditional model-theoretic and proof-theoretic issues of logic. This traditional approach presumes that one is principally interested in determing whether some formula is entailed by some set of premises. This is certainly one problem an agent faces when making decisions, but there are others which need to be discussed and made explicit. People reason about their world in time. They draw inferences based on whatever partial information they have at the time, which later may need to be withdrawn when additional information becomes available. Previous conclusions are applied to future problems, without having to redo the reasoning steps which led to them. Ideally, somehow the *reasons* accepted for believing something are documented or stored, so that a belief, when called into question, can be defended by checking whether those reasons are still valid, without having to rediscover arguments supporting the belief from first principles.

Interestingly, there have been significant efforts within AI to deal with just this sort of problem. The topic has come to be known as *Reason Maintenance* and work in the field has been less theoretical than pragmatic. The principal papers on the subject describe particular methods for constructing software systems which perform reason maintenance. Reinfrank has written a clear and informal introduction to field [Reinfrank 86]. Again, there is only room here for a quick overview of the principal approaches, of which there are two: the TMS approach first developed by Doyle [Doyle 79], and the ATMS approach developed by DeKleer [DeKleer 86a].

The basic service provided by both of these approaches is the mangagement of *dependencies* among *premises*, which are formulas assumed to be true, and other formulas which have been derived from these premises. For the purpose of reason maintenance, the logical language in which these formulas are expressed is, surprisingly perhaps, unimportant. The content and structure of the individual formulas play no role in reason maintenance. Each formula is viewed as an unstructured unit, comparable to the sentence letters of propositional logic. The formulas which have been entered into the reason maintenace system (RMS) are called simply *nodes*; they in effect loose their role as formulas while reason maintenance is being performed. For this reason, the service offered by a RMS is actually much more general than suggested by its application to the problem of managing premises and theorems for some logic. A RMS can mangage dependencies among arbitrary pieces of data.

The structure used to record dependencies is called a *justification*. This is, of course, a very technical use of the term "justification" and should not be confused with its legal meaning, although as we will see there is a connection to be made between these two uses of the word. There are two types of justifications, monotonic and nonmonotonic. Following Reinfrank, [Reinfrank 86], let us use a simple notation for writing down justifications:

$$(I; O \Rightarrow n).$$

Here, n is the node being justified, and I and O are *sets* of nodes, called, for historical reasons, the *in list* and *out list*, respectively. If there are no nodes in

the out list, then the justification is called a *monotonic justification*, otherwise it is called a *nonmonotonic justification*.

To understand what these justifications have to do with nonmonotonic reasoning, we need a further concept, that of a *labelling*. At any given point in time, the state of a reason maintenance system consists of a set of nodes and a set of justifications for these nodes. Intuitively speaking, the task of the RMS is to determine which of the propositions represented by the nodes to believe given the current set of justifications. Technically speaking, the task of the RMS is to attach a *label* to each node indicating whether it is currently *in* (believed) or *out* (not believed). Again, the RMS does this without examining the internal structure of the nodes. (This characterization of reason maintenance is not quite correct in the case of DeKleer's ATMS, but is accurate enough for present purposes.)

Given a set of nodes and a set of justifications, how does a RMS go about determining which nodes to label *in* and which to label *out*. The basic idea is to try to find for each node a justification for which each of the nodes in the *in list* is *in* and each of the nodes in the *out list* is *out*. Such a justification is called a *valid* justification. Thus, intuitively, a justification can be seen as a kind of rule stating sufficient reasons for believing the proposition denoted by its conclusion.

The problem with this simple characterization of reason maintenance, of course, is its circularity. To determine whether some node is *in*, we must determine whether other nodes are *in* or *out*. How do we break this circle? Part of the answer is that some of the nodes are marked as being *premises*, i.e, they are unconditionally *in*. Valid justifications need not be sought for premises. (Alternatively, premises can be marked by asserting justifications with empty *in* and *out* lists.)

Premises are only part of the solution to circular arguments, however. What should be done in the following situation, for example. Suppose there are two justifications, $(\{b\};\{\} \Rightarrow a)$ and $(\{a\};\{\} \Rightarrow b)$. These justifications say, in effect, that a should be *in* when b is *in*, and b should be *in* when a is *in*. The RMS will have to be carefully implemented so as to avoid endless looping in such cases. Clearly, if there are no further justifications which can be used to support a or b then they should be both made *out*.

Although the "correct" labelling in the previous example was fairly clear, how should a and b be labelled if there are only these two justifications: $(\{\};\{b\} \Rightarrow a)$ and $(\{\};\{a\} \Rightarrow b)$? These say that a should be *in* if b is *out* and vice versa, b should be *in* if a is *out*. The correct solution here depends, of course, on our intentions and purposes. Reason maintenance systems *do* have a semantics, in the sense that there are implementation and algorithm independent specifications of their intended behavior. But it is unclear which specification or semantics is appropriate here. The usual approach defines the concept of an *admissible labelling* for a set of nodes. When no nonmonotonic justifications are allowed, then there is always exactly one admissible labelling. When nonmonotonic justifications are supported, and given the usual semantics of reason maintenance systems, due to Doyle [Doyle 83], there may be many admissible labellings, or even none. In the example above, either a or b could be labelled *in*, but not both. Furthermore, at least one of them *must* be labelled *in* given Doyle's semantics; they cannot both be left *out*. Note that this is just the multiple extensions problem discussed above in the section on Default Logic, but in another guise.

Reason maintenance would not not differ significantly from the ordinary conception of theorem proving, if it were not conceived so as to support the *incremental* relabelling of nodes. It helps to imagine the reason maintenance system as being a module or process interacting with the user and other problem solving components in some larger AI system. Communication with the RMS takes place through a small protocol of messages or commands which the RMS has been programmed to evaluate. These commands include asserting new justifications and adding or *delet-*

ing premises. Immediately after modifying the set of justifications and premises in this way, the task of the RMS is to *update* the labellings of the nodes. The RMS does so by reconsidering only those nodes whose label may have been affected by the action, using the dependency information contained in the justifications to make this determination. The labelling of other nodes, of course, remain untouched. The user or problem solving module can then inspect the nodes to discover their current labelling, i.e. to see if the proposition denoted by some node is currently believed. The important point here is that inspecting the label of a node is a cheap operation; it does not trigger any expensive "theorem proving" processes.

Given this functional view of the services offered by a RMS, however, it might be argued that it is unimportant whether or not the RMS actually updates labels in an incremental fashion. The user's only concern is whether or not the RMS is capable of quickly informing him whether or not a node is *in* at inspection time. A theorem prover for a nonmonotonic propositional logic would be adequate for this task, if only the theorem prover were fast enough (and the number of nodes small enough) so as to be able to produce a response within a tolerably short period of time. Until there are efficient theorem provers for some nonmonotonic propositional logic, this perspective is rather academic. Efficient incremental algorithms for nonmonotonic reason maintenance already exist, e.g. [Goodwin 87].

DeKleer's ATMS, meaning *Assumption-Based Truth Maintenance System*, differs somewhat from the description of reason maintenance given above, which more accurately describes systems based on Doyle's approach. DeKleer distinguishes between *assumptions* and *premises*. Both are unconditional nodes in that they do not depend on other nodes; but the propositions denoted by premises are expected to be certainly true whereas the propositions represented by assumptions may or may not, in fact, be true. The practical importance of this distinction should be clear shortly.

Rather than labelling each node *in* or *out*, the ATMS labels each node with the set of *environments* in which the node would be *in*. An environment is a set of assumptions. Now the special role of assumptions, in contrast to premises, can be appreciated. A *context* is a consistent environment completed with all the other nodes which can be derived from the assumptions in the environment using the current justifications. Thus, the principal task of the ATMS is not to determine whether a node is *in* or *out* in some particular context, as in a TMS, but to determine *all* of the contexts in which a node would be *in*. An ATMS can handle multiple contexts simultaneously. To consider multiple contexts using a TMS, the user must assert and retract premises for each context of interest, and the TMS must recalculate the labels every time. To determine whether a node is *in* in some particular context using an ATMS, it is only necessary to check whether one of the environments of the label of the node is a subset of the context.

An ATMS could be of interest for constructing legal arguments. The lawyer would not need to make an early commitment to a particular view of the facts. The ATMS would inform the lawyer which of the various alternative views of the facts are capable of supporting some legal conclusion. Rather than having to prove some particular view of the facts, it would be enough to show that at least one one of the alternative views of the facts needed to prove some legal conclusion must have been the case. That is, using the terminology of ATMS, it would be enough to show that at least one of the environments of the label of the node for the proposition of interest must be satisfied by the actual context, without having to show exactly which one of the environments is satisfied.

The disadvantage of DeKleer's basic ATMS is that it does not support nonmonotonic justifications. Belief in a proposition cannot be conditioned upon disbelief in some other propositon. As the principal reason for discussing reason maintenace here is to review various approaches to nonmonotonic reasoning, our discus-

sion of the basic ATMS is somewhat out of place. Various methods for extending the ATMS so as to support nonmonotonic reasoning have been proposed, however [DeKleer 86b, Junker 88].

5 The Argument Construction Set

The Argument Construction Set (ACS) is the latest in a series of prototype expert system shells developed at the German Research Institute for Mathematics and Data Processing (GMD) for building legal expert systems. The previous system, Oblog-2 [Gordon87], was a hybrid knowledge representation system combining a Prolog-like rule language with taxonomic hierarchies for *types* and *attributes*. Rules were associated with types and nonmonotonic reasoning was supported in a way similar to property inheritance in frame-based systems, as discussed above in the section on inheritance hierarchies. Oblog suffered from a number of difficulties:

1. The terminological component could not infer that one type was a subtype of another type, as is usual for such reasoners. Rather, type (and attribute) relations had to be explicitly asserted by the user.

2. To assert exceptions to rules, new types had to be introduced into the the type hierarchy. These types tended to be rather artifical; they often did not represent conventional terms of the application domain. (For example, there would be a type for contracts with minors, even though, "contract-with-minor" is not a term of art in contracts law.)

3. As in Prolog, solutions to goals were not stored. If the same goal were later encountered, Oblog would begin by again applying the most general rules first, even though it had previously been discovered that some exception applies.

4. Oblog did not have a negation operator. New attributes had to be introduced to express the negation of existing attributes. This is a problem Oblog inherited from Prolog. Although Oblog's rule language was based on Prolog, the special kind of negation used in Prolog was not adopted, as it depends on there being complete information about the facts of some case. (Technically speaking, in legal reasoning the *closed world assumption* cannot be made.)

5. Finally, Oblog was based on the deductive model of legal reasoning. It did not directly support the interactive construction of legal arguments in the way suggest by Fiedler.

The Argument Construction Set addresses most of these deficiencies. It is a completely new system and approaches the problem of nonmonotonic reasoning from another angle. Rather than using the idea of property inheritance, a reason maintenance system supporting nonmonotonic justifications has been adopted. This change has made the object-oriented aspects of Oblog unnecessary. ACS does not presently include a terminological reasoner, but it in principle could be extended to include such a component. We have concentrated in this version on the problems of reason maintenance and supporting the argument construction view of legal decision making.

ACS can be seen as consisting of the following main components, or modules:

Reason Maintenance Module Our RMS is similar to those in [Doyle79,Goodwin87]. That is, it supports nonmonotonic justifications but, unlike an ATMS, is able to manage, in ATMS terms, only one *context* at a time. In contrast to the usual semantics for an RMS supporting nonmonotonic justifications, however, a set of justifications in the ACS always has exactly one extension. That is,

every node can be uniquely determined to be either *in* or *out*, even when nonmonotonic justifications are present.

Tactical Reasoner A lawyer could use the reason maintenance subsystem of ACS alone for constructing arguments. However, a *tactical* reasoner exists for assisting the user in finding new justifications sufficient to make some node either *in* or *out*, as desired. The tactical reasoner is a kind of expert system in its own right. An expert in some domain can build knowledge bases for this component, using a rule language similar, again, to Prolog. The task of the module is to make suggestions to the lawyer regarding interpretations of the law and facts sufficient to make an argument for or against some proposition.

Dialog and Explanation Manager This module is responsible for directing the interaction of ACS with the user. It poses questions to the user about the facts of the case and about whether an interpretation of the law suggested by the tactical component is acceptable. It is able to explain just why some node in the reason maintenance module is either *in* or *out*, and why a question is being asked. Finally, the dialog manager provides tools for editing the set of rules used by the tactical component and for asserting, retracting or modifying the justifications used by reason maintenance system.

At the highest level, ACS assists the user in constructing two kinds of objects: 1) a set of justifications for various propositions regarding some particular case or set of facts, and 2) a set of rules representing interpretations of some area of law. These two objects are stored in separate files as *cases* and *models*. Any number of models can be used by the tactical component to assist the user in constructing arguments for a case. Note that arguments as such are not represented explicitly; they are implicit in the structure of the current set of justifications. The dialog manager constructs the arguments as it explains why some particular proposition is believed or not.

A typical session with ACS would be something like this. The goal of the lawyer is to construct an argument for some proposition. Let's stay with our standard example and suppose the lawyer wants to show that some contract, a, is valid. He would start ACS and then open a new case. A window appears on the monitor for the case. The justifications for the issues which arise in the case will be appear in this window. There are menu commands for asserting, retracting and modifying justifications. The dialog manager assures that justifications have the proper form. At any point, the user can open another window, called the worksheet, to ask whether an argument exists supporting some proposition. That is, the worksheet is used to see whether some node in the reason maintenance net is currently *in* or *out*. There are *buttons* on the worksheet window for issuing commands to the dialog manager. There is a button, e.g, for obtaining explanations.

In our example, the user may first assert a new premise, i.e. a justification with an empty *in list* and an empty *out list*, stating that a is a contract. By asserting this as a premise, the user is in effect saying that he does not expect there to be any dispute as to whether or not a is a contract, or at least that he is confident that this can be proved and does not want to be concerned with the details of this proof at present. After asserting this premise, the user could turn to the worksheet and command ACS to show whether or not $contract(a)$ is *in*. ACS would respond, of course, that it is; if the user then asks why, it would be noted that a premise for this proposition had been asserted. A user would not actually use ACS in this way, as it is quite obvious that the premise he just asserted is *in*; but this simple example illustrates some of the features of a typical interaction cycle with the system. Justifications are asserted, retracted or modified, then the worksheet is

consulted to see if the desired legal conclusions are *in*. If not, the cycle is repeated by first modifying or extending the set of justifications in some way.

Thus, we see that the only function of the reason maintenance system is to ensure that the arguments made by the user in support of some proposition are *sound*. Soundness here has, of course, a nonstandard meaning, as the reason maintenance system is nonmonotonic. The RMS subsystem does not assist the user in interpreting the law or illiciting the facts necessary to support some conclusion. It only insures that arguments made are internally coherent, to satisfy the formal constraint of justice requiring arguments to be logical.

As mentioned above, in our RMS there is always exactly one labelling to be computed, and there is a deterministic algorithm which can compute this unique labelling. As this behavior is at odds with Doyle's semantics for reason maintenace systems supporting nonmonotonic justifications [Doyle 83], our RMS needs some explanation. The basic intuition is this: we conjecture that in legal reasoning every proposition playing a role in an argument must be constructively justified. That is, the proponent of the argument must positively show a reason for believing the proposition. On the other hand, it is not necessary for the propopent to show why some proposition is not believed. That is, we suppose legal reasoning to be skeptical; no proposition is believed unless there is a valid argument made in its support.

Without going into detail, let us contrast this position with the philosophy underlying Doyle's semantics for reason maintenace systems. His concern is whether the beliefs of an agent *may* be true given what is presumed to be true about the actual world. If so, then in Doyle's system such a set of beliefs is justifiable or, using his terminology, admissible. Thus, an agent may believe whatever he likes so long as his beliefs are internally coherent.

Doyle's semantics does require beliefs to be *grounded*, in the technical sense that there be at least one valid justification for the belief in the labelling chosen by the agent (or RMS). We however are suggesting that a stricter kind of groundedness is required in legal reasoning: belief in a proposition is grounded only if the proposition *must* be believed given the current set of justifications. If we take a constraint satisfaction view of reason maintenance, Doyle's semantics allows the node denoting a proposition to be *in* so long as the constraints represented by the justifications are satisfied; our system allows the node to be *in* only if the constraints require the proposition denoted by the node to be believed. Our view of reason maintenance has yet to be formalized. However Konolige, in his comparison of Default Logic and Autoepistemic Logic [Konolige 87], notes that

> Default logic is *brave* in the sense that in the presence of competing defaults, individual extensions will satisfy a maximally consistent set of defaults; in contrast, a natural corresponding circumscriptive theory for defaults would be *cautious*, inferring only what the competing extensions have in common.

This suggests there may be a connection between circumscription and the kind of reason maintenance system we have designed for ACS.

Perhaps the service offered by the reason maintenance component of ACS would be enough to warrant using the system. The tactical component of ACS, however, extends the functionality of the system by making suggestions to the user about how to extend or modify the set of justifications in order to construct an argument in favor of some proposition. The current implementation of the tactical component is rather limited, as will be seen, but in principal it can become arbitrarily complex, including knowledge bases for assisting with the task of interpreting primary sources of law or making recommendations regarding particular arguments based on knowledge about the previous behavior of some particular judge, for example.

The tactical component does not in general presume any particular role or purpose of its user. Different instances of the component can be imagined for advocates defending the interests of some client and for judges whose task, let us assume, it is to further and protect interests other than the limited interests of some particular party to the case before the court.

In his paper criticizing expert systems which presume there are clear legal rules [Leith 85], Philip Leith did suggest that expert systems may play a role similar to that of the tactical component of ACS:

> Thinking beyond the deterministic system it might be considered how systems might be built which can aid the person trying to prepare a case. The system might do this by helping to pinpoint the various potential outcomes — or rule breaking strategies. ... Experts are often wrong, so it seems essential to build legal advisory systems which encapsulate various expert models, and which might allow the user to add to the expertise that he or she has to the decision process — in other words, we are basing our systems upon a "textbook model" rather than a rule based model. ... Commentaries and suggested interpretations of the law might be handled (and easily updated). Also, other factors can be handled which cannot be held in a textbook; for instance, the builder of the system may inform the program that an extremely conservative bias might well be present in the eventual judicial decision, or that past experience has shown Judge X to have a view Y on the type of problem under investigation ...

The current tactical component of ACS uses rule bases consisting of Prolog-like rules representing interpretations of the law. I say 'Prolog-like', as there are important differences. First of all, the language supports ordinary negation rather than Prolog's interpretation of negation as failure. Second, there is an *unless* operator which is used to support nonmonotonic reasoning. Informally, *unless* p can be understood to mean $M\neg p$, using the modal operator M. That is, *unless* p is satisfied if $\neg p$ is consistent. The trick here concerns how consistency is determined. We adopt the same approach used in Goodwin's WATSON system [Goodwin 87]; namely, $\neg p$ is considered to be consistent if p is *out* given the current set of justifications. This approach makes it very easy, and efficient, to determine whether the nonmonotonic conditions of a rule are satisfied. Unfortunately, it is not at all clear just what an appropriate semantics for this approach would be. It does not appear to conform to the semantics of Moores Autoepistemic Logic [Moore 85], for example.

It is not necessary for an expert to develop one unified theory of some area of law in order to build expert systems for use with ACS. Incompatible interpretations of the law can be asserted into a single knowledge base. Alternatively, the end user can load and apply knowledge bases constructed by a variety of experts having incompatible interpretations of the law, even though each expert has done his best to construct an internally consistent view of the law.

The reason consistent knowledge bases are not required by ACS is that the user is asked to accept the interpretation of the law recommended by the expert before it is used to create justifications. Moreover, every node in the reason maintenance system is made explicitly dependent on the rules used to justify the node. This makes it easy to reconsider an interpretation of the law. If the rule is made *out*, by retracting the premise representing the rule, e.g., all the conclusions which depend on that interpretation will also be made out by the RMS.

I mentioned that the current tactical component is rather limited. This is because the knowledge bases used by the component are restricted to collections of rules representing possible or recommended interpretations of some area of law. There is no way presently to represent knowledge about which interpretation to

prefer given the goals of the user. Nor does the tactical reasoner make suggestions about which premises to *retract* or modify to make some node *in*.

To show how the tactical component is used in the current version of ACS, let us continue with the session started above. The user has asserted a premise stating that a is a contract. Suppose he wants to argue that a is valid. Clearly, $valid(a)$ is not currently *in*, as there is not yet a justification for this proposition, let alone a valid justification. At this point the user could load into the system one or models concerning German contracts law and then ask the tactical system to try and *refute* the current belief status of $valid(a)$. As $valid(a)$ is currently *out*, asking the system to refute this is the same as asking it to make suggestions about justifications sufficient to make $valid(a)$ *in*. Refutation always attempts to switch the current label of some node. The tactical subsystem would in this case look for rules capable of supporting an argument in favor of validity, and then try to solve the subgoals of these rules. In the process of trying to find an argument of $valid(a)$, justifications may be asserted by the tactical component, and the user may be asked questions about the facts of the case. The interaction at this stage is very similar to the usual kind of expert system in law.

Supposing that there is a rule stating that contracts are valid unless one of the parties is a minor, the tactical component would suggest asserting a justification to the effect, roughly speaking, that $valid(a)$ is *in*, if $contract(a)$ is *in* and $contract\ with\ minor(a)$ is *out*. The reason maintenance module would then make $valid(a)$ *in*, as there is still no argument to be made supporting $contract\ with\ minor(a)$.

It is important to notice that $valid(a)$ being *in* does not imply that $\neg valid(a)$ is *out*! As far as the RMS is concerned, both of these propositions can be *in* or believed at the same time. It is the users responsibility to avoid contradictions of this sort. The tactical module can, in principle, be extended so as to help the user with this task, however. Reason maintenance does not necessarily mean *consistency* maintenance.

Up to this point, we have imagined that there is a single lawyer using ACS to try to construct an argument in favor of some proposition. ACS could be used by two lawyers simultaneously, however, to engage in a kind of debate or dialog regarding some issue. In this case it is useful to think of ACS as a kind of computer-assisted game. An ACS *case*, in which justifications are asserted and modified, plays the role of the game board. Legal moves of the game are restricted to asserting justifications. The two debaters, or to continue with this game metaphor, *player's*, take opposing positions with respect to main issue. One player tries to make the proposition *in*, the other tries to make it *out*. Players take turns asserting justifications. A players turn is completed when sufficient justifications have been asserted to make the proposition *in* or *out*, depending on the players goal. The game is over when one player is unable to find justifications sufficient to switch the label of the proposition back to the status he is advocating. The other player is then declared the winner.

This game oriented view of ACS is reminiscent of Dialog Logic [Felscher 86], which suggests there may be a connection between reason maintenance and intuitionistic logic. However Dialog Logic seems to me to be 'just' a calculus for 'standard' intuitionistic logic which, unlike reason maintenance systems of the type used in ACS, does not support nonmonotonic reasoning.

I do not want to say too much about the current implementation of ACS here. The implementation is being done by Karsten Schweichart as part of his *Diplomarbeit*, and his realization of the ideas presented here will be described fully in his thesis. It is implemented in Prolog and Modula-2 and runs on Macintosh computers. The user interface was programmed by Kai Diestelmeier.

6 Concluding Remarks

What has been accomplished with ACS? It represents, I believe, an honest and realistic approach to computer-supported legal decision making. It surmounts the problem of the lack of clear legal rules, which Leith is quite right in pointing out is a fundamental problem for legal expert systems based on the deductive model of legal reasoning. Leith suggested that if we try to construct a computer system for supporting legal decision making in the face of a lack of such clear rules, "... we are no longer attempting to design systems with 'artificial intelligence' ..." [Leith 85]. He of course was raising the difficult issue as to just what intelligence is, and when it can be said that a computer sytem exhibits intelligence. Whatever position one wishes to take on this issue, it must be conceded that the technologies necessary to build software systems like the Argument Construction Set are coming out the of *field* of Artificial Intelligence. So I must disagree with Leith on this point. If we are to construct expert systems which assist lawyers with legal reasoning, then AI, in the technical sense understood by researchers in the field, is a *necessity*.

References

[Brewka 87] Gerhard Brewka; *Nonmonotonic Logics: An Introductory Overview*; GMD Research Paper; Number 274; 1987.

[deBessonet 84] C. G. deBessonet; *An Automated Intelligent System Based on a Model of a Legal System*; Rutgers Computer & Technology Law Journal; Volume 10; 1984.

[DeKleer 86a] Johan DeKleer; *An Assumption-Based TMS*; Artificial Intelligence; Volume 28; 1986.

[DeKleer 86b] Johan DeKleer; *Extending the ATMS*; Artificial Intelligence; Volume 28; 1986.

[Doyle 79] Jon Doyle; *A Truth Maintenance System*; Artificial Intelligence; Volume 12; 1979.

[Doyle 83] Jon Doyle; *The Ins and Outs of Reason Maintenance*; International Joint Conference on Artificial Intelligence; Karlsruhe; 1983.

[Etherington 83] D. W. Etherington and R. Reiter; *On Inheritance Hierarchies with Exceptions*; AAAI-83; 1983.

[Etherington 87] D. W. Etherington; *A Semantics for Default Logic*; IJCAI-87; Milan; 1987.

[Felscher 86] Walter Felscher; *Dialogues as a Foundation for Intuitionistic Logic*; in Handbook of Philosophical Logic; Volume III; D. Gabbay and F. Guenthner, eds.; D. Reidel Publishing Company; 1986.

[Fiedler 85] Herbert Fiedler; *Expert Systems as a Tool for Drafting Legal Decisions*; II Convegno Internazionale Logica, Informatica, Diritto; Florence; 1985.

[Genesereth 87] Michael R. Genesereth and Nils J. Nilsson; *Logical Foundations of Artificial Intelligence*; Morgan Kaufmann; 1987.

[Ginsberg 87] Matthew L. Ginsberg, editor; *Readings in Nonmonotonic Reasoning*; Morgan Kaufmann; Los Altos; 1987.

[Goodwin 87] James W. Goodwin; *A Theory and System for Non-Monotonic Reasoning*; Linköping Studies in Science and Technology; No. 165; 1987.

[Gordon 87] Thomas F. Gordon; *OBLOG-2: A Hybrid Knowledge Representation System for Defeasible Reasoning*; First International Conference on Artificial Intelligence and Law; Boston; 1987.

[Gordon 88] Thomas F. Gordon; *The Importance of Nonmonotonicity for Legal Reasoning*; 1988; to appear.

[Junker 88] Ulrich Junker; *Reasoning in Multiple Contexts*; GMD Research Report; in preparation; 1988.

[Konolige 87] K. Konolige; *On the Relation Between Default Theories and Autoepistemic Logic*; In Readings in Non-Monotonic Reasoning; M. L. Ginsberg, ed.; Morgan Kaufmann; Los Altos; 1987.

[Leith 85] Philip Leith; *Clear Rules and Legal Expert Systems*; II Convegno Internazionale Logica, Informatica, Diritto; Florence; 1985.

[McCarty 77] L. Thorne McCarty; *Reflections on TAXMAN: An Experiment in Artificial Intelligence and Legal Reasoning*; Harvard Law Review; Volume 90; 1977.

[McCarthy 80] J. McCarthy; *Circumscription — a Form of Non-Monotonic Reasoning*; Artificial Intelligence; Volume 13; 1980.

[McDermott 80] Drew McDermott and John Doyle; *Non-Monotonic Logic I*; Artificial Intelligence; Volume 13; 1980.

[McDermott 82] Drew McDermott; *Non-monotonic Logic II: Non-Monotonic Modal Theories*; Journal of the ACM; Volume 29; Number 1; 1982.

[Minsky 75] Marvin Minsky; *A Framework for Representing Knowledge*; in Readings in Knowledge Representation; eds. Ronald J. Brachman and Hector J. Levesque; Morgan Kaufmann; Los Altos; 1985.

[Moore 85] Robert Moore; *Semantical Considerations of Nonmonotonic Logic*; Artificial Intelligence; Volume 25; 1985.

[Nilsson 80] Nils J. Nilsson; *Principles of Artificial Intelligence*; Tioga; 1980.

[Reinfrank 86] Michael Reinfrank; *Reason Maintenance Systems*; Workshop on Truth Maintenance Systems; Berlin; October, 1986.

[Reiter 80] R. Reiter; *A Logic for Default Reasoning*; Artificial Intelligence; Volume 13; 1980.

[Roberts 77] R. B. Roberts and I. P. Goldstein; *The FRL Primer*; Memo 408; MIT Artificial Laboratory; 1977.

[Sergot 86] M. J. Sergot, F. Sadi, R. Kowalski, R. A. Kriwaczek, P. Hammond and H. T. Cory; *The British Nationality Act as a Logic Program*; Communications of the ACM; Volume 29; 1986.

[Stamper 80] R. Stampler; *LEGOL: Modelling Legal Rules by Computer*; Computer Science and Law; ed. Niblett; 1980.

[Susskind 87] Richard E. Susskind; *Expert Systems in Law*; Oxford University Press; 1987.

[Touretzky 86] David S. Touretzky; *The Mathematics of Inheritance Systems*; Morgan Kaufmann; 1986.

[Winston 84] Patrick H. Winston and Berthold K. P. Horn; *Lisp*; second edition; Addison-Wesley; 1984.