Oblog-2
A Hybrid Knowledge Representation System
for
Defeasible Reasoning

Thomas F. Gordon*
German Research Institute for Mathematics and Data Processing
Postfach 1240
5205 Sankt Augustin 1
West Germany

April 8, 1987

Abstract

Oblog-2 is a hybrid knowledge representation system comparable to Krypton and KL-TWO. It combines a terminological reasoner with a Prolog-like inference mechanism. The terminological component supports the description of type and attribute taxonomies. Entities are instances of a set of types. Procedures for determining the values of attributes are Horn clause rules indexed by type. The known types of an entity determine its set of applicable rules, which changes as our knowledge about the types of the entity is refined, supporting a form of defeasible reasoning. Oblog-2 has been designed for modeling legal domains. Laws can be represented as general rules with exceptions, a technique traditionally used in the law, together with burden of proof rules, for reaching decisions when less than perfect information is available.

1 Introduction

Oblog, for OBject-oriented LOGic, is an experimental hybrid knowledge representation and reasoning system. Although Oblog is comparable to other hybrid knowledge representation systems such as, for example, Krypton [3], unlike such general purpose systems, Oblog has been designed with a particular field of application in mind, legal expert systems. For some time now, researchers in the field of legal computer science have been concerned with the problem of supporting lawyers with the task of deciding cases under the law. This task has been viewed, naively, as merely a form of deduction: the law concerned and facts of the case must be, somehow, represented in a formal language as *axioms* and the legal results would follow by the application of sound rules of inference as *theorems*. Although deduction is still believed to play a central role in legal reasoning, it has become clear that there are serious problems with this view [5]. One problem, of *knowledge representation*, concerns whether and how legal knowledge can be symbolically represented in some formal language. We must, e.g., discover adequate means of modeling the *open texture* [9] of legal concepts. The meaning of legal concepts tends to change during the application of the law to a case. McCarty's *prototypes and deformations* theory attempts to find computational mechanisms for representing this process [13]. Then there are problems of *computational complexity*. In general it is not feasible to try all combinations of inference rules when trying to derive a theorem. *Heuristic* approaches try to find rules of thumb and other strategies for controlling the size of the set of alternatives to be examined. Additionally, others are examining non-standard logics, such as *intuitionistic* or *relevance* logics, e.g. [14], which have proof procedures computationally less complex than those for classical predicate logic. The difficulty is in finding a suitable balance between computational complexity and expressibility; the more expressive and flexible logics, such as first-order predicate logic, tend to have less effective proof procedures. Finally, there are other important reasoning processes to be examined. General principles are extracted from cases. Here inductive and statistical reasoning plays a role. Moreover, principles are extracted from one area of law and transferred to another, perhaps only remotely related area of law. So we must devote attention to the problems of reasoning by analogy. The Proceedings of the Ninth International Joint Conference on Artificial Intelligence [19] contains a discussion about some of these subfields of legal computer science.

In the *logische Programmierung im Recht* (LORE) project, we at the *Forschungsstelle für Informationsrecht* (FS-INFRE) of the GMD are currently concerned with the knowledge representation and computational complexity problems mentioned above. The Oblog system described here, which is now in its second version, has been designed for describing legal concepts and relations, representing the facts of legal cases, and for interactively supporting the deciding of legal cases. It is hoped that ideas developed in Oblog can be used for legal *expert systems*, and perhaps as a basis for computer-aided legal instruction, although legal education is not the primary focus of our activities.

The first version of Oblog [7] was an extension to MRS [6] for structured knowledge representation. There we tried to retain the truth semantics of first-order logic by interpreting inheritance hierarchies as similarity networks. Type hierarches were used only as a construction aid for knowledge bases; assertions about a type were not inherited by its subtypes. Rather, the

*I would like to thank Reinhard Budde, Gerd Brewka, Reinhard Linz and Marek Sergot for their helpful suggestions. Especially, however, I would like to thank Herbert Fiedler, who, as director of the research group for information law of the GMD, made this work possible. Thanks also to Manfred Fidelak for his help with TEX.

type hierarchy was used merely to remind the knowledge engineer that assertions attached to a supertype may also be applicable to subtypes. It was the responsibility of the knowledge engineer to decide whether some assertion was indeed applicable to some subtype, in which case he was required to make an additional assertion to that effect. This approach suffered from several inadequacies. There was no inference rule that an instance of some type is also an instance of all supertypes of the type, which is clearly counter-intuitive. (Leo the lion could not be shown to be an animal.) Another shortcoming of the previous version of Oblog was its inadequate support for plausible reasoning. We desire some means of making tenative judgments with less than complete information. With additional information, we should be able to reach conclusions that are, in some sense, qualitatively better.

The current version of Oblog also supports structured knowledge representation, although in a somewhat different manner. There is now a terminological component for defining taxonomies of *types* and *attributes*. These are similiar to the *concepts* and *roles* of KL-ONE [2]. *Entities* in Oblog are the structured, frame-like objects. They are instances of types and have attribute values. An entity which is an instance of some type is now also an instance of all supertypes of that type. From a frame language perspective, attributes are similar to *slots*. Oblog *rules* are sets of Prolog-like definite clause procedures for computing attribute values. Each rule is associated with a type and an attribute. There may be multiple rules concerning some attribute for various types.

Oblog supports a form of defeasible, or default, reasoning. Rules are partially ordered by a *generality* relation. Note that the generality of rules is not directly asserted, but is implied by the structure of the type hierarchy. As each rule is associated with a type, its generality can be determined by the level of that type in the type hierarchy. Rules attached to types higher in the hierarchy are considered to be more general than other rules concerning the same attribute attached to lower-level types. This approach avoids the problem of determining relative generality by a direct comparison of the bodies of clauses. In effect, attaching a rule to a type is an *assertion* as to the rule's generality and is a responsibility left to the user. If a rule x is more general than some other rule y, then y is an *exception* to x.

The generality relation is used by Oblog's control structure to determine the sequence in which rules are to be applied. Given a query, q, and some Oblog knowledge base, the Oblog inference machine returns a stream of sets of substitutions for the variables of q, as in Prolog. As general rules are applied before their exceptions, the stream of sets of substitutions is so ordered that rough, approximate solutions appear before more precise ones.

A substitution set, or *frame*, can be considered to be a function mapping terms to terms, where variables in the input term are replaced by their values in the frame. Thus, if q is a query term and f is a frame in the solution stream, then $f(q)$ is the query with its variables replaced by their values in the frame. Such $f(q)$ can be called the "answers" to the query.

Oblog does not have a formal semantics. Informally, however, the intended meaning of answers is as follows. We want to distinquish between answers and theorems. As the rules applied may be subject to exceptions which have yet to be examined, it is not expected that the interpretation of an answer necessarily be a true proposition. However, we can view an answer as a sentence which *would have* a true interpretation if some implicit, qualifying condition is satisfied. This qualifying condition can be determined by examining the remaining exceptions. The complexity of a qualifying condition depends on the number and complexity of the exceptions remaining to be examined. If no exceptions remain, then the answer is unqualified and therefore a theorem. Even when exceptions remain, we can consider an answer to be a theorem if we are willing to *assume* that its qualifying condition is satisfied.

This informal semantics is admittedly very sketchy and should be developed in detail. Note, however, that Oblog's *inference machine*, to be described below, is a syntactic device which does not depend on this semantics, although it is of course motivated by it and must be faithful to it in order to preserve correctness. In particular, qualifying conditions play no role in the inference machine.

The structuring of statutes as general rules with separate exceptions is common practice in legal drafting, and appears to serve a variety of purposes. General rules are shorter and easier to remember and apply. This facilitates the normative function of the law; the law would have little effect on social behavior if its rules were so convoluted that persons could only with great difficulty, if at all, predict the legal consequences of their actions. Generalized rules also permit persons to acquire quickly a superficial understanding of the law, and to deepen their knowledge gradually as the need arises. There are also basic economic considerations supporting this structuring of the law. Perfect information is not available to the courts or the parties about the relevant facts of a case. These facts need to be discovered, which usually entails considerable costs. A careful structuring of general rules and exceptions is one method of allocating the burden of proof, which may be placed on the party for which it is expected that the relevant information is available at least cost. (Other considerations may, of course, override the goal of minimizing costs; the point here is only that general rules with exceptions are one method for taking such factors into account when designing legislation.) Also, by allowing certain facts to be assumed unless challenged by an interested party, the cost of proving such facts can be avoided altogether. This provides a means of tailoring the expense of proving a case to the value of the case to the interested parties. The legal system is ineffective as a means of resolving disputes to the extent the cost of deciding a case exceeds the value of a judgment to the parties. In a dispute over the sale of a bicycle, for example, rational parties will only resort to the legal system if a judicial process is available which entails costs considerably less than the value of the bike. Special procedures, such as small claims courts and arbitration exist for this purpose.

Reflection about the use of exceptions in the law motivated our design of Oblog. The rule/exception structure of the law does not require us, of course, to adopt the same approach for dealing with complexity in our computational models of the law. But the long tradition of this approach in legal systems suggest that this strategy may be useful as well in the context of legal expert systems. Although legal expert systems have their particular requirements, we believe that legal practice and theory can help provide solutions to the problems of large knowledge-based computer systems in general and, indeed, would not be surprised if the rule/exception approach were to find applications outside of the law.

The paradigm of general rules with separate exceptions is certainly inadequate for handling the full variety of knowledge representation problems we face when designing legislation, or building computational models of legislation. The purpose of Oblog is to help make concrete the issues involved in reasoning

with exceptions. Such experiments are an important counterpart to purely theoretical research.

2 System Overview

Figure 1 is an outline of the structure of an Oblog knowledge base. The case-specific information consists solely of a set of entities. The model of a law consists of three sub-components: a type taxonomy, an attribute taxonomy and a set of entities of general applicability. The type and attribute taxonomies are *partial orders* and not limited to tree structures. That is, a type may have arbitrarily many *supertypes* and *subtypes*. (The attribute taxonomy is analogous.) In tree-structured taxonomies, each type may have at most one supertype. Types are used in Oblog to represent *generic concepts* such as persons, contracts, or events. Particular instances of these concepts are represented by entities. Attributes are used to represent the properties of entities and the relationships between entities.

In the discussion that follows, we will be using an example based on a section of the German Civil Code, BGB 108, to my knowledge originally suggested in [17], which has become somewhat of a standard example in Germany: A contract is presumed to be enforceable. A contract with a minor, however, is presumed not to be enforceable. Finally, a contract with a minor that has been ratified by the guardian of the minor is enforceable. BGB 108 includes many more levels of exceptions, but these three will suffice for our purposes.

In *figure 2* we see the structure of types in greater detail. Three types are pictured: contracts, contracts with minors and ratified contracts with minors. The arrows between the types show their relation to one another in the type taxonomy. If an entity is an instance of some type, then it is also an instance of all supertypes of the type. Thus, an entity is an instance of a set of types. For example, a ratified contract is also, of course, a contract.

As depicted in *figure 2*, a type consists of two parts: a set of *recognizers* and a set of *rules*. The recognizers are Prolog-like *clauses* for deciding whether an entity is an instance of the type. The recognizers are not definitions in the usual sense. It may be that we have independent means for determining that an entity is an instance of some type, so it is permissible to assert this fact even when the recognizers for the type would not allow us to conclude type membership. If there are no recognizers for a type, then we can assert that entities are instances of the type, but the Oblog system can be of no help in deciding whether other entities are instances of the type as well.

Oblog handles the problem of the open-texture of legal concepts and rules somewhat more gracefully than Prolog. First, recognition rules state sufficient but not necessary conditions for determining whether an entity of some type exits. Thus it is left open whether or not there are other sets of sufficient conditions. Prolog's use of negation as failure, however, is correct only for the *completion* of a Prolog program [4], which asserts the "only if" halves of the definitions of predicates. Second, if a case should fall under the "penumbra of doubt", and it is decided that the general rule was not intended to cover the case in question, exceptions provide a means of clarifying the law without having to modify the general rule.

The rules of a type are sets of definite clauses for determining the values of attributes of entities of the type. In the figure there are rules for determining whether contracts, contracts with minors and ratified contracts with minors are enforceable. If there is no rule for some attribute of a type, then a rule may be *inherited* from the supertypes of the type. Thus we can associate a rule with the highest type for which it is applicable, and it will automatically be applied to subtypes of the type. Exceptions are rules concerning the same attribute for some subtype of the type of the general rule. Exceptions cancel the applicability of the general rule for entities known to be instances of the subtype. Exceptions, of course, may be themselves subject to further exceptions.

The rules for contracts and ratified contracts here consist of bodiless clauses, known as *facts* in Prolog, but all definite clauses are permitted. As for contracts with minors, instead of asserting clauses for overriding the general rule applicable to all contracts, we have merely cancelled the general rule. Cancelling the general rule prevents us from concluding that a contract known to be with a minor is enforceable, but does not permit us to conclude affirmatively that such a contract is not enforceable. Oblog does not yet support negation, so this is the best we can do. (We have simply avoided addressing the problems of negation in this version of Oblog.)

Figure 3 depicts in greater detail the structure of entities. An entity consists of a set of known types and a set of known attribute values. For example, *c1* here is known to be a contract. Although each of the entities in the diagram have been asserted to be instances of one type each, in general an entity may be asserted to be an instance of more than one type. We may wish, for example, to assert that some person is both a German and a lawyer. Attributes may be *multivalued*. The contract for example has been asserted to have two parties, Jeffrey and Hildegard. Unfortunately, there is no way currently in Oblog to place constraints on the number or type of potential values of an attribute. The types and attribute values asserted for an entity represent only the information initially known about the entity. Additional types and attribute values may be inferred by applying the rules of the known and inferred types of the entity.

3 The Inference Engine

The task of the Oblog *inference engine* at any moment is to determine the types and attribute values of some entity. One limitation of the current version of Oblog is that the inference engine is not capable of inferring the *existence* of some entity. The user must assert that the entity of interest exists, and some initial information about the entity. In particular, the entity must be asserted to be an instance of at least one of the types in the legal model. So, for example, we may assert that there is a contract, *c1*, and use Oblog together with a theory of contract law to help us determine whether the contract is enforceable. How do we go about making this determination? We need to find the rules which make conclusions about enforceability and find some sensible order in which apply them. This is where Oblog's *control structure* comes into play. The inference engine first examines what is known about the entity in question. In our example, all that is known is that *c1* is a contract and that there are two parties, Jeffrey and Hildegard (see *figure 3*). Next we look at the known types of the entity for rules regarding enforceability. In *figure 2* we see that there is a rule for contracts regarding enforceability, namely that all contracts are enforceable. (The single clause of the rule in this case has no conditions. That is, in Prolog terminology, it is a clause without a *body*. As mentioned above, however, an Oblog rule may consist of a set

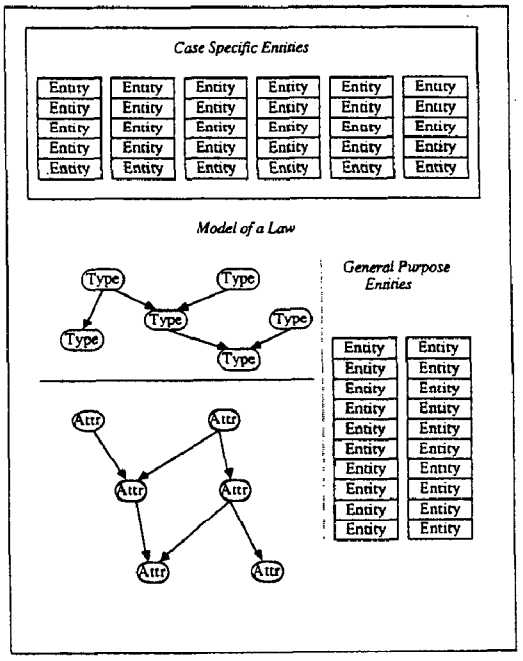

Figure 1: Knowledge Bases

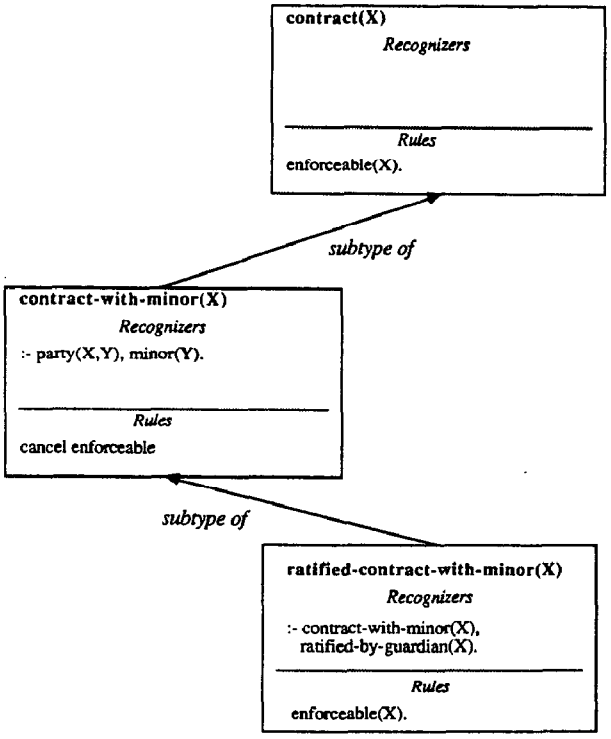

Figure 2: Type Hierarchies

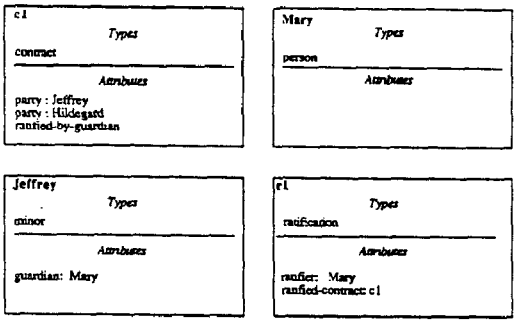

Figure 3: Entities

of arbitrary Prolog clauses concerning the attribute. To satisfy the goals of a clause, if there are any, we would apply the same control strategy we are now describing to each of the subgoals.) This gives us our first solution.

As there are exceptions to the general rule that contracts are enforceable, our first solution is subject to an implicit qualifying condition that the contract is not a contract with a minor. We have devised a procedure for measuring the complexity of the qualifying condition of a solution. This measure is a kind of *certainty factor*, and one should be careful not to confuse this measure with some notion of probability. The measure here is just an index in the range of zero to one giving a rough notion of how many exceptions remain to be tried. If no exceptions remain in the model, or if the proposition was asserted by the user, then it is given an index of 1. Should we wish to probe deeper into an issue, either because we are unsatisfied with the level of certainty of the solution, or because it would be beneficial to our position to find a different solution, we ask Oblog to look for exceptions to the rules previously applied. In our example Oblog would next find the rule regarding the enforceability of contracts with minors and then the rule for ratified contracts with minors.

To summarize, the basic control strategy of Oblog is as follows: apply the relevant rules of the known types of the entity, inheriting rules from the supertypes of the known types if necessary and possible. Then, if a more precise analysis is desired, look for exceptions in the subtypes of the known types. A subtlety here is that, before an exception may be applied, we must show that the entity in question is also an instance of the subtype containing the exceptional rule. In our example, we must demonstrate that *c1* is a ratified contract with a minor, which was not directly asserted, before we may use the rule that such contracts are enforceable. For this purpose, we apply the recognizers for the subtype. In contrast to ordinary rules, recognizers do not have exceptions. Nonetheless, we may conclude with less than complete certainty that an entity is an instance of some type, as each of the subgoals in the recognizer may be satisfied by applying general rules subject to exceptions. It is not required that an entity be shown with complete certainty to be an instance of some type before that type's rules may be applied. But the certainty here will have a bearing on the certainty of the solutions to our top-level goal, should solutions be found using rules of the subtype.

Oblog's control strategy may be viewed as an SLD-refutation procedure whose *ordering rule* [11] traverses the directed net of Oblog rules defined by the generality relation. In Prolog, clauses are selected in the (fixed) order in which they appear in the program. In Oblog, we use the type taxonomy for ordering clauses. The Oblog ordering rule must also consider "facts", i.e. ground unit clauses arising from the asserted entities of the knowledge base. Such facts are applied before rules, and general rules are applied before their exceptions.

Facts arise from various sources in Oblog. The asserted types and attributes of an entity are facts. Facts are also implied by the structure of the type and attribute taxonomies of a knowledge base. Special purpose inference procedures are used for deducing type and attribute relationships. These inference procedures generate candidate ground unit clauses for use during resolution.

Only "relevant" rules are applied. Which rules are relevant is a function of the predicate of the goal and the known types of the entity concerned. That is, rules are indexed by predicate and by type. The ordering of the rules is based on the partial ordering of their associated types. If the type of a rule, x, is a supertype of the type of some other rule, y, then x is more general than y and precedes it in the list of rules to be tried. This scheme is complicated somewhat by the attribute hierarchy. Not only the rules for the attribute denoted by the predicate of a goal may be relevant, but also the rules for all subattributes of that attribute. In effect there is an Oblog rule of inference which states that if a_1 and a_2 are attributes and a_2 is a subattribute of a_1 then, for all entities e, $a_2(e) \rightarrow a_1(e)$.

To be more precise, given a unit goal, g, and knowledge base, kb, our ordering rule is defined by the following functions, where *kb-clauses* is the top-level function returning the ordered sequence of clauses:

kb-clauses: [goal × knowledge base] → clause*
; the relevant clauses of a knowledge base concerning a goal
kb-clauses(g, kb) =
  if the entity argument of g is bound then
    entity-clauses(g, e), where e is the entity denoted by the first argument of g
  else
    interleave(entity-clauses(g,e_1) ... entity-clauses(g,e_n))
    where $e_1 \ldots e_n$ are the asserted entities of kb

entity-clauses: [goal × entity] → clause*
; the relevant clauses of an entity concerning a goal
entity-clauses(g,e) =
  if the predicate of g, p, denotes a type then

if e is known to be an instance of the type, checking the taxonomy if necessary, then the ground unit clause $p(f)$, where f is the functor naming e
else
the clauses of the recognition rule for the type denoted by p
if p denotes an attribute then
append(fact-clauses(g,e), rule-clauses(g,e))

fact-clauses: [goal × entity] → clause*
; clauses for the asserted attributes of an entity
fact-clauses(g,e) = $p(f,v_1)\ldots p(f,v_n)$ where p is the predicate of g, f is the functor representing e, and each v_i is an asserted value for e of the attribute denoted by p, or subattribute of the attribute denoted by p

rule-clauses: [goal × entity] → clause*
; clauses for the relevant rules of an entity concerning a goal
rule-clauses(g,e) =
interleave(append(applicable-rule(t_1,a_1) ...
applicable-rule(t_n,a_1))
⋮
append(applicable-rule(t_1,a_n) ...
applicable-rule(t_n,a_n))
where $a_1 \ldots a_n$ is a sequence of attributes; a_1 equals the attribute denoted by the predicate of g, and each subsequent a_i is a subattribute of a_1 such that if a_x is a superattribute of a_y then $x < y$
and $t_1 \ldots t_n$ is a sequence of types, including each of the asserted types of the entity e and all of the subtypes of these asserted types such that if t_x is a supertype of t_y then $x < y$

applicable-rule: [type × attribute] → clause*
; clauses of rule asserted for some type, or inherited
applicable-rule(t,a) =
let l = the lookup function of a *; see discussion below*
if $l(t) \neq nil$ then
if $l(t) = nullrule$ then
the empty sequence of clauses
else the clauses of the rule of $l(t)$
else append the clauses of each of the rules obtained by applying l to each of the parent types of t

The *interleave* function used in *kb-clauses* and *rule-clauses* merges a list of streams together into one stream. The first element of each input stream appears in the output stream before the second element of an input stream, and so on. We use *interleave* instead of *append* here so that the general rules of each stream are applied before their exceptions.

The *lookup* function mentioned in *applicable-rule* maps a type to either the *nullrule* or a list of clauses, ($type \rightarrow \{nullrule\} + clause*$), and is defined extensionally by a set of rule statements. That is, *lookup* is essentially a dictionary associated with an attribute. Each rule statement adds a new entry to the dictionary. If for some *type* × *attribute* pair, (t,a), no rule statement has been asserted, then $l_a(t) = nil$, where l is the lookup function of a. This should be contrasted with the case where a rule statement with an empty set of clauses has been asserted, in which case $l_a(t) = nullrule$. The *nullrule* is used to cancel inheritance, as will be shown in the following example. If no rule has been asserted for (t,a), then *applicable-rule* appends and returns the applicable-rules of each of the parent types of t. This is Oblog's form of *multiple inheritance*. Notice that more than one rule may be inherited. The clauses of all inherited rules are applied, and are considered alternatives. Another strategy would be to assume that the set of inherited rules are conflicting. Criteria for selecting one of the rules would then be required.

Returning to BGB 108, let us restate the example as a sequence of Oblog statements. (The syntax used here is not the concrete syntax of the current Lisp implementation.)

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

Type statements have three parts: the name of the new type, a list of supertypes and a recognition rule, which is a list of clauses. The supertypes and recognition rule are optional. Rule statements include the names of a type and attribute, followed by a list of clauses, which may be empty. Attribute statements, i.e. relation and property statements, include the name of the new attribute, followed by a list of superattributes. The superattributes are optional.

Note here that recognition rules may, but need not, include "type-checking" subgoals. The recognition rule for contracts with minors in statement 4, e.g., does not include a *contract(X)* subgoal; but the recognition rule for ratified contracts with minors in statement 6 does include a *contract-with-minor(X)* subgoal. Such "type-checking" subgoals are not added automatically by the Oblog interpreter. Again, recognition rules are not definitions. They merely state sufficient conditions for showing that an instance of the type exists. In the example, it is assumed that only contracts have parties. Thus, as $\forall x(\exists y(party(x,y)) \rightarrow contract(x))$ is the case, a *contract(X)* subgoal in the recognition rule of statement 6 would be redundant.

In statement 9, note that no clauses have been asserted. The sole purpose of this *nullrule* is to cancel inheritance. It prevents the general rule of statement 8, that contracts are enforceable, from being applied to contracts which have been asserted to be contracts with minors. (See *figure 2*.)

The case specific information can be represented as:

```
11) entity c1 (contract)
        party : Jeffrey
        party : Hildegard
        ratified-by-guardian
12) entity Jeffrey (minor)
13) entity Mary (person)
14) entity r1 (ratification)
        ratifier : Mary
        ratified-contract : c1
```

Entity statements can be considered syntactic sugar for a set of ground unit clauses. Statement 11, e.g., is equivalent to

```
contract(c1)
party(c1,Jeffrey)
ratified-by-guardian(c1)
```

Figure 4 depicts the Oblog search tree showing the order in which rules are applied in the BGB 108 example. The nodes of this tree are goal clauses. The root node is our query. The arcs are labelled by the statement in the knowledge base used in the inference step, together with a certainty factor. (Certainty factors are discussed in more detail below.) More than one clause may be represented by a single Oblog statement, so it may be, as in the case of statement 11 here, that there are references to a single statement for solving various kinds of goals.

Our goal is to show that *c1* is enforceble. As *c1* is a constant, we need only look at the entity denoted by *c1* for relevant clauses. We see that *c1* is asserted to be a contract and look for rules concerning enforceability of contracts, finding the clause generated by statement 8. This clause has no body so we immediately succeeed. On backtracking, we search for additional clauses regarding enforceability for subtypes of contract. The *nullrule* has been asserted for contracts with minors, so there are *no* clauses there to be tried. The failure branch of the diagram has been drawn in another pattern to indicate that we do not mean the usual kind of failure here, where clauses have been unsuccessfully tried; the branch is depicted merely to show the effect of the *nullrule* on the certainty factor of the first solution. Had we asserted that *c1* was a contract with a minor, this *nullrule* would have blocked the applicability of statement 8, and thus our first solution. Here, however, we merely continue searching for relevant clauses, finding the rule for ratified contracts with minors. This clause then leads to a second proof of the enforceability of the contract. Notice that, before we can apply the rule regarding the enforceability of ratified contracts with minors, we must first show that *c1* is such a contract. In contrast to recognition rules (cf. above), Oblog *does* automatically introduce "type-checking" goals before applying exceptions. This is the only place where subgoals are automatically inserted.

Each clause in the knowledge base is associated with a *certainty factor*. Unlike certainty factors in production rule systems such as Emycin [20], in Oblog these factors are implicit in the structure of the knowledge base. Their purpose is merely to suggest to what extent exceptions remain to be tried. A rule with no exceptions has a certainty factor of 1, as do facts. If there are exceptions, the certainty of the rule currently depends only on the number of exceptions, no matter how they are distributed below the type of the rule. More precisely, the certainty factor of a rule is $1/(1+n)$, where n is the number of exceptions, i.e. rules concerning the same predicate for subtypes of the type of the rule.

Returning to our example, the certainty of the rule regarding the enforceability of contracts, for example, is 1/3, as there are two exceptions, one for contracts with minors and one for ratified contracts with minors. We also speak of the certainty of nodes in an Oblog search tree. The certainty of a node is simply the minimum of the certainties of the arcs to the node. Thus, in the example, the certainty of the first solution is 1/3 and the certainty of the second solution is 1. In this example, at least, certainty increases as we search for alternative proofs, which is the behavior we are striving for in general.

It may be argued that two "yes" answers to the question of whether or not *c1* is enforceable are not very informative. A good interface to Oblog should display upon request, for each answer, its certainty factor, the implicit qualifying conditions, and an explanation of how the answer was computed.

4 Problems

It should be emphasized that Oblog is an experimental system under development. There is, however, a prototype implementation of the version described here, in Scheme, a dialect of Lisp. First, a minimal Prolog interpreter was implemented. This Prolog "core" was then modified and extended to create Oblog.

The current version suffers from a variety of problems. One problem, which Oblog inherits from its Prolog core, is that its control component has no memory of goals already solved or the state of its reasoning about some problem. If it faces a problem (e.g. a subgoal) again during the solving of some larger problem, it begins again at the top of the rule hierarchy, instead of picking up the problem where it last left off. Returning to our example, once we have determined that some contract is a contract with a minor, this information is lost. If we later ask whether the contract is enforceable, we will again first apply the general rule that contracts are enforceable. Some kind of *reason maintenance system* (RMS), perhaps along the lines suggested in [10] is required. Conclusions, along with their justifications, would be cached by the RMS, and the RMS would be consulted by Oblog to see if a problem has been previously addressed before applying the usual Oblog control strategy. An RMS for Oblog would be somewhat different, if not necessarily more complex, than an RMS for, say, first-order predicate logic, in that the conclusions cached may be defeasible. Some type of *continuation*, representing the exceptions remaining to be tried, would need to be stored in addition to the justifications.

Other problems concern Oblog's terminological component. The relationship between two types in Oblog must be asserted. Other terminological reasoners, such as those of Krypton [3] and Login [1], can infer such relationships by a direct comparison of the *descriptions* of the types. The descriptions are the types. Types may be named but, unlike in Oblog, may also be anonymous, just as in Lisp unnamed procedures can be described using *lambda* expressions. Also, Oblog does not provide a facility for stating or enforcing value and number constraints of attributes. It would be convenient, for example, to be able to state that every person must have exactly two parents, a mother who is a woman and a father who is a man.

A relatively minor problem concerns the method of defining exceptions to general rules. As rules are tied to types, a subtype must be created in order to define an exception. These types are usually very artificial; they do not describe technical terms or concepts already accepted in some field of law. In the BGB 108 example, we had to define new types, *contract-with-minor* and *ratified-contract-with-minor*, neither of which belong to the terminology of the German Civil Code, in order to override the general rule that contracts are enforceable. A solution to this problem is to divorce the rules from the type hierarchy and introduce a separate rule hierarchy. This solution would take us a step away from the object-oriented programming metaphor which originally motivated our design of Oblog. The idea had been to combine a frame language (the object-oriented aspect)

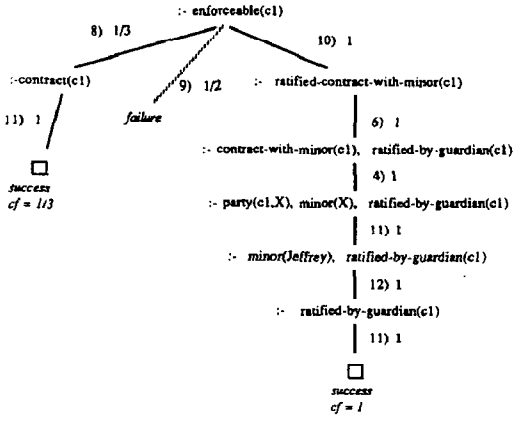

Figure 4: Search Tree

with Prolog, where Horn clauses would be used instead of some procedural language to define the methods for calculating slot values. Nonetheless, a separate rule hierarchy appears to be an attractive possible extension.

A separate rule hierarchy alone would not lessen the need for type and attribute hierarchies. These hierarchies allow a class of inferences to be made for which, if the type and attribute relations were expressed in clauses, an SLD-refutation proof procedure using "blind" depth-first search would not terminate.

Finally, Oblog does not have a negation operator. Prolog uses negation as failure, which is a weaker form of the *closed world assumption*, a nonmonotonic inference rule stating that $\neg p$ is derivable if it is not the case that p is derivable, where p is a ground atom [11]. Negation as failure is correct only for the *completion* of a set of Prolog program clauses [4], as mentioned above. To be able to assert the completion of a Prolog program, we must have perfect information about the application domain. In the context of legal expert systems, we are not able to make this assertion precisely because we know that perfect information is not available about the facts of cases. So, although it would have been convenient to adopt Prolog's approach to negation in Oblog, this option is not available and we must search for other solutions.

Note that Prolog is a nonmonotonic system. Negation as failure is a kind of default rule. Oblog too is a nonmonotonic system, as *additional* exceptions modify the implicit qualifying conditions of answers, and thus may cause some formulas to no longer be theorems. Oblog however includes no blanket, all-purpose default rule comparable to negation as failure. Nonmonotonicity as such, apparently, is not enough. Clearly, some type of negation is desired. We would like to conclude, provisionally, that contracts with minors are *not* enforceable. How to achieve this remains an open question.

5 Conclusion

Oblog has been used to build a variety of "toy" legal expert systems. We intend to improve the efficiency and user-interface of the current version, and are considering ways of using the system in courses for law students on legal applications of AI technology. We intend to begin a more methodical comparison of various approaches to defeasible and nonmonotonic reasoning in 1987, including Reiter's default logic [18], McCarthy's circumscription [12], Michalski and Wilson's "variable precision" logic [15], and Nute's Logic for Defeasible Reasoning, *LDR* [16]. We felt it important to first build a prototype system for building experimental legal expert systems, however ad hoc, to fuel our imagination about the kinds of services such systems might eventually provide and to gain practical experience in AI programming methods.

References

[1] Aït-Kaci, H. and Nasr, R.; *Login: A Logic Programming Language with Built-in Inheritance*; The Journal of Logic Programming; vol. 3; pp. 185-215; 1986.

[2] Brachman, R. J. and Schmolze, J. G., *An Overview of the KL-ONE Knowledge Representation System*; Cognitive Science; 9(2); 1985; pp. 171-216.

[3] Brachman, R. J.; Gilbert, V.P.; Levesque, H.J.; *An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton*; Proceedings of the Ninth International Joint Conference on Artificial Intelligence; 1985.

[4] Clark, K. L.; *Negation as Failure*; in Logic and Databases; H. Gallaire and J. Minker (Eds.); Plenum Press; New York; pp. 293-322; 1978.

[5] Fiedler, H.; *Expert Systems as a Tool for Drafting Legal Decisions*; Proceedings of the Second International Congress on Logic, Computer Science, and Law (Logica, Informatica, Dirritto); Florence; 1985.

[6] Genesereth, M. R. and Smith, D. E.; *Meta-Level Architecture*; Stanford Heuristic Programming Project; Memo HPP-81-6; 1982.

[7] Gordon, T. F.; *Object-Oriented Predicate Logic and its Role in Representing Legal Knowledge*; In Computer Power and Legal Reasoning; ed. Walter, C.; West Publishing Co.; 1985.

[8] Gordon, T. F.; *The Role of Exceptions in Models of the Law*; in Formalisierung im Recht und Ansätze juristischer Expertensysteme; J. Schweitzer Verlag; Munich; 1986.

[9] Hart, H. L. A.; *The Concept of Law*; Oxford University Press; 1961.

[10] de Kleer, J.; *An Assumption-based TMS*; Artificial Intelligence; vol 28; pp. 127-162; 1986.

[11] Lloyd, J. W.; *Foundations of Logic Programming*; Springer Verlag; 1984.

[12] McCarthy, J.; *Circumscription — a Form of Non-Montonic Reasoning*; Artificial Intelligence; vol. 13; pp. 27-39; 1980.

[13] McCarty, L. T. and Sridharan, N. S.; *The Representation of an Evolving System of Legal Concepts: II. Prototypes and Deformations*; Proceedings of the Seventh International Joint Conference on Artificial Intelligence; pp. 246-53; 1981.

[14] McCarty, L. T.; *Fixed-Point Semantics and Tableau Proof Procedures for a Clausal Intutionistic Logic*; LRP-TR-18; State University of New Jersey, Rutgers; 1986.

[15] Michalski, R. and Winston, P.; *Variable Precision Logic*; Artificial Intelligence; vol. 29; pp. 121-146; 1986.

[16] Nute, D.; *LDR: A Logic for Defeasible Reasoning*; FNS-Bericht-86-11; Forschungstelle für natürlich-sprachliche Systeme; Universität Tübingen; 1986.

[17] Philipps, Lothar; *Der Computer als Hilfsmittel zu einer interessengerechten Normierung*; DVR Beiheft 17; J. Schweitzer Verlag; Munich; 1984.

[18] Reiter, R.; *A Logic for Default Reasoning*; Artificial Intelligence; vol. 13; pp. 88-92; 1980.

[19] Rissland, E. L.; *AI and Legal Reasoning*; Proceedings of the Ninth International Joint Conference on Artificial Intelligence; pp. 1254-1261; 1985.

[20] vanMelle, W.; *A Domain-Independent System that Aids in Constructing Knowledge-Based Consultation Programs*; Stanford Heuristic Programming Project; Report No. STAN-CS-80-820; 1980.