A Language for Legal Discourse
I. Basic Features

L. Thorne McCarty
Computer Science Department
and
Faculty of Law
Rutgers University

1 Introduction

In two previous papers on the prospects for intelligent legal information systems [22, 27], I advocated the development of "deep conceptual models" of particular legal domains. My motivation was both practical and theoretical. On the practical side, I argued, our long-term goal should be an integrated analysis/planning/retrieval system that matches as closely as possible the way a lawyer actually thinks about a legal problem. On the theoretical side, my work with Sridharan on the TAXMAN project [31, 32, 44] had clarified the importance of an adequate domain theory in any attempt to model the arguments of lawyers in hard cases. For both purposes, I claimed, deep conceptual models are essential.

Although some commentators have expressed puzzlement about the meaning of the term "deep conceptual model" [45, pages 149-155], the basic idea is easy to state. There are many common sense categories underlying the representation of a legal problem domain: space, time, mass, action, permission, obligation, causation, purpose, intention, knowledge, belief, and so on. The idea is to select a small set of these common sense categories, the ones that are most appropriate for a particular legal application, and then develop a *knowledge representation language* that faithfully mirrors the structure of this set. The language should be formal: it should have a compositional syntax, a precise semantics and a well-defined inference mechanism. The semantic interpretation of the common sense categories should be intuitively correct, that is, it should generate exactly those entailments that ordinary people (and ordinary lawyers!) generate in similar situations. The inference mechanism for the language should be complete and sound, in principle, but, in practice, completeness and soundness would often be sacrificed for computational tractability, just as they are in ordinary human (and ordinary legal!) reasoning. Clearly, if a language of this sort could be developed, it would provide a uniform framework for the construction of a legal analysis/planning/retrieval system, and a solid foundation for further theoretical work.

In this paper, I will describe the basic features of a *Language for Legal Discourse (LLD)*, which takes a first concrete step towards the realization of this goal. An example of the surface syntax of *LLD* is shown in Figure 1. This example is from the TAXMAN II project, and it shows the influence of the frame-based representation language AIMDS [43] in which TAXMAN II was originally implemented. The difference lies in the fact that each syntactic feature of *LLD* has a precisely defined semantic interpretation and an associated inference procedure, which was not the case for the AI representation languages of the late 1970's. The design of *LLD* has also been influenced by the work of Bonner [6] and McGuinness [33] on a seller's remedies for breach of contract under Article 2 of the Uniform Commercial Code, and by my joint work with Dean Schlobohm on estate planning with prototypes [40]. As of this writing, the language has been partially implemented (in Common LISP on a SUN/3 workstation): the various inference mechanisms have been specified, a parser from the surface syntax of Figure 1 to the compiled proof procedures has been written, and a unification algorithm that handles sorted count terms and mass terms (see Sections 2.2 and 2.3) has been thoroughly tested.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© ACM 0-89791-322-1/89/0600/0180 $1.50

However, most of the work on *LLD* to date has been devoted to the theoretical foundations of the language [23, 24, 25, 26, 28, 29, 30].

My plan is to write several papers on a *Language for Legal Discourse*, of which this paper is the first. Section 2 explains some of the peculiar features of the atomic formulae in *LLD*, and Section 3 describes the rules and proofs for the first-order sub-language, which is intuitionistic rather than classical. Section 4 then explains how these rules and proofs are extended to handle various modal features, such as time, action, permission and obligation. One important issue in the design of *LLD* concerns the integration of these several features into a single language, and this issue is addressed in Section 5. My point here is that the intuitionistic semantics at the core of the language facilitates the development of an integrated system, and simplifies the proof procedures for the various modalities.

2 Atomic Formulae

An atomic formula in *LLD* has both an internal syntax, for use by the proof procedures, and a surface syntax for communication with an external user. Some examples of the internal syntax are:

```
(Own O1 (Actor A) (Property P)),

(Own 'Own-2 (Actor 'John)
            (Property 'Book-3)),

(Own - (Actor 'John) (Property 'Book-3)),
```

in which constants are quoted, and the symbol '-' denotes an anonymous existential variable. The surface syntax adds identifiers to the arguments:

```
(Own - {subject (Actor A)}
       {object (Property P)}),
```

and permits relationships to be inverted, so that we can talk about "an actor A who is the subject of the ownership of a property P." Several examples of this construction are evident in Figure 1. Note also in Figure 1 that the same syntactic form is used for objects, relations and actions, and this syntactic uniformity is extended to permissions, obligations and the other modalities as well.

Since this treatment of atomic formulae is standard, this section will discuss only three points that are somewhat distinctive: (1) the use of reified relationships; (2) sorts and subsorts; (3) count terms and mass terms.

2.1 Reified Relationships

Note that every relationship in *LLD* is treated as an individual object, either a constant or a variable. Thus we can talk about an "ownership O1" or the "ownership 'Own-2". There are several reasons for this: First, the reification of relationships accords with common linguistic practice, which means that it has become part of legal practice as well. Second, it provides a useful technical device for representing changes in the state of the world, as in the "holds" formalism of Kowalski [17]. Third, the device can be generalized in a natural way to represent individual events, actions, obligations, beliefs, etc., and it therefore contributes to the integration of several modalities in a single system. These advantages have been recognized by several authors [19, 5, 16], but the logical status of reified relationships is unclear. My own view on this issue is developed in a forthcoming paper [30].

2.2 Sorts and Subsorts

All terms in *LLD* are *order sorted*, e.g., `Person < Actor`, `Corporation < Actor`, and the unification algorithm respects the sorts. This means: (1) the unification of two variable terms succeeds if the sorts of both variables have a common subsort; and (2) the unification of a constant term and a variable term succeeds if the sort of the constant term is less than or equal to the sort of the variable term. In either case, the sort of the unified term is the common subsort. This treatment is standard, and is based on [3, 35, 11].

Somewhat more novel is the treatment of exclusive and exhaustive subsorts. Suppose we wanted to say that an Actor cannot be both a Person and a Corporation. This could be represented by a Horn clause (see Section 3.1):

```
FALSE  <==  (Person A)

            AND  (Corporation A),
```

in which the atom FALSE denotes a contradiction. To say that the two subsorts Person and Corporation are mutually exhaustive, we could write:

```
(Person A)

<==  (Actor A)

     AND NOT (Corporation A)
```

and

```
(Corporation A)
```

```
<==  (Actor A)

     AND NOT (Person A),
```

using the intuitionistic negation rules from Section 3.2. Because of the characteristics of intuitionistic negation, however, the effects of these rules can be efficiently encoded in the unification algorithm itself. I will develop this point further in a future paper.

2.3 Count Terms and Mass Terms

Person and **Corporation** are count terms; **Cash** and **Stock** are mass terms. Either form can be used in *LLD*, but the semantic interpretation in each case is different. A mass term is treated as an infinite set of infinitesimal particles, and it may have one or more *measures* attached to it:

```
(Cash P {subjectof
          (Value -
            {unit (Dollar -)}
            {quantity (Number N)}})}).
```

Although the logic of mass terms is complex in general [9], the intuitionistic semantics of Section 3.2 allows a substantial simplification in which Horn clauses involving mass terms have a structural similarity to Horn clauses involving count terms. As a result, all inferences involving mass terms can be incorporated into the proof procedures that work for count terms. I will develop this point further in a future paper.

3 Rules and Proofs

All rules in *LLD* have a standard form: The left-hand side of the rule is an atomic formula, and the right-hand side is a compound expresssion. If the right-hand side is a conjunction of atomic formulae, of course, the rule is a Horn clause (see Section 3.1). But the right-hand side of the rule could also be a *negation* or an *embedded implication* (see Section 3.2), and it could include a *default expression* (see Section 3.3). By using an intuitionistic semantics for *LLD*, we guarantee that the proof procedures for these more complex expressions have some of the same computational properties as the proof procedures for Horn clauses [25, 26, 28]. Finally, although the standard form of a rule does not allow disjunctive assertions, we can achieve a similar effect by using *prototypes and deformations*, as described in Section 3.4.

All of these features are necessary for the proper representation of legal rules. For example, consider §1.-(2) of the British Nationality Act, which was shown in [41] to pose difficult problems for a language restricted to Horn clauses:

1.-(2) A new-born infant who, after commencement, is found abandoned in the United Kingdom shall, unless the contrary is shown, be deemed for the purposes of subsection (1)

(a) to have been born in the United Kingdom after commencement; and

(b) to have been born to a parent who at the time of the birth was a British citizen or settled in the United Kingdom.

Stripped to its essentials, §1.-(2)(b) says that at least one of the parents of an abandoned infant is presumed to be a British citizen, unless proven otherwise. This is clearly a default rule. See Section 3.3. To show the contrary, we would have to identify *both* parents and show that *neither one* is a British citizen. Thus, for each parent, we would have to prove a negative fact. In Section 3.2, I suggest that the proper approach here is to *assume* that the mother (respectively, the father) is a British citizen, and then to try to show that this assumption leads to a contradiction. But the only way that the mother (respectively, the father) could have become a British citizen is by the operation of the statute itself, or by the operation of a prior statute, and thus the sufficient conditions for British citizenship listed in the statute (and its predecessors) would be construed as necessary conditions as well. See Section 3.4. Taking §1.-(2)(b) literally, then, we would have to show that every possible route by which the mother or father could have acquired British citizenship leads to a contradiction, and this requires a general mechanism for constructing disjunctive proofs. In practice, of course, a *prima facie* showing on the major categories of British citizenship would probably be sufficient to shift the burden of proof back to the other party. This phenomenon can be explained by a theory of prototypes and deformations.

The actual definition of British citizenship is fairly complex. In the following sections, I will illustrate these rules with much simpler examples: controlled corporations; sterile containers; unowned properties; unemployed dropouts; and red and green blocks.

3.1 Horn Clauses

Since many legal rules can be represented by Horn clauses [1, 21, 41], these are one of the more important building blocks of *LLD*. The following is an example of a Horn clause in *LLD* syntax:

```
(Control -
  {subject (Actor A)}
  {object (Corporation C)}})}
```

```
<==   (Own -
        {subject (Actor A)}
        {object (Stock S)})

      AND

      (Issued -
        {subject (Corporation C)}
        {object (Stock S)}
```

This is a simplified version of the definition of "control" in §368(c) of the Internal Revenue Code. For a more realistic representation of a fragment of the Internal Revenue Code in Horn-clause logic, see [21]. I will assume that the reader is familiar with Horn-clause logic programming, however, and I will not discuss it further here.

One point to note: The logic of Horn clauses is the same whether interpreted classically or intuitionistically [25].

3.2 Negations and Embedded Implications

To extend *LLD* further, we allow Horn clauses to be *embedded* on the right-hand side of a rule. For example, we can say that "C is a sterile container if every bug B inside C is dead." This rule would be written in *LLD* syntax as follows:

```
(Sterile - {object (Container C)})

<==   FOR ALL (Bug B):

      (Dead - {object (Bug B)})

      <==

      (Inside - {subject (Bug B)}
                {object (Container C)}).
```

A similar construction is used for negation. For example, we can say that "P is unowned property if, for every actor A, it is not the case that A owns P." This rule would be written as follows:

```
(Unowned - {object (Property P)})

<==   FOR ALL (Actor A):

      FALSE  <==

      (Own - {subject (Actor A)}
             {object (Property P)}).
```

Note that we are construing 'NOT $\mathcal{A}$' as an abbreviation for 'FALSE <== $\mathcal{A}$'.

If the negation and embedded implication rules were interpreted classically, they would be equivalent to full first-order logic. But interpreted intuitionistically, as explained in [25, 26], they generate a proper subset of first-order logic with useful semantic and computational properties. First, they possess an analogue of the *unique minimal model* property of Horn-clause logic [46, 4], so that every successful query has a definite answer substitution, exactly as in PROLOG. Second, and closely related, the *tableau proof procedure* for these rules is a straightforward generalization of SLD-refutation for Horn clauses.

For example, suppose we wanted to show that a particular container, 'PetriDish-1, is sterile. We would begin our proof in the *initial tableau* $\mathcal{T}_0$. When we encounter the definition of a "sterile container," however, we would construct an *auxiliary tableau* $\mathcal{T}_1$ with

```
(Inside - {subject (Bug !B-1)}
          {object (Container 'PetriDish)})
```

in its data base, and we would try to show that

```
(Dead - {object (Bug !B-1)})
```

is provable in $\mathcal{T}_1$. Here, !B-1 is a newly created symbol that is interpreted as a constant in every unification step inside $\mathcal{T}_1$. How can we prove that !B-1 is dead? Suppose we have some additional Horn clauses in our rule base stating that "bugs are killed by heating," and that "anything inside a container is heated whenever the container is heated." Suppose we have also been told that 'PetriDish-1 has been heated. Then the Horn-clause proof would succeed in $\mathcal{T}_1$, and the proof that 'PetriDish-1 is a sterile container would succeed in $\mathcal{T}_0$.

The proofs for the negation rules are similar. To prove that 'Blackacre is "unowned property," we would construct an auxiliary tableau $\mathcal{T}_1$ with

```
(Own - {subject (Actor !A-1)}
       {object (Property 'Blackacre)})
```

in its data base, and we would try to show that FALSE is provable in $\mathcal{T}_1$. For this proof to succeed, of course, we would have to find some rule in our rule base that makes an explicit negative assertion. For example, if we are told authoritatively that 'Blackacre is "unregistered property," and if we know that "registration" is a necessary condition for "ownership," then the proof in $\mathcal{T}_1$ would succeed. Note the overall strategy of this proof: We *assume* that some actor !A-1 owns 'Blackacre and we show that this assumption leads to a contradiction.

For a more detailed discussion of this proof procedure, including a soundness and completeness theorem, the reader should consult [26]. Further results appear in [7, 8].

3.3 Default Rules and Default Proofs

The use of *default rules* in legal reasoning has been analyzed by Gordon [13, 14]. In general, whenever we see the words "unless" or "except" in a statute [2], a proper representation of the rule requires the use of some form of default reasoning. In *LLD*, this facility is provided by combining a *failure* operation with intuitionistic negation in a particular way. For example, to say that "an adult is presumed to be employed unless shown to be a dropout" [39], we would write the following rule:

```
(Employed - {object (Person P)})

<==  (Adult - {object (Person P)})

     AND FAIL NOT NOT

     (Dropout - {object (Person P)}).
```

To use this rule, the system would try to prove `FALSE` from the assumption that `P` is *not* a `Dropout`, and if the attempt failed the rule would succeed.

This approach to default reasoning is similar to the approach of Poole [37]. When several default rules interact, however, the inferences become quite complex, and it is necessary to provide a semantics and a proof theory to clarify the intended behavior of the system. The default proofs in *LLD* are based on the theory in [28], which has several advantages over alternative approaches in the literature. First, the proof procedure in *LLD* is local rather than global, as it is in Reiter's default logic [38] and Moore's autoepistemic logic [34]. Second, the revision of a default proof in *LLD* is simple and straightforward, and the proof tree itself can serve as the principal data structure for a truth maintenance system [12]. Third, it is possible to "tune" a set of default rules very precisely in *LLD*, to block unintended contrapositive inferences and to enforce priorities among defaults. I argue in [28] that these features are essential for a practical system of default reasoning.

3.4 Prototypes and Deformations

The standard syntax for a rule in *LLD* does not allow disjunctive or existential assertions, but we can achieve a similar effect by using *prototypes and deformations*. Imagine that every rule is a definition giving sufficient conditions for the atomic formula that appears on its left-hand side, and imagine that certain rules provide necessary conditions as well. Obviously, these "if-and-only-if" definitions could be used to make disjunctive assertions, but it would then be necessary to construct arbitrary disjunctive proofs. Instead, in *LLD*, we designate a particular disjunct in the definition as *prototypical*, and we represent every other disjunct as a *transformation* of the prototype. Then, whenever we need to use the definition in a proof, we simply construct the "prototypical proof" using the "prototypical disjunct" and we update it, as needed, by applying the transformations.

For example, consider the following rule giving sufficient conditions for the concept of a "Christmas block":

```
(ChristmasBlock B)

<==   (Block B)  AND

      [(Painted -
         {object (Block B)}
         {quality (Color 'Red)})

       OR

       (Painted -
         {object (Block B)}
         {quality (Color 'Green)})].
```

This rule is equivalent to a pair of Horn clauses, and it tells us that both red blocks and green blocks are "Christmas blocks." If, in the course of a proof, we discover that a particular block `'B-1` is a "Christmas block," what would we conclude? If we assume that the rule expresses necessary as well as sufficient conditions, then we would conclude that `'B-1` is either red or green, and we would be forced to *split* the proof at this point. Instead, we will designate a red block as the prototypical example of a "Christmas block," and we will continue the proof on the assumption that `'B-1` is red. Now suppose this proof succeeds. We then have two choices: We could apply the 'red:green' transformation directly to the prototypical proof, and check to see if the transformed proof still succeeds. This would give us a sound and complete disjunctive proof procedure. Or we could terminate the computation at this point, and apply the 'red:green' transformation only if the prototypical assumption later leads to a contradiction. In this latter case, our proof procedure would be complete, but not sound, and yet it might be justifiable if we knew that an unsound conclusion could always be revised in the light of conflicting information.

A theory justifying this approach is presented in [29]. In the "Christmas block" example, the necessary conditions are given by Clark's *predicate completion* [10], as suggested above, but whenever the rules are recursive the necessary conditions are given by McCarthy's *circumscription* [20] and the "proofs" (which cannot be complete) are specified by a set of *induction schemas.* I suggest in [29] that there is a relationship between these prototypical proofs and our intuitive sense of conceptual coherence. A concept is *coherent* (in a certain context, and for a certain purpose) if its representation in terms of prototypes and deformations yields a tractable inference problem (in the specified context, and for the specified purpose). The theory is thus intended as a formalization of my earlier work with Sridharan on the use of prototypes and deformations in legal argument [31, 32, 44].

A further example of the use of prototypes and deformations, this time in a modal context, is outlined in Section 4.3.

4 Modalities

The rules in Section 3 constitute a (somewhat unconventional) first-order language, but most of our common sense categories involve modal concepts: time, action, permission, obligation, causation, purpose, intention, knowledge, belief, and so on. In this section, I will discuss the modalities that have been incorporated into *LLD* so far: time (Section 4.1); events and actions (Section 4.2); and permissions and obligations (Section 4.3).

The foundation for this treatment of modalities is my earlier work on deontic logic [23, 24]. But the present work goes further in two respects. First, the action language is based on an intuitionistic semantics, so that it possesses the properties discussed in Section 3.2. In particular, it has unique minimal models and definite answer substitutions. Second, the rules from Section 3 are used here as well, to define abstract actions in terms of more concrete actions. This means that the proof procedure for the first-order language can be generalized to cover both the action language and the deontic language, thus simplifying the overall system.

4.1 Time

In the terminology of Shoham [42], the temporal component of *LLD* is based on a *reified temporal logic*. For example, we can assert that the `Control` relationship holds between two corporations at a particular time:

```
(State -
  {relation
    (Control -
      {subject (Corporation A1)}
      {object (Corporation A2)})}
  {time (Time T)}),
```

and we can make a similar assertion about a time interval:

```
(State -
  {relation
    (Control -
      {subject (Corporation A1)}
      {object (Corporation A2)})}
  {time1 (Time T1)}
  {time2 (Time T2)}).
```

These conventions then generalize easily to the language of events and actions.

4.2 Events and Actions

The action language in *LLD* is exactly the same as the action language in [23, 24], except that it is based on intuitionistic logic. I argued in [24] that an action language should be defined on *partial models*, but this becomes cumbersome when the semantics for partial models is classical. However, as I showed in [25], a more natural semantics for partial models is intuitionistic. This choice has several advantages: Most important is the fact that a change in the state of the world can now be defined by a pair of *unique minimal partial models.*

With this modification, the representation of events and actions in *LLD* follows [23, 24] very closely. Elementary events are represented by *statechanges*:

```
(StateChange -
  {relation1
    (Own O1
      {subject (Actor A1)}
      {object (Property P)})}
  {relation2
    (Own O1
      {subject (Actor A2)}
      {object (Property P)})}
  {time1 (Time T1)}
  {time2 (Time T2)}),
```

and complex events are constructed by the operations of disjunction, sequential and parallel composition, and universal and existential quantification applied to the elementary statechanges. An *action* is a relationship between an actor and an event, which may be either elementary or complex. In addition, the rule syntax of *LLD* can be used to define various abstract events and actions. For example, in Figure 1, `TransferProperty` is defined by an elementary event, `DistributeProperty` is defined by a complex event, and both of these actions are used in the definition of `DistributeDividend`. Finally, by adding to the action language an analogue of the default rules in Section 3.3, we obtain a solution to the "frame problem" [15]. The resulting language is similar in its effects to the *event calculus* of Kowalski and Sergot [18].

4.3 Permissions and Obligations

Since I have described the logic of permissions and obligations in my earlier papers [23, 24], I will abbreviate the discussion here. In *LLD* syntax, an *obligation* is written as follows:

```
(Obligation -
  {condition
    (Issued -
      {subject (Corporation C)}
      {object (CommonStock S)})}
  {action
    (DistributeDividend -
      {agent (Corporation C)}
      {object (Cash &P
                {subjectof
                  (Value -
                    {unit (Dollar -)}
                    {quantity (Number N)})})}
      {recipient (Actor &A
                   {subjectof
                     (Own -
                       {object
                         (CommonStock S)})})}
      {time1 (Time T1)}
      {time2 (Time T2)})}
  {time (Time T1)}),
```

where the obligatory action in this example is the same DistributeDividend action that was defined in Figure 1. Although most systems of deontic logic would attempt to prove the validity of any well-formed formula in the language, I have argued in my earlier papers that this objective is both unnecessary and impractical in a legal reasoning system. Instead, *LLD* allows a user to construct a rule base of deontic rules in the form shown above, and then to query whether, under a particular condition ϕ, a particular action α is permitted, forbidden, obligatory, etc. Restricted to these PROLOG-like inferences, I claim, the proof procedure for the deontic modalities becomes tractable.

I will illustrate this point with a simple example from [24], but augmented here by the theory of prototypes and deformations suggested in Section 3.4. Suppose we have two deontic rules in our rule base:

> If C is a corporation, then C is obligated to transfer some security to 'Jones.
>
> If any actor A owns a bond, then A is forbidden to transfer that bond to 'Jones.

Assume also that 'Dupont is a corporation, and that we want to know whether 'Dupont is obligated to transfer a stock to anyone, and, if so, to whom. (Intuitively, the answer is: Yes, DuPont is obligated to transfer a stock to 'Jones.) I outlined a proof strategy in [24]: We assume that there is no such obligation and we try to derive a contradiction, returning an answer substitution if possible. At one point in the proof, however, we arrive at the assumption that there exists a partial world w in which the action TransferSecurity is true but the action TransferStock is not true. Since TransferStock is defined by a set of sufficient conditions in the action language, this part of the proof is straightforward. But in order to use our assumption about the action TransferSecurity, we need to adopt the position, from Section 3.4, that the sufficient conditions in the definition of TransferSecurity are necessary conditions as well, and this leads to the construction of a prototypical proof. In this example, as the discussion in [24] indicates, the prototypical proof simply contains a skolem function for the unidentified security, and there are no transformations. In other examples, however, the abstract actions might have disjunctive definitions, and the transformations would become important. In general, since a deontic proof can use the definition of an action in either a "forward" or a "backward" direction, we cannot avoid the use of both necessary and sufficient conditions.

Deductive inference in a modal logic is notoriously difficult [47, 36]. But prototypical proofs are relatively simple. If this example can be generalized, as I believe it can, it suggests a plausible account of human common sense reasoning in complex modal contexts.

5 An Integrated Language

Most of the components of a *Language for Legal Discourse* have appeared elsewhere, in other knowledge representation languages, but it remains a substantial challenge to combine these several components into a single system. I have tried to identify in this paper some of the features of the language that contribute to integration: (1) the fact that every relationship, and every event, action, obligation, etc., is treated as an individual object; (2) the fact that the semantics is intuitionistic throughout, so that each component of the language is defined on partial models; (3) the use at each language level of a standard rule syntax that generalizes Horn-clause logic; and (4) the use of a tableau proof procedure that can be extended from the first-order case to the various modalities. In future papers, I will explore the applications of *LLD* in several legal domains, and I will try to evaluate how successful this attempt at integration has been.

References

[1] L.E. Allen. Symbolic logic: A razor-edged tool r drafting and interpreting legal documents. *Yale Law Journal*, 66:833–879, 1957.

[2] L.E. Allen and C.S. Saxon. Some problems in designing expert systems to aid legal reasoning. In *Proceedings of the First International Conference on Artificial Intelligence and Law*, pages 94–103. ACM Press, May 1987.

[3] H. Aït-Kaci and R. Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[4] K.R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.

[5] J. Barwise and J. Perry. *Situations and Attitudes*. Bradford Books, MIT Press, 1983.

[6] A.J. Bonner. A PROLOG framework for reasoning about permissions and obligations, with applications to contract law. Unpublished seminar paper, Rutgers University, 1985.

[7] A.J. Bonner. Hypothetical datalog: Complexity and expressibility. In *Proceedings of the Second International Conference on Database Theory*, pages 144–160. Springer-Verlag, 1988. Lecture Notes in Computer Science, volume 326.

[8] A.J. Bonner. A logic for hypothetical reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 480–484. AAAI, 1988.

[9] H.C. Bunt. *Mass Terms and Model-Theoretic Semantics*. Cambridge University Press, 1985.

[10] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum, 1978.

[11] A.G. Cohn. A more expressive formulation of many sorted logic. *Journal of Automated Reasoning*, 3:113–200, 1987.

[12] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[13] T.F. Gordon. OBLOG-2: A hybrid knowledge representation system for defeasible reasoning. In *Proceedings of the First International Conference on Artificial Intelligence and Law*, pages 231–239. ACM Press, May 1987.

[14] T.F. Gordon. The importance of nonmonotonicity for legal reasoning. In H. Fiedler, F. Haft, and R. Traunmüller, editors, *Expert Systems in Law: Impacts on Legal Theory and Computer Law*, pages 111–126. Attempto-Verlag, Tübingen, 1988.

[15] P. Hayes. The frame problem and related problems in artificial intelligence. In A. Elithorn and D. Jones, editors, *Artificial and Human Thinking*, pages 45–59. Jossey-Bass, 1973.

[16] R. Jackendoff. *Semantics and Cognition*. MIT Press, 1983.

[17] R. Kowalski. *Logic for Problem Solving*. North Holland, 1979.

[18] R.A. Kowalski and M.J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[19] J. McCarthy. First order theories of individual concepts and propositions. In J. Hayes, D. Michie, and L. Mikulich, editors, *Machine Intelligence*, volume 9, pages 129–147. Ellis Horwood, 1979.

[20] J. McCarthy. Circumscription: A form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[21] L.T. McCarty. Reflections on TAXMAN: An experiment in artificial intelligence and legal reasoning. *Harvard Law Review*, 90:837–93, 1977.

[22] L.T. McCarty. Intelligent legal information systems: Problems and prospects. *Rutgers Computer and Technology Law Journal*, 9(2):265–294, 1983. Also published in C. Campbell, editor, *Data Processing and the Law*, pp. 125-151 (Sweet and Maxwell, 1984).

[23] L.T. McCarty. Permissions and obligations. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 287–294, 1983.

[24] L.T. McCarty. Permissions and obligations: An informal introduction. In A.A. Martino and F. Socci Natali, editors, *Automated Analysis of Legal Texts: Logic, Informatics, Law*, pages 307–337. Elsevier North-Holland, 1986. Also available as Rutgers Technical Report LRP-TR-19.

[25] L.T. McCarty. Clausal intuitionistic logic. I. Fixed-point semantics. *Journal of Logic Programming*, 5(1):1–31, 1988.

[26] L.T. McCarty. Clausal intuitionistic logic. II. Tableau proof procedures. *Journal of Logic Programming*, 5(2):93–132, 1988.

[27] L.T. McCarty. Intelligent legal information systems: An update. In H. Fiedler, F. Haft, and R. Traunmüller, editors, *Expert Systems in Law: Impacts on Legal Theory and Computer Law*, pages 15–25. Attempto-Verlag, Tübingen, 1988. Also published in *Law and Computers*, No. 5, pp. 196-202 (Law and Computers Association of Japan, July 1987).

[28] L.T. McCarty. Programming directly in a nonmonotonic logic. Technical Report LRP-TR-21, Computer Science Department, Rutgers University, September 1988.

[29] L.T. McCarty. Computing with prototypes (preliminary report). Technical Report LRP-TR-22, Computer Science Department, Rutgers University, March 1989.

[30] L.T. McCarty. Real relationships. Forthcoming, 1989.

[31] L.T. McCarty and N.S. Sridharan. A computational theory of legal argument. Technical Report LRP-TR-13, Computer Science Department, Rutgers University, 1981.

[32] L.T. McCarty and N.S. Sridharan. The representation of an evolving system of legal concepts: II. Prototypes and deformations. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 246-53, 1981.

[33] D.L. McGuinness. Reasoning with permissions and obligations in contract law. Unpublished seminar paper, Rutgers University, 1986.

[34] R.C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[35] D. Moshier and W. Rounds. A logic for partially specified data structures. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 156–167, 1987.

[36] H.J. Ohlbach. A resolution calculus for modal logics. In *Proceedings, Ninth International Conference on Automated Deduction*, pages 500–516, 1988.

[37] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[38] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[39] R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 270–276, 1981.

[40] D.A. Schlobohm and L.T. McCarty. EPS II: Estate planning with prototypes. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*. ACM Press, June 1989.

[41] M.J. Sergot, F. Sadri, R.A. Kowalski, F. Kriwaczek, P. Hammond, and H.T. Cory. The British Nationality Act as a logic program. *Communications of the ACM*, 29:370–386, 1986.

[42] Y. Shoham. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33:89–104, 1987.

[43] N.S. Sridharan. Representing knowledge in AIMDS. *Informatica e Diritto*, 7:201–221, 1981.

[44] N.S. Sridharan. Evolving systems of knowledge. *AI Magazine*, 6:108–120, 1985.

[45] R.E. Susskind. *Expert Systems in Law: A Jurisprudential Inquiry*. Oxford University Press, 1987.

[46] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[47] L.A. Wallen. Matrix proof methods for modal logics. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 917–923, 1987.

```
(DistributeDividend -
  {agent (Corporation C)}
  {object (Cash &P1
           {subjectof (Value - {unit (Dollar -)}
                               {quantity (Number N)})})}
  {recipient (Actor &A
              {subjectof (Own -
                          {object (Stock &S2
                                   {objectof (Issued -
                                              {subject (Corporation C)})})})})}
  {time1 (Time T1)}
  {time2 (Time T2)})

<==    FOR ALL (Stock S2) (Number N2):

       (Issued -
         {subject (Corporation C)}
         {object (Stock S2
                  {subjectof (Amount - {unit (Share -)}
                                       {quantity (Number N2)})})}
         {time (Time T1)})

        ==>

        [(TransferProperty -
           {agent (Corporation C)}
           {object (Cash &P1
                    {subjectof (Value - {unit (Dollar -)}
                                        {quantity (Number N)})})}
           {time1 (Time T1)}
           {time2 (Time T2)})

         AND

         (DistributeProperty -
           {agent (Corporation C)}
           {object (Cash &P1
                    {subjectof (Value - {unit (Dollar -)}
                                        {quantity (Number N)})})}
           {recipient (Actor &A
                       {subjectof (Own -
                                   {object (Stock &S2
                                            {objectof (Issued -
                                                       {subject (Corporation C)})})})})}
           {measure (Stock &S2
                     {subjectof (Amount - {unit (Share -)}
                                          {quantity (Number N2)})})}
           {time1 (Time T1)}
           {time2 (Time T2)})].
```

Figure 1: "Corporation C distributes N dollars in cash to the owners of the stock issued by corporation C."