Applications of Deontic Logic in Computer Science: A Concise Overview

R.J. Wieringa J.-J.Ch. Meyer*

Vrije Universiteit
Faculty of Mathematics and Computer Science
De Boelelaan 1081 a
1081 HV Amsterdam
The Netherlands
e-mail: roelw@cs.vu.nl, jules.cs.vu.nl

Abstract

Deontic logic is the logic that deals with actual as well as ideal behavior of systems. In this paper, we survey a number of applications of deontic logic in computer science that have arisen in the eighties, and give a systematic framework in which these applications can be classified. Many applications move in the direction of programming a computer in deontic logic to make the computer prohibit, permit or obligate people to do something. We discuss conditions under which this possibility is realistic and conditions under which it would be admissible to do so.

1 Introduction

Deontic logic is the logic to reason about ideal and actual behavior. From the 50's, Von Wright [62, 64], Castañeda [12], Alchourrón [1] and others developed deontic logic as a modal logic with operators for permission, obligation, and prohibition. Other operators are possible, such as formalizations of the system of concepts introduced by Hohfeld in 1913, containing operators for duty, right, power, liability etc. [20].

Deontic logic has traditionally been used to analyze the structure of normative law and normative reasoning in law. It is therefore only natural that interest in the application of deontic logic in computer science started in the area of legal applications. The *International Conference on Logic, Informatics, Law* [14, 34, 36, 35] has been held every four years since 1982, and its proceedings contain a large number of papers on the application of deontic logic to the automization of legal automation. More recently, the *International Conference on Artificial Intelligence and Law* [21, 22, 23],

*Also at the University of Nijmegen, Nijmegen, The Netherlands

held biannually since 1987, is starting to publish a number of papers on the applications of deontic logic to the problems of artificial intelligence in law.

Only recently, it has been realized that deontic logic can be of use outside the area of legal analysis and legal automation. Deontic logic has a potential use in any area where we want to reason about ideal as well as actual behavior of systems. Examples of computer science applications that we will review below include the specification of fault-tolerant systems, the specification of security policies, the automation of contracting, and the specification of normative integrity constraints for databases.

We start with a brief survey the applications of deontic logic to computer science in roughly chronological order (section 2). Next, we take a systematic viewpoint in section 3 and try to classify all possible applications into a few simple categories. This allows us to bring some system in the research already done and to identify possible new areas of application. Finally, we discuss some limits to the application of deontic logic that stem from its peculiar nature as a medium to *prescribe* human behavior.

2 A chronological survey of applications

2.1 Legal automation

By legal automation we mean the use of computers to support tasks in the legal process. In general, this may range from text processing and electronic data interchange (EDI) to information retrieval and legal advice-giving. Here, we concentrate on the last kind of task and look at the role deontic logic can play in the automation of legal advice-giving.

There are two approaches to computer support in legal advice-giving, the factual approach in which there is no distinction between actuality and ideality, and the deontic one in which there is such a distinction. We briefly discuss this difference and then turn to some examples of deontic approaches.

2.1.1 The factual and deontic approaches to legal automation

In 1957, Layman Allen already pointed out that formal logic can be used to identify ambiguities in legislation and to draw logical consequences from legal rules [3]. This in turn can help the legislator to eliminate unwanted ambiguities and to clarify and simplify the text of laws. Allen illustrated this in two papers published in the early 80's [4, 5], using essentially first-order logic without deontic operators. His approach has actually be used in a legislative process in Tennessee [18].

Representation of a part of the law in logic was also done in 1985 by the logic programming group at Imperial College, which implemented part of the British Nationality Act of 1981 as a Prolog program [54]. In this approach to formalization of law, legislation is viewed as a set of *definitions* rather than as a set of obligations, permissions and prohibitions issued by authorities. Thus, the concept of British citizenship as defined in the 1981 U.K. law is formalized by a series of rules in Prolog. Like Allen's, this is an example of the factual approach to formalizing law.

Other examples of the factual approach are legal expert systems, which as a rule do not represent the logic of the difference between ideal and actual situations. For example, the TAXMAN [38] system developed by Thorne McCarty represents the legal concepts pertaining to U.S. tax laws that apply to corporate reorganizations. Basically, the system formalizes the definitions of these concepts and applies them to the facts of a reorganization to see whether the reorganization can be classified as a tax-free transaction.

Other examples of fact-based approaches in legal automation abound. Marek Sergot [53] gives a comprehensive survey of approaches to the representation of law as computer programs, including factual and deontic ones. In another survey paper [52], he concentrates on the factual approach, but discusses philosophical issues involved in any approach to formalizing and automating legal advice-giving, including the deontic approach. We take up some of these issues below in section 4.

As remarked by Andrew Jones [24], there is nothing wrong with the factual approach as long as all one wants is find out how the definitions of concepts given in the law apply to the case at hand. A system built along these lines can help one in making clear what the law says or implies (according to the interpretation used in formalizing the text of the law). But this approach is characterized by not being able to consistently express *violations* of these definitions and the ability to do just that is the hallmark of deontic logic. The need for such a logic in the representation of law as logic programs was already pointed out by Sergot in 1982 [51]. Jones gives an example of rather simply normative statements, whose formalization involves some of the deepest problems of deontic logic and which cannot be avoided if we want to formalize these statements. Briefly, Jones shows that the Imperial College library regulations includes the following rules, where p is a person and d is a document that can be borrowed by p.

1. p shall return d by date due.
2. If p returns d by date due then disciplinary action is not taken against p.
3. If p does not return d by date due then disciplinary action shall be taken against p.

If we now encounter a case where the following is true,

4. p does not return d by date due,

then we have an example of what deontic logicians call *Chisholm's paradox* or the paradox of contrary-to-duty imperatives [13]. Roughly, the paradox is that a reasonable formalization of these sentences implies that there is simultaneously an obligation to take disciplinary action against p and an obligation not to take disciplinary action against p. Avoiding this paradox involves either a reduction of the example to triviality (for example because the truth of sentence 4 makes the truth of sentence 2 trivial) or else the resolution of some of the deepest problems of philosophical logic, such as the formalization of counterfactual conditionals. Tomberlin [57] gives a critical survey of some of the issues involved and in a recent paper, Jones and Pörn [25] propose a resolution of the paradox using a refined concept of obligation. Meyer [43] gives a

possible solution using dynamic logic. The matter is far from resolved, however, and research in this issue continues.

More arguments for the use of deontic logic in the specification of normative system behavior are analyzed in the contribution of Jones and Sergot to this volume [26].

2.1.2 Examples of deontic approaches

An early example of the deontic approach is LEGOL project led by Ronald Stamper from about the middle of the 70's, then at Imperial College. The goal of the LEGOL project was the development of conceptual modeling methods for information system development in organizations that results in more accurate models of reality. The "classical" approach to formal modeling, based on denotational semantics, was deemed inappropriate for this goal, and a different approach, based on actions and social norms, was developed. The representation of legislation in computers is one of the application areas of LEGOL and this is the reason why LEGOL can be classified as a legal automation project. The project resulted in a language to specify information system models that looks like relational algebra extended with operators to handle time [27]. One of the extensions of this language contains operators that deal with deontic concepts like right, duty, privilege and liability [55]. This idea of applying deontic logic was not followed up, however. Stamper has now moved to the University of Twente, where a different project was started, although still with the goal to find an appropriate semantics for the modeling of business processes.

Another early example of the use of deontic logic for the computer representation of legal reasoning is research that followed the TAXMAN project mentioned above. The original, factual approach in the TAXMAN project ran against some limitations, one of which was the limitation that differences between ideal and actual states of the world cannot be represented in the factual approach. Still, this difference is needed in the computer representation of corporate tax law. For example, the differences between some kinds of stocks and bonds can only be characterized by giving the rules of permission and obligations which are binding on the corporation and its securityholders [40]. McCarty therefore extended his approach to incorporate deontic reasoning in law, and did this by developing a version of dyadic deontic logic and reported on this in papers published in 1983 and 1986 [40, 41]. The language containing his deontic operators contains constructs to specify actions as well as deontic operators. It is part of a more expressive language containing in addition constructs for specifying sorts and subsorts, events, and time, called a Language for Legal Discourse (LLD) [42]. In LLD, one can specify the rule that any corporation that owns cash has an obligation to distribute cash to all its stockholders in a Lisp-like syntax as follows [41, page 323]:

```
(obligate ?
        (own ? (corporation ?X) (cash ?Y))
        (distribute-dividend ?) (corporation ?X)))
```

The question marks denote anonymous variables, ?X and ?Y are named variables.

Another example of application of deontic logic to legal automation is the ESPLEX system, described in a 1987 paper by Biagioli et al. [10]. ESPLEX deals with the rules

of agricultural tenancies in Italy as laid down in an Act from 1982. For example, one of the conditions under which the termination of tenancy is permitted can be specified in ESPLEX in a Prolog-like syntax by

```
Permitted (termination, tenant, tenancy) :-
        cond (small farmer, tenant),
        proc (termination, tenancy).
```

The prefix `cond` gives a predicate that must be satisfied, the prefix `proc` gives a legal procedure, defined elsewhere in the system, that must have been followed in order for the conclusion of the rule to be valid. Biagioli et al. do not give a logic for their system.

In 1985, Layman Allen and Charles Saxon [7] showed how to use the system of concepts defined by Hohfeld to define a formal language in which to analyze legal texts precisely in order to disambiguate them. These concepts include various nuances of the standard deontic concepts of permission, obligation and prohibition, as well as others like right, duty, privilege, power, liability and immunity. This volume contains an example of an application of their approach to the Imperial College Library Regulations [6], in which they show that 2560 different interpretations of these regulations are possible. They present a system for generating multiple interpretations, called MINT, that can help the people who issue rules (in government as well as in private organizations) to discover ambiguities and clarify the language of the rules.

This brings us to the intended use of the discussed systems. LEGOL was meant to be used for the specification of business models that contain a normative component. McCarty's aim is to study legal reasoning as actually performed by lawyers [42]. Although the intended use of ESPLEX is not explicitly stated in the paper [10], from the examples it appears that it is to be used as an expert system shell that can be used to build expert systems in various areas of law. These expert systems will presumably be used for advice-giving in the application of law to actual problems. Allen's work, finally, is not oriented on supporting the application of law to actual problems, but on the support of legislators in the designing and drafting of texts that express law, both in a public and private areas. These different uses of advice-giving systems will be discusses systematically in section 3.

2.2 Authorization mechanisms (Minsky and Lockman 1985)

Authorization is the mechanism by which actors are provided with the permission to perform certain actions. Authorization mechanisms are used in computer science, among others, to protect the integrity of resources in operating systems and databases, to guard the security of sensitive resources, and to protect the privacy of sensitive data. In 1985, N. Minsky and A. Lockman [45] convincingly argued that existing authorization mechanisms are deficient in that they lack the concept of an obligation. First of all, the concept of obligation to perform an action is relevant even independently from that of permission to perform actions. For example, the beginning of a database transaction involves locking data to prevent other users from accessing the data during the transaction. Execution of the *begin transaction* event contains an implicit obligation to end the transaction in a reasonable period of time by either a *commit transaction*

or a *roll back transaction*. Thus, if an actor performs certain events, then by this performance he incurs an obligation to do something else as well.

In the context of authorization mechanisms, Minsky and Lockman argue that granting permissions without "strings attached" is often not adequate as authorization mechanism. They give the following examples.

- Traditionally, permissions to perform certain actions, e.g. to read a file or update a database field, are granted without putting the actor under an obligation when he or she actually performs the action. However, in many cases such an obligation implicitly exists. For example, if a library grants me permission to borrow books, then by actually borrowing a book I incur the obligation to return the book. This obligation should be specified together with the specification of the permission granted to me to borrow books.

- Some constraints on a computer system may temporarily be violated, but violation will create an obligation to undo the violation. For example, suppose we want to grant permission to someone to allocate employees to jobs, with the constraint that vital jobs must never be unfilled more than five days. We then want to grant permission to allocate jobs, and simultaneously stipulate that releasing an employee from a vital job creates an obligation to fill the job before the next weekend.

- Suppose we grant someone permission to perform actions from the set A_1 and independently grant the same person permission to perform actions from the set A_2. These permissions may not be additive, i.e. there are situation where the performance of an action from one set precludes performance of an action from the other. This can be specified if we can grant permission in such a way that actual performance of an action from one set creates an obligation not to perform an action from the other.

Minsky and Lockman propose a language to express authorizations with "strings attached." For example, the employee allocation permission can be specified as

can $release(e, j)$ **where** $department(j) = D$ **and** $vital(j)$
requiring to do $appoint(e', j)$ **by** $weekend$
or else $default\_fill(j)$

This expresses the permission to release employee e from job j, where j is in department D and the job is vital, but that if this release actually takes place, an obligation is triggered that requires the appointment of an employee by the weekend. An enforcement mechanism is presupposed that monitors fulfillment of obligations. If the obligation is violated, the enforcement mechanism takes the action $default\_fill(j)$.

Minsky and Lockman propose syntactic constructs that deal with nested obligations, deadlines, triggers, and negative actions (refraining from action). They give only an informal semantics of these constructs and no logic is given.

2.3 System specification (Khosla and Maibaum 1987)

Formal specification of system behavior in the style of VDM is done by specifying pre- and postconditions of actions. In these specifications, preconditions are used to specify the *effect* of an action as well as to specify the *context* in which the action is allowed to occur. For example, database transactions are often specified with preconditions so that, if the precondition is satisfied, certain static constraints on the allowable states of the database will not be violated.

Khosla and Maibaum [29, 28] point out that this involves two different uses of preconditions, and that these uses should be separated. First of all, the precondition of an action is used to identify the context in which the action is performed, in such a way that the postcondition can define the effect of the action in a declarative way. If this would be the only role of preconditions, then if the action has no precondition, this would not mean that the action is always permitted to occur, but merely that the effect of the action does not depend upon the context in which it occurs. There is a second use of preconditions, however, which is that they state when the action is *permitted* to occur. Viewed in this way, the absence of a precondition means that the action is always permitted to occur.

Khosla and Maibaum propose to separate these two uses of preconditions by using preconditions only for the definition of the effect of an action, and using deontic logic to express the conditions under which an action is permitted to occur. By separating the specification of the permission to perform an action from the specification of the pre- and postconditions of an action, the specification of the effect of an action is simplified. Secondly, this separation allows the specification of *fault-tolerant systems*, in which for some reason bad behavior cannot be banished altogether, for example because hardware failure remains a possibility. The use of deontic logic allows the specification of which corrective action should be taken to undo or at least ameliorate the result of bad system behavior.

Khosla and Maibaum define an extension of modal action logic called Deontic Action Logic (DAL). Informally stated, in DAL every possible state of the system is labeled as either permitted or forbidden. First, in a *permitted* state of the system, actions are permitted iff they lead to permitted states of the system and they are forbidden iff they lead to forbidden states of the system. Second, it is left open in which cases actions that start from a *forbidden* state of the system are permitted or forbidden. If the system is in a forbidden state, we can specify every action to be forbidden, or permit only actions that lead to a permitted state of the system, or selectively permit some actions, even if they do not bring the system closer to a permitted state.

Using DAL, Khosla and Maibaum [29] specify a telephone system. Example axioms in this specification are:

$$[Ex, BusyInd(t)]BusyTone(t) \tag{1}$$

$$[Ex, RingInd(t)]RingTone(t) \tag{2}$$

$$[Ex, RingBell(t)]BellRinging(t) \tag{3}$$

$$Busy(t') \rightarrow [Ex, Connect(t, t')]\mathbf{O}(Ex, BusyInd(t)) \tag{4}$$

$$\neg Busy(t') \rightarrow [Ex, Connect(t, t')]\mathbf{O}(Ex, RingInd(t) \| RingBell(t')) \qquad (5)$$

The first three axioms specify the effect of three actions in a context- independent way, because there are no preconditions. (1) says that if the exchange issues a $BusyInd(t)$ action to telephone t, t will sound a tone that indicates that the called telephone (the callee) is busy, (2) says that a $RingInd(t)$ signal will make t sound a tone that indicates that the callee is ringing, and (3) says that a $RingBell(t)$ signal causes t to ring its bell. The last two axioms specify what the exchange should do when it finds the callee busy (4) and when it finds the callee available (5). Thus, (4) says when the busy tone should be sent to t. (5) says that if the callee (t') is not busy, then after the exchange performs the connect action it should indicate to the caller (t) that the callee is ringing, and simultaneously he should send a bell-ringing signal to the callee.

DAL is based upon modal action logic and contains operators like parallel composition, sequential composition and choice, to combine actions into more complex processes. This line of research is continued by José Fiadairo of INESC in collaboration with Tom Maibaum [16] in an attempt to integrate deontic specification in a wide-spectrum specification language. As discussed in our companion survey of deontic logic, they reduce deontic logic to temporal logic [44].

2.4 Electronic contracting (Lee 1986)

In 1984, Kimbrough, Lee and Ness [30] pointed out that office documents often not only have informative value but also have performative value. For example, a customer order to a supplier has *informative* value because it contains data like the name and address of the customer, identifications of the ordered goods, etc. In addition, it has *performative* value because it constitutes a question to the supplier to deliver goods, which the supplier ought to answer, as well as a promise of the customer to pay for the goods when delivered. This performative value is often represented by a signature or by other means that are intended to prevent forgery.

By the advent of office information systems, informative as well as performative documents are often stored in and manipulated by computer systems. For example, the customer and supplier may be connected by an electronic data interchange (EDI) network and the order may be sent to the supplier automatically, say every end of the month, to replenish the stock of goods of the customer. Thus, an analysis of both the informative and the performative structure of these documents is important for the development of office information systems.

However, traditional information development methods do not deal with the performative aspects of the developed information system. Data models use a subset of first-order logic to represent the structure of the data and do not explicitly represent the performative structure of the data or of the manipulations of the data. Hence there is a need for an extension of traditional methods to deal with performative aspects.

The general logic of the performative role of information systems should be based on speech act theory and could be some form of illocutionary logic [49]. However, particular kinds of performative acts could be formalized with less heavy means. Kimbrough et al. briefly consider the suitability deontic logic as a representation

mechanism for performatives related to *contracts*. For example, relevant actions in contracting whose logic must be represented are

Oblige	Some action not obligated becomes obligated
Waive	Some obligated action becomes not obligated
Permit	A forbidden action becomes permitted
Forbid	A permitted action becomes forbidden.

In 1988, Lee worked this out in a system that can monitor the sequence of activities involved in contracting [32]. One of the main issues in contracting is the meeting of deadlines, which in turn involves the representation of processes and real time. Lee uses Petri nets to give a semantics to processes, and the logic of change developed by Von Wright [63] to represent them in his logic. The logic of absolute time presented by Rescher and Urquhart [47] is used to represent deadlines. Deontic operators are added using a variety of Anderson's [8] reduction of standard deontic logic to alethic logic. This many-colored specification language is then translated into an executable Prolog-like language, to which a natural language interface is added. It allows the formal specification of contracts like

```
Jones agrees to pay \$500 to Smith by May 3, 1987.
Following that,
Smith agrees to deliver a washing machine
to Jones within 30 days.
```

The system may them be questioned as follows:

```
?- at 5-may-1987 whatif nothing.
```

to which it responds with

```
Part Jones defaults at May 4, 1987,
because Jones failed to pay $500 to Smith by May 3, 1987.
```

No formal semantics or inference system is given for the specification language. Further work on the automation of contracting over EDI networks is done by a Ph.D. student of Lee, Sandra Dewitz [15].

Also in 1988, Lee published a paper analyzing administrative organizations as deontic systems [31]. Using roughly the same specification language, he specifies rules for granting parking permits on a university campus and shows how a rule-based system can be used to find out whether actors have permissions to perform certain actions and if so, to find out how this follows from the rules.

2.5 Deontic integrity constraints (Wieringa, Meyer and Weigand 1989)

Integrity constraints for databases are formulas that should be satisfied, in some sense to be formally defined, by the states and state transitions of a database. For example,

a static constraint is that the age of a person, measured in years, cannot be negative. Examples of dynamic constraints are that a salary should never decrease and that no one can be fired before he is hired. In a paper published in 1989, we argue that there is a crucial distinction between two types of constraints, which we call *necessary* constraints and *deontic* constraints, respectively [59]. The difference can be explained if we realize that each database represents a part of the real world by storing data about that part. *Necessary constraints* are formulas which are true of the real world because they cannot be violated by the real world. The constraint that the age of a person cannot be negative is an example of this kind. It cannot be violated by the real world because it is an *analytic truth* that follows from the meaning of the words used in expressing the constraint. Similarly, the constraint that no one can be fired before he is hired is an analytical truth that does not constrain real-world behavior at all. Because these are analytic truths, they do not constrain on the the states or behavior of the real world (given the current use of our language), and precisely because of that, they can be used as a *constraint* on the possible states and behavior of the database. A negative age field in a database cannot be a correct representation of reality, as is a record in a historical database of fire event that is not preceded by a hire event. A database in such a state must therefore be wrong.

For the purpose of database modeling, the class of necessary constraints can be extended to empirical truths that are not analytical truths but that nevertheless can be regarded as true in all possible states of the world in which we are interested. For example, the constraint that the age of a person, measured in years, cannot be larger than 150, is true in all states of the world we are interested in. It is *empirically* true, however, which means that we are convinced of its truth on the basis of our experience and that we can, in principle, find a state of the world in which it is falsified (without that being a result of a change in our use of our language). For the purpose of data modeling we can regard this constraint as a necessary truth, however, true in all states of the world that will ever be represented by the database, and therefore we can give it the same treatment as purely analytical truths. We can therefore also use it as a *constraint* on the possible states of the database and regard every database state in which a person is represented to have an age over 150 to be incorrect. Clearly, we can do this only if we put the border where the empirical truth may cease to be true at a safe distance away. The empirical truth of statement that a person cannot have an age over 100 is too uncertain to be used as a constraint on the possible states of the database, for example. In The Netherlands, at least, such a constraint is known to have bogged down the entire database system of a large insurance company, because one client happened to reach the happy age of 101.

We argue in [59] that many, if not most of the examples of database constraints published in the literature are not necessary truths in the sense explained above but are instead normative statements that apply to the real world and that can be violated in the real world. An example is the constraint given above that a salary should not decrease (a favorite example of many database researchers). When such a constraint is violated by the real world, this is not because we change the meaning of words, nor is it a falsification of an empirical generalization, but it is a violation of a real-world norm. Thus, it is truly a constraint on the real world and not on a database of facts

about the real world. A database representing data about this part of the world should be able to represent violations of this norm. In particular, it should be able to represent this not merely as just another fact which happens to be true, but *as a violation of a norm*. Otherwise, the difference between raising and decreasing a salary would not be represented, and this difference is a fact about the world in which we are interested in many applications. This leads us to deontic logic as the appropriate logic to specify database integrity constraints.[1]

As explained in our companion survey of deontic logic [44], we use an Anderson-like reduction of deontic logic to dynamic logic. This similar to Khosla and Maibaum's reduction to action logic in DAL. Thus, we label every state of the world as either forbidden or permitted. One difference with DAL is that in our logic, every action that leads to a forbidden state of the world is forbidden and every action that leads to a permitted state of the world, is permitted. Another difference is that we represent the reason for violation in the violation predicate, which provides us information to specify the appropriate corrective action and to give informative error messages. Furthermore, we make heavy use of the concept of action negation to define the relation between the three modal operators *permission*, *prohibition* and *obligation*. Other differences concern the use of propositional negation (which is used to enforce deterministic processes) and the kind of semantic structure we define for specifications. In a later paper [61], we study the problem of the inheritance of deontic constraints in a taxonomic network. Current research concentrates on the concepts of actors, initiative and action negation [60].

Using our specification language, we can specify constraints like

$$[borrow(p,b)]\mathbf{O}(return(p,b)_{(\leq 21d)}) \tag{6}$$

$$V : return(p,b) \rightarrow \mathbf{O}(pay(p,\$2,b) \tag{7}$$

$$[return(p,b)]\neg V : return(p,b) \tag{8}$$

(6) says that after p borrows a book b, an obligation exists to return the book within 21 days. $return(p,b)_{(\leq 21d)}$ is a choice between returning the book within 0, 1, ... 21 days, and $\mathbf{O}(return(p,b)_{(\leq 21d)})$ says that this choice is obligated. $V : return(p,b)$ is a *violation predicate* with two arguments, p and b. There is such a predicate for every action. It becomes true when p does not return b in time and when it becomes true, we say that a violation flag is raised. (7) says that whenever the violation flag $V : return(p,b)$ is raised, there is an obligation on p to pay two dollars. (8) says that returning the book lowers the violation flag.

[1] Deontic integrity constraints should really be called real-world constraints, because they constrain the real world and not the database. Necessary database constraints should simply be called database constraints, because they constrain the database and not the real world. However, the term "integrity constraint" is entrenched in the language of the database community to such an extent, that we continue using it and prefix it with the adjectives "deontic" or "necessary," depending on whether we mean constraints on the real world or constraints on the database.

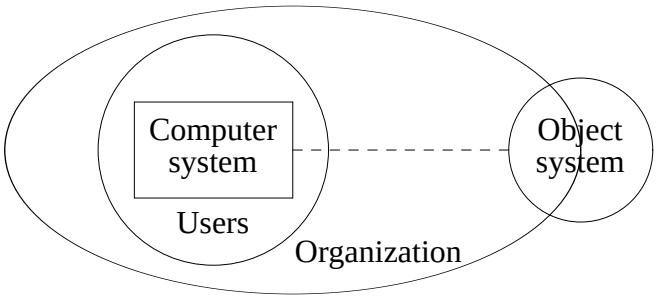

Figure 1: The structure of computer applications.

2.6 Database security (Glasgow, MacEwen and Panangaden 1989)

Glasgow, MacEwen and Panangaden [17] use deontic logic to analyze security policies for databases. They use epistemic logic with positive and negative introspection and combine this with a version of deontic logic in which $P_i(\phi)$ means "user i is permitted to know that ϕ." The following axioms are given for P_i, where $K_i(\phi)$ means "user i knows that ϕ."

$$P_i(\phi \to \psi) \to (P_i(\phi) \to P_i(\psi)) \tag{9}$$

$$P_i(\phi) \to P_i K_i(\phi) \tag{10}$$

$$K_i(\phi) \to P_i(\phi) \tag{11}$$

(9) is the usual axiom that must hold for any Kripke structure. (10) is defines permission as permission to know. Together with the K_i axioms given by Glasgow et al., it allows the derivation of $P_i(\phi) \leftrightarrow P_i K_i(\phi)$. (11) is called the security axiom, and says that a user only knows what he or she is permitted to know. Since any provable formula ϕ is known (i.e. $\phi \vdash K_i(\phi)$), (11) implies that any provable formula is permitted to be known (i.e. (i.e. $\phi \vdash P_i(\phi)$).

The logic and semantics of their system is sketched only briefly. They specify an information flow model of security by defining a relation $dominates$ defined on users, together with the axiom

$$P_i(\phi) \wedge dominates(j, i) \to P_j(\phi).$$

Glasgow note that to show that a policy for a particular $dominates$ relation is secure, a model of the axioms should be exhibited.

3 A systematic view of applications in computer science

The applications of deontic logic to computer science reviewed above, and other possible applications, can be put into simple systematic framework if we look at the structure of any computer application, shown in figure 1. Every computer system is a system that can store and manipulate data. The computer system could be a database, an expert system, a knowledge-based system, a decision support system, an operating system, etc. The kind of computer applications we are interested in contain data that

represent a part of reality. The represented part of reality is called the *object system* of the application. Users of the computer system supply it with data, put requests to it, and read the answers to the requests. Users and computer system jointly are a subsystem of an organization. The object system may fall partly inside and partly outside the organization. (It may even be or contain part of the computer system.)

Using this general structure, we may classify the applications of deontic logic to computer science by looking at the domain whose behavior is specified in deontic logic. In each case, the behavior specified is actual as well as ideal behavior. This leads us to the following classification of applications.

1. Fault-tolerant computer systems.
2. Normative user behavior.
3. Normative behavior in or of the organization.
   (a) Policy specification.
   (b) Normative organization behavior (e.g. contracting).
4. Normative behavior of the object system.
   (a) The specification of law.
   (b) The specification of legal thinking.
   (c) The specification of normative rules as deontic integrity constraints.
   (d) Other applications, not discussed above. One possible application discussed in more detail below is the application to scheduling problems.

More in detail, the list of possible applications is as follows.

1. **Fault-tolerant computer systems**. No computer system is fail-safe and there are cases where we want to specify what should happen in the case of violations of normal computer behavior, such as when parts of the hardware break down. *Fault-tolerant* computer systems can engage in behavior that is not ideal but still behave meaningfully, given the circumstances. This application of deontic logic may be considered as the specification of exception-handling, where the exceptions are generated by the computer system. The approach of Khosla and Maibaum is an example of this kind of application. Some examples given by Minsky and Lockman, concerning constraints on computer behavior that may temporarily be violated, also fall under this heading.

2. **Normative user behavior**. Users behave in all sorts of ways, many of which are not ideal. They may press the wrong keys, provide data inconsistent with data already stored in the system, refuse to provide data they are asked by the system to provide, or ask questions they are not supposed to ask, etc. Independently from whether this behavior is detectable by the computer system (some of it is and some of it isn't), we may want to make a clear distinction between the

behavior that is desired from the user and the behavior he or she may actually engage in. This would make the specification of desired user behavior simpler and it would allow us to specify what should happen in case the use behaves not in an ideal way. This application of deontic logic may be considered as the specification of error-handling, where these errors are generated by users. It is not dealt with in any of the example applications treated above.

3. Application of deontic logic to organizations can be divided into two groups, applications to the behavior of employees *in* an organization and application to the behavior *of* an organization.

   (a) **Policy specification**. Taking the whole of the organization into account, we can use deontic logic to prescribe behavior of organization components, such as employees or departments. This is done in organizational policies, which are guidelines for behavior that are meant to be followed up by whomever they are directed at. In many cases, policy designers are interested in what would happen if their policies are *not* followed up, and this makes deontic logic relevant. Deontic logic can be used, for example, to make the policies unambiguous and to explore the consequences of different specifications of the policies. This application is similar to applications in the legal domain, except that policies are not laws because they are issued by private organizations.

   Application of deontic logic to the specification of **security policies** is a special case of policy specification. This does not mean that deontic logic is used to *program* secure computer systems. A property of secure computer systems is that they do not allow violations of security norms. However, deontic logic can be used to formulate the security policies, explore their consequences, and possibly also to prove that a particular computer program actually implements the policy, i.e. that it does not allow behavior deemed undesirable under the policy. Some of Minsky and Lockman's examples fall under this class of applications, as does the paper by Glasgow et al.

   (b) **Normative organization behavior**. We may also prescribe the behavior of the organization in its environment using deontic logic. The analysis of contracting given by Lee is an example of this. Because the behavior of an organization should comply with the law, this can be viewed as a special kind of legal application of deontic logic.

4. **Specifying the behavior of the object system in deontic logic**. The object system is part of reality, data about which are to be represented in the computer system. This can be anything from a library to an elevator system, so this is really a kind of waste-basket class into which all applications fall that we haven't mentioned yet. We have seen the following applications in the review above.

   (a) **Specification of law in deontic logic**. The law is a vast area of application of deontic logic, with or without the help of computers. In terms of figure 1, the object system in this kind of application consists of people and a system

of laws that regulates their behavior. Facts and laws are formalized and manipulated according to the rules of some deontic logic to yield conclusions. Such a system can be used as an as a tutoring system for law students, or as an advice-giver in the process of legislation, or as an advice-giver in the process of applying legislation. Allen's work is an example of the second kind of application, the intended use of ESPLEX is an example of the second kind of use. Sergot [53] gives examples of all these kinds of uses. A fourth kind of application, the use of the system to give binding advice, is very controversial and will be discussed in section 4. In that section we will also discuss problems with the simple view of law application just given and which therefore plague the more modest advice-giving use of deontic logic in legal automation as well.

(b) **Simulating legal thinking in deontic logic**. McCarty's approach to legal automation is quite different from the previous one, because it aims at the simulation of the process by which lawyers and judges actually think. The object system formalized in this case is not a system of laws and people, but a psychological process taking place in the heads of some specialized people. Because this psychological process happens to concern itself with norms and facts, deontic logic may be useful. However, the representation of the object system in the computer system in this case can in no way be called a representation of law. Rather, it is a representation of the way lawyers and judges have come to think about the law. In this application, deontic logic is not used to prescribe behavior in the object system (in this case, the behavior of thought processes), but as a medium in which to express empirical hypotheses about how this behavior (thinking about normative systems) takes place.

(c) **Specifying deontic integrity constraints**. The specification of deontic integrity constraints is a simple case of formalizing rules that apply to people in the object system. Our own application falls under this heading, as do some of the examples given by Minsky and Lockman.

(d) Other applications. One other possible application, not mentioned before, is the application to **scheduling problems**. In these problems, tasks are to be allocated to resources under some constraints. For example, many databases must run processes periodically, say every week, subject to some constraints that force a sequential order on the performance of the processes, and subject to temporal constraints that put a deadline to the real time that a process is actually run. Computers being finite machines, these constraints may not always be met, and we want to specify what should happen when a constraint is violated. The allocation of jobs to machines, of airplanes to airport bays, of travelers to airplane seats, of police to the handling of incidents, and in general of widgets to fidgets under some normative scheduling constraints which in some actual situations may be violated, are interesting areas of application.

One final remark about the above classification is in order. All applications of

deontic logic involve some sort of specification of norms and actual behavior in deontic logic. None of the listed applications, however, need involve implementing the deontic logic specification in executable form in a computer. Although this suggestion is particularly strong in the case of fault-tolerant systems and legal automation, strictly speaking it is not required even here. A deontic specification of the behavior of systems in these examples can be buseful without being implemented in a computer. We therefore distinguish *specifying* the behavior of a system in deontic logic from *programming* a computer in deontic logic. If we specify a system in an executable version of deontic logic, then these two cases coincide. In general, however, this need not be the case and it is useful to keep these two cases apart.

4 Discussion: directing human behavior by computer

Most of the applications of deontic logic in computer science involve the prescription of human behavior in deontic logic. The specification of norms for user behavior, of organization policies, of organizational behavior, of law, of deontic integrity constraints and even of some scheduling problems are all applications in which norms applicable to the behavior of people are specified in deontic logic. If the specification is implemented in a computer, then we have the novel situation that computers may actively deduce permissions, obligations or prohibitions applicable to people. This situation is novel in that it has never been done before in the application of deontic logic. However, it is not new in computer science to let computers direct human affairs, although applications till now have not been controversial. As Sergot [53] points out, even a payroll program applies law in its computation of tax deductions and it therefore determines how much people are entitled to by law. Even the direct prescription of human actions by machines in general is not new, witness the use of traffic lights. However, the breadth and complexity of this use of machines increases dramatically if we reach the situation where we can implement deontic logic specifications in computers. A survey of possible applications of deontic logic in computer science should therefore contain a discussion of the possibility and admissibility to direct human affairs by computer. We discuss this under the headings of the *possibility* and the *authority* to do this.

4.1 The possibility to direct human affairs by computer

The use of a computer to direct human affairs is hotly debated in the field of artificial intelligence and law and we will restrict our discussion to this area. However, the discussion applies equally well to the application of company policies to people or to any of the other applications of deontic logic in computer science where the application of norms for human behavior is automated.

Computer scientists are prone to a simplistic view of the application of law to facts that basically takes it to be similar to the application of instructions to data by a computer. This is the view we assumed in our systematic list of applications of deontic logic to computer science above. Reality is more complex, however, and in order to assess the possibility of using computers to direct human behavior we must briefly turn

to the differences between the way people apply norms (e.g. laws) to facts on the one hand and the way computers would apply norms to facts on the other.

When a human judge applies law to facts, the laws are interpreted in the light of the relevant facts and the facts are simultaneously interpreted in the light of the relevant laws.[2] In The Netherlands, the places where the judge can find law are legislation issued by government, jurisprudence, accepted social custom and some international treaties. Only some of this is relevant in a particular case, and whatever is relevant is usually found in the form of general statements that will have to be particularized in terms of the facts at hand. Thus, choices are made from the multitude of possibly relevant sources of law, and these sources are interpreted to make them applicable. The choices and interpretations are made in the light of the facts to be judged.

Conversely, as information analysts know, everything in the world is, ultimately, connected to everything else, and from the potentially infinite set of possibly relevant facts a choice will have to be made in a particular case. For example, if a police officer notices a car parked on the pedestrian side-walk, its registration number will be noted and the time and place of the observation, but not the color of the car, the weather at that time of day nor the width of the pavement -unless for some strange reason some of this may appeared relevant to the police officer. In addition, the observation will be stated in terms defined in the law. The meaning of "car," for example, is defined in the law. So choices are made from the multitude of possibly relevant facts, and the chosen facts are interpreted in order to make the relevant law applicable. The choices and interpretations are made in the light of the laws to be applied.

Contrast this with what happens if we program a machine to apply a computer representation of law to a computer representation of facts. First, an area of the law is selected, such as the British Nationality Act [54], corporate tax law [38], or latent damage law [11]. This selection is not done by a judge in the light of facts that constitute a possible violation of law, but it is done by a legal specialist (or a knowledge engineer) with a view to the feasibility of representing this part of the law in a computer. This involves criteria such as technical complexity and relative isolation from the rest of the law, to which we turn below. Whatever criteria are used, they precede the selection of relevant facts in a case and are therefore not used in the light of such facts.

Next, written legislation and possibly also jurisprudence is translated into a computer representation. This translation involves interpretation, but not in the light of facts to be tried, for these are not there yet, but in the light of the understanding of one or more persons. For example, the British Nationality Act was programmed in Prolog according to the understanding of F. Sadri [54] and the Latent Damage Law system expresses the understanding of P.N. Capper [11].

Third, when the system is applied to facts, these facts have to be presented to the system in some form suitable for storage and manipulation. This involves interpretation by the person doing this. The facts should be presented to the system on the basis

[2]This is explained very clearly in a textbook used at some Dutch universities as introduction to the philosophy of law [2]. Sergot [53, pages 18-31] contains a useful summary of discussion, conducted usually in the area of the philosophy of law, of the question whether judges simply apply laws to facts, and places this discussion in the context of the computer representation of law. See also Susskind's Ph.D. Thesis [56] for an extensive discussion.

of terms known to the system, and since these terms are computer representations of already interpreted terms defined in the law, this is quite different from the interpretation of facts in the light of the law that occurs in real life.

Finally, we have a computer representation of law that is applied to a computer representation of facts. In the end, all a computer does at this stage is applying instructions to data, but this process is more intelligibly described at a higher level as the manipulation of data according to certain rules. These may be the rules of some subset of first-order logic such a declarative Prolog, or some more obscure set of rules such as procedural Prolog (which includes mechanisms such as cut which or not well-understood), some other kind of rule-based mechanism, or one of the many versions of deontic logic that are around. Whatever mechanism is used, it should be clear that this is not a representation of the legal reasoning process of judges and lawyers -nor is it claimed to be, we should add.

Importantly, the choices and interpretations that have played a role until now -choices and interpretations of facts and laws, choice of the rules of manipulation- are invisible at this stage. For this reason, the application of a computer representation of law to a computer representation to facts is often thought to be impartial and less subjective than the application of law to facts by a judge. This is far from the truth, of course, for the choices and interpretations are merely made invisible by the use of a computer [33]. Instead of saying that the computer is impartial and objective, we should say that its partiality and subjectivity are fixed.

To sum up, it should be clear that the process by which a human judge applies law to facts is quite different from the process that leads to a computer applying (a computer representation of) law to (a computer representation of) facts. Just as in the case of the application of law to facts by a judge, in an application of a computer representation of law to a computer representation of facts, choices are made and interpretation takes place; but the choices and interpretations of law are independent of the choices and interpretation of facts, and both are invisible once the computer representation of law is applied to the computer representation of facts. Absence of the human interpreter gives a semblance of impartiality to the application of a computer representation of law to the computer representation of facts.

Having made clear that the application of law by computer differs radically from the application of a computer representation of law by machines, how can the second kind of process be of use in the first? We can distinguish the following conditions that, in one way or another, circumvent the problems generated by the differences between the application of law to facts by a human judge and the application of a computer representation of law to a computer representation of facts.

- The law selected for computer representation is to a large extent isolated from other areas of the law. This avoids problems with terms defined in other parts of the law and in general keeps the size of the system manageable. All areas of law selected for computer representation in the examples above -corporate tax law, agricultural tenancies, British citizenship- satisfy this condition.

- Applying the law selected for computer representation requires little common sense. This avoids one of the hardest problems of artificial intelligence, the com-

puter representation of common sense reasoning. See also Thorne McCarty [39] for a discussion of this issue.

- The selected area of law has an unambiguous interpretation, agreed upon by the legal community. This avoids the problem of the invisible interpreter who introduces subjectivity in the guise of mechanical impartiality. Sergot [53] makes this requirement explicit.

- The selected area of law is not subject to change, at least not the kind of change that is realized by a process of ongoing jurisprudence. This way, we avoid having to program the computer as a machine that participates in a social learning process. Learning is another hard problem of artificial intelligence, let alone learning by participating in a social process. On a more mundane level, selecting stabile areas of law avoids the problem of having to frequently change a computer representation of law by hand and therefore simplifies system maintenance.[3]

Avoiding problems is not a way of solving them and we are not saying that these problems should not be researched in artificial intelligence nor that we should not try to tackle harder problems in the application of deontic logic in computer science. However, we are saying that at the current state of research, it seems reasonable to believe that the computer representation of law could be *practically feasible* when these conditions are satisfied. Even here, success is not guaranteed, however. We mentioned already that even seemingly simple systems of norms like library regulations involve some of the deepest paradoxes of deontic logic, such as Chisholm's paradox.

We should point out that possibility does not imply usefulness. If we succeed in programming routine normative problems such as those that satisfy the above list into a machine, then the resulting system *may* be useful under certain conditions, that should be added to the above list. One reason for usefulness is a backlog in the processing of routine cases. This reason is operative in some of applications that are closest to being actually used: the TESSEC system that automates decisions about entitlement to welfare benefit [46] may be used in the near future to alleviate work-pressure in Dutch social service departments [19], and the allocation of military conscripts to parts of the military forces in The Netherlands, which must be done for almost a million conscripts per year, has been automated [37] for the same reason: the sheer volume of the cases together with the routine-like nature of most of the problems.

Another reason why automating the application of a computer representation of law to a computer representation of facts may be useful is that the area of law is

[3]It also avoids the problem of having to represent *open-textured concepts*. Open texture is the phenomenon that the set of all possible instances of a concept is not determined in advance of the application of that concept [9]. An example is the concept of a boat, for which it at was not clear until recently whether a surf plank was an instance of it. In the legal context, only a judge has the authority to decide whether an object is an instance of an open-textured concept or not. In The Netherlands, for example, a surf plank was ruled by court not to be a boat, so that the Vessel Traffic Law is not applicable to surf planks. Open texture is one of the ways a law-giver deals with unforeseen change. It is impossible to foresee in advance all possible applications of concepts like boat. Open texture is essential to facilitate legal change but its computer representation is one of the hard problems of artificial intelligence.

technically so complex that human lawyers have problems understanding and applying it. Corporate tax law (TAXMAN [38]) and latent damage law [11] are examples of this kind of use. Incidentally, these two reasons nicely illustrate the reasons why computers are useful in general: reliable high-speed data processing and reliable high-volume data storage.

4.2 The authority to direct human affairs by computer

Whenever we tell people what to do or not to do or what rights they have, we utter a *speech act* that under certain conditions counts as a command [48]. One important condition under which our utterance can consider to have directive force for people is that we have the proper *authority* to issue a prescription [50]. The question we want to discuss here is whether a computer can have such an authority.

A hint that the answer may affirmative is the fact that we are obligated to obey the instructions issued to us by a traffic light. If a traffic light switches its color to red, we may construe this as an act of the appropriate authorities, which they *delegated* to a machine (a system of traffic lights). The authorities, of course, keep the responsibility for the acts of the traffic light but the question is whether we can meaningfully say that the lights have the *authority* to tell us to stop. This has nothing to do with machine intelligence and is purely a matter of social convention, or whatever it is that makes a normative structure of rules and authorities exist. Now, delegation is an act in which the authority to perform a task is transferred to someone, but the final responsibility for the performance of the task is kept by the one who delegates the task. This is the formal structure of, for example, the relation between a shop assistant and his or her employer. The assistant sells goods, and receives money from the customer for the goods, in the name of his or her employer. The employer acts through the assistant, just as the appropriate authorities act through the traffic light. To see that the two situations are analogous, compare a person who impersonates a shop assistant without having received proper authority to act as a shop assistant to a traffic light put on the road by someone who likes making traffic lights as a hobby but has no authority to put them on the road. This person is like someone who tells his friend to impersonate a shop assistant, and the traffic light is like the would-be shop assistant who behaves like the real thing but has no proper authority to *be* the real thing. In both cases, the observable behavior of the illegal surrogates is indistinguishable from that of the real thing; the difference is in the absence of a delegation of authority in the surrogate case. Although we think more research should be done on this topic, tentatively we would say that the delegation of authority to a machine is possible, and even necessary, if the machine is to issue instructions to people.

Now take as an example Lee's [32] example of a computer system programmed in deontic logic with the regulations for granting parking permits on a university compound. Suppose a car owner requesting permission to park on the university compound puts this question to a clerk, who types in the required data in the computer system and receives an answer "permission granted" or "permission refused." There are two interpretations of what happens here.

1. The clerk has been delegated the authority to grant the permits and types in the data, and his or her decision in the computer system. The system checks if the permit has been granted according to the rules and raises a violation flag if it discovers an error. Since the clerk grants or refuses the permit, however, the clerk may overrule the advice of the system. For example, he or she may write the decision on a card, to be entered in the system after its rules have been corrected, or the system may have been programmed in deontic logic and the clerk's decision may be entered immediately (but the rules must be corrected anyway, because a violation was incorrectly raised). To save time, the computer may even apply the rules itself and inform the clerk of the result. Even in this case, the clerk has the authority to grant the permits, and he can overrule the advice of the machine.

2. The computer has been delegated the authority to grant or refuse permits and the clerk just enters the appropriate data. In this case, the clerk cannot overrule the decision of the machine if he or she has a different opinion.

In both cases, there is (or should be) a possibility for the car owner to appeal to a higher authority of he or she does not agree with the decision. What concerns us here is the relation between the clerk and the computer system in these two cases.

In the first case, the system is used as a *legal advice system* and in the second case, as a *automated judge*, taking "judge" in a wide sense as any entity that or who has authority to issue prescriptions (permissions, obligations, prohibitions) to people. The difference between the two cases is *not* observable in the behavior of the machine. The same program can be used in both cases, and it may be as dumb or smart a program as we can write. The crucial difference is one of social structure surrounding the human-machine system, viz. the distribution of authority is different. An important difference implied by this is that the bearer of authority in the first case is also someone who, as a human being, is a bearer of responsibility. In the second case, the bearer of authority is a machine and has no responsibility whatsoever.

This small example makes clear that the method of introducing an automated system that issues commands or permissions that apply to people must be contain special procedures. During the conversion from a manual to an automated parking permit system, for example, it was, or should have been, made clear to all parties involved that from now on the machine has the authority to grant parking permits, and the relevant people should also agree with the state of affairs. Also, the allocation of responsibility to someone responsible for the acts of the machine should have been made clear to all. Clearly, this is an interesting and urgent area of further research and here we can only point out that it exists and needs to be pursued. More on this can be found in Wieringa [58]. Dewitz [15] contains an interesting discussion of these issues in the case of an EDI network that autonomously takes legal actions.

5 Conclusions

We reviewed a number of applications of deontic logic in computer science in chronological order and then systematically classified them non-exhaustively as the specification in deontic logic of

- fault-tolerant systems,
- desired user behavior,
- company policies (including security policies),
- organization behavior (i.e. contracting),
- law,
- legal thinking,
- normative integrity constraints, and
- scheduling under normative constraints.

In all these cases, the difference between actual and ideal behavior is relevant and in all cases, we have a choice to implement the specification in a computer. Finally, we noted that many applications of deontic logic in computer science concern the automation of normative utterances directed at people. We identified routine problems that satisfy a number of conditions as potential applications of this kind of automation, and we pointed out that the matter who has the authority for these automated decisions, and who is responsible for them, should be resolved before such systems are actually used.

One of the exciting features in the application of deontic logic in computer science is the mix of challenging technical problems, hard problems of the philosophy of law, and practical problems of social morality, that are encountered. This calls for further interdisciplinary research in all these areas and in their interaction.

References

[1] C.E. Alchourrón and E. Bulygin. *Normative Systems*. Springer, 1971.

[2] N.E. Algra and K. van Duyvendijk. *Rechtsaanvang: Hoofdstukken over recht en rechtswetenschap voor het onderwijs in de Inleiding tot de Rechtswetenschap. 4e druk*. Samsom H.D. Tjeenk Willink, 1989.

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

[4] L.E. Allen. Language, law and logic: Plain drafting for the electronic age. In B. Niblett, editor, *Computer Science and Law*, pages 75–100. Cambridge University Press, 1980.

[5] L.E. Allen. Towards a normalized language to clarify the structure of legal discourse. In A.A. Martino, editor, *Deontic Logic, Computational Linguistics and Legal Information Systems*, volume 2, pages 349–407. North-Holland, 1982. Edited versions of selected papers from the international conference on "Logic, Informatics, Law," Florence, Italy, April 1981.

[6] L.E. Allen and C.S. Saxon. A-Hohfeld: A language for robust structural representation of knowledge in the legal domain to build interpretation-assistance expert systems. This volume.

[7] L.E. Allen and C.S. Saxon. Analysis of the logical structure of legal rules by a modernized and formalized version of Hohfeld fundamental legal conceptions. In A.A. Martino and F.S. Natali, editors, *Automated Analysis of Legal Texts*, pages 385–450. North-Holland, 1986. Edited versions of selected papers from the Second International Conference on "Logic, Informatics, Law," Florence, Italy, September 1985.

[8] A.R. Anderson. The formal analysis of normative systems. In N. Rescher, editor, *The Logic of Decision and Action*, pages 147–213. University of Pittsburgh Press, 1967.

[9] T. Bench-Capon and M. Sergot. Toward a rule-based representation of open texture in law. In C. Walter, editor, *Computer Power and Legal Language*, pages 39–60. Quorum Books, 1988.

[10] C. Biagioli, P. Mariani, and D. Tiscornia. ESPLEX: A rule and conceptual based model for representing statutes. In *The First International Conference on Artificial Intelligence and Law*, pages 240–251. ACM, May 1987.

[11] P.N. Capper and R.E. Susskind. *Latent Damage Law - The Expert System*. Butterworths, 1988.

[12] H.-N. Castañeda. *Thinking and Doing. The Philosophical Foundations of Institutions*. Reidel, 1975.

[13] R.M. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24:33–36, 1963.

[14] C. Ciampi, editor. *Artificial Intelligence and Legal Information Systems*, volume 1. North-Holland, 1982. Edited versions of selected papers from the international conference on "Logic, Informatics, Law," Florence, Italy, April 1981.

[15] S.D. Dewitz. Contracting on a performative network: Using information technology as a legal intermediary. In R.K. Stamper, P. Kerola, R. Lee, and K. Lyytinen, editors, *Collaborative Work, Social Communications and Information Systems*, pages 271–293. North-Holland, 1991.

[16] J. Fiadeiro and T. Maibaum. Temporal reasoning over deontic specifications. *Journal of Logic and Computation*, To be published.

[17] J. Glasgow, G. MacEwen, and P. Panangaden. Security by permission in databases. In C.E. Landwehr, editor, *Database Security II: Status and Prospects*, pages 197–205. North-Holland, 1989. Results of the IFIP WG 11.3 Workshop on Database Security (October 1988), Kingston, Ontario, Canada.

[18] G.B. Gray. Statutes enacted in normalized form: The legislative experience in Tennessee. In C. Walter, editor, *Computer Power and Legal reasoning*, pages 467–493. West Publishing Co., 1985.

[19] H.J. van den Herik. *Kunnen Computers Rechtspreken?* Gouda Quint B.V., 1991.

[20] W.N. Hohfeld. Fundamental legal conceptions as applied to judicial reasoning. *Yale Law Journal*, 23:16–59, 1913.

[21] *The First International Conference on Artificial Intelligence and Law*. Association for Computing Machinery, May 1987.

[22] *The Second International Conference on Artificial Intelligence and Law*. Association for Computing Machinery, June 25-28 1989.

[23] *The Third International Conference on Artificial Intelligence and Law*. Association for Computing Machinery, June 25-28 1991.

[24] A.J.I. Jones. Deontic logic and legal knowledge representation. *Ratio Juris*, 3:237–244, 1990.

[25] A.J.I. Jones and I. Pörn. Ideality, sub-ideality and deontic logic. *Synthese*, 65:275–289, 1985.

[26] A.J.I. Jones and M. Sergot. On the role of deontic logic in the characterization of normative systems. This volume.

[27] S. Jones, P. Mason, and R. Stamper. LEGOL 2.0: A relational specification language for complex rules. *Information Systems*, 4:157–169, 1979.

[28] S. Khosla. *System Specification: A Deontic Approach*. PhD thesis, Department of Computing, Imperial College, London, 1988.

[29] S. Khosla and T.S.E. Maibaum. The prescription and description of state based systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, pages 243–294. Springer, 1987. Lecture Notes in Computer Science 398.

[30] S.O. Kimbrough, R.M. Lee, and D. Ness. Performative, informative and emotive systems: The first piece of the PIE. In L. Maggi, J.L. King, and K.L. Kraenens, editors, *Proceedings of the Fifth Conference on Information Systems*, pages 141–148, 1984.

[31] R.M. Lee. Bureaucracies as deontic systems. *ACM Transactions on Office Information Systems*, 6:87–108, 1988.

[32] R.M. Lee. A logic model for electronic contracting. *Decision Support Systems*, 4:27–44, 1988.

[33] M. Lupoi. The decisional computer as a source of law. In C. Ciampi, editor, *Artificial Intelligence and Legal Information Systems*, volume 1. North-Holland, 1982. Edited versions of selected papers from the international conference on "Logic, Informatics, Law," Florence, Italy, April 1981.

[34] A.A. Martino, editor. *Deontic Logic, Computational Linguistics and Legal Information Systems*, volume 2. North-Holland, 1982. Edited versions of selected papers from the international conference on "Logic, Informatics, Law," Florence, Italy, April 1981.

[35] A.A. Martino, editor. *Preproceedings, Third International Conference on "Logic, Informatics, Law"*, Firenze, Italy, 1989.

[36] A.A. Martino and F.S. Natali, editors. *Automated Analysis of Legal Texts*. North-Holland, 1986. Edited versions of selected papers from the Second International Conference on "Logic, Informatics, Law," Florence, Italy, September 1985.

[37] L.V. Mazel. Geautomatiseerd beschikken, een praktijkvoorbeeld. In E.M.H. Hirsch Ballin and J.A. Kamphuis, editors, *Trias Automatica*, pages 89–94. Kluwer, 1985.

[38] L.T. McCarty. Reflections on TAXMAN: An experiment in artificial intelligence and legal reasoning. *Harvard Law Review*, 90:837–893, March 1977.

[39] L.T. McCarty. Some requirements for a computer-based legal consultant. In *Proceedings of the First Annual National Conference on Artificial Intelligence*, pages 298–300, Standford, August 1980.

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

[41] L.T. McCarty. Permissions and obligations: An informal introduction. In A.A. Martino and F.S. Natali, editors, *Automated Analysis of Legal Texts*, pages 307–337. North-Holland, 1986. Edited versions of selected papers from the Second International Conference on "Logic, Informatics, Law," Florence, Italy, September 1985.

[42] L.T. McCarty. A language for legal discourse I: Basic features. In *The Second International Conference on Artificial Intelligence and Law*, pages 180–189. Association for Computing Machinery, June 25-28 1989.

[43] J.-J.Ch. Meyer. A simple solution to the 'deepest' paradox in deontic logic. *Logique et Analyse, Nouvelle Série*, 30:81–90, 1987.

[44] J.-J.Ch. Meyer and R.J. Wieringa. Deontic logic: A concise overview. This volume.

[45] N.H. Minsky and A.D. Lockman. Ensuring integrity by adding obligations to priviliges. In *8th IEEE International Conference on Software Engineering*, pages 92–102, 1985.

[46] M.A. Nieuwenhuis. *TESSEC: Een expertsysteem voor de Algemene Bijstandswet*. PhD thesis, Universiteit Twente, 1989.

[47] N. Rescher and A. Urquhart. *Temporal Logic*. Springer, 1971.

[48] J. Searle. *Speech Acts. An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[49] J. Searle and D. Vanderveken. *Foundations of Ilocutionary Logic*. Cambridge University Press, 1985.

[50] J.R. Searle. A taxonomy of illocutionary acts. In *Expression and Meaning*, pages 1–29. Cambridge University Press, 1979.

[51] M. Sergot. Prospects for representing the law as logic programs. In K.L. Clark and S.-A. Tärnlund, editors, *Logic Programming*, pages 33–42. Academic Press, 1982.

[52] M. Sergot. Representing legislation as logic programs. In J.E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11*, pages 209–260. Clarendon Press, 1988.

[53] M. Sergot. The representation of law in computer programs: A survey and comparison. In T.J.M. Bench-Capon, editor, *Knowledge Based Systems in the Law*. Academic Press, 1990.

[54] 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.

[55] R. Stamper. LEGOL: Modelling legal rules by computer. In B. Niblett, editor, *Com[puter Science and Law*, pages 45–71. Cambridge University Press, 1980.

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

[57] J.E. Tomberlin. Contrary-to-duty imperatives and conditional obligation. *Noûs*, 16:357–375, 1981.

[58] R.J. Wieringa. Three roles of conceptual models in information system design and use. In E.D. Falkenberg P. Lindgreen, editor, *Information System Concepts: An In-Depth Analysis*, pages 31–51. North-Holland, 1989.

[59] R.J. Wieringa, J.-J. Ch. Meyer, and H. Weigand. Specifying dynamic and deontic integrity constraints. *Data and Knowledge Engineering*, 4:157–189, 1989.

[60] R.J. Wieringa and J.-J.Ch. Meyer. Actors, actions, and initiative in normative system specification. Technical Report IR-257, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, October 1991. Submitted for publication.

[61] R.J. Wieringa, H. Weigand, J.-J. Ch. Meyer, and F. Dignum. The inheritance of dynamic and deontic integrity constraints. *Annals of Mathematics and Artificial Intelligence*, 3:393–428, 1991.

[62] G.H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.

[63] G.H. von Wright. *And Next*. Acta Philosophica Fennica, Fasc. 18. North-Holland, 1965.

[64] G.H. von Wright. *An Essay in Deontic Logic and the General Theory of Action*. Acta Philosophica Fennica, Fasc. 21. North-Holland, 1968.

Contents

1 Introduction **1**

2 A chronological survey of applications **2**
2.1 Legal automation 2
2.1.1 The factual and deontic approaches to legal automation 2
2.1.2 Examples of deontic approaches 4
2.2 Authorization mechanisms (Minsky and Lockman 1985) 5
2.3 System specification (Khosla and Maibaum 1987) 7
2.4 Electronic contracting (Lee 1986) 8
2.5 Deontic integrity constraints (Wieringa, Meyer and Weigand 1989) . . 9
2.6 Database security (Glasgow, MacEwen and Panangaden 1989) 12

3 A systematic view of applications in computer science **12**

4 Discussion: directing human behavior by computer **16**
4.1 The possibility to direct human affairs by computer 16
4.2 The authority to direct human affairs by computer 20

5 Conclusions **22**