DX: A Deontic Expert System

Ronald M. Lee*
Young U. Ryu§

* Erasmus University Research Institute
  for Decision and Information Systems (EURIDIS)
  Erasmus University
  3062 PA Rotterdam
  The Netherlands
  leer@fac.fbk.eur.nl

§ JO 4.4, P.O. Box 830688
  The University of Texas at Dallas
  Richardson, Texas 75083-0688
  U.S.A
  ryoung@utdallas.edu

(November 21, 1994)


* Ronald M. Lee is currently Director of the Erasmus University Research Institute for Decision Information System (EURIDIS) in Rotterdam, the Netherlands. Prior to that, he was Associate Professor of Information Systems at the Management Science and Information Systems Department at the University of Texas at Austin. He has a Ph.D. in Decision Sciences (Wharton, 1980), and has previously served as a research scholar at the International Institute for Applied Systems Analysis in Vienna, Austria, and as Visiting Professor of Management at the Universidade Nova de Lisboa, in Lisbon, Portugal. Current research focuses on applications of artificial intelligence to business; special focus is ‘logic modeling,’ the use of formal logic representations for management science applications; current projects involve the use of logic modeling to represent and manage formal business communications systems focusing on electronic commerce and governmental bureaucracies. Theoretical aspects include the role of deontic and illocutionary logic in representing formal business conversations. Practical value is the reduction of bureaucratic and legalistic red tape. This work also includes multi-lingual business communications, supporting structured communications between parties with different native languages.

§ Young U. Ryu is Assistant Professor of Information Systems, Decision Sciences Department, The University of Texas at Dallas. He received a Ph.D. degree in Management Science and Information Systems from The University of Texas at Austin. His research interests include logic-based modeling of bureaucratic policies and law, defeasible reasoning, non-monotonic logic, artificial intelligence applications of management, and constraint logic programming of mathematical models.
Social norms have a pervasive effect on our lives, affecting how and what we eat, how we
dress, where and how we travel, our recreation, the way we work, and our participation in
the society at large. Other social norms make constraints on the behavior of commercial
companies and other institutions, affecting the way they do business, their treatment of em-
ployees, their treatment of the environment, and so on. As societies grow and become more
diverse, the system of norms becomes correspondingly complex. This paper is an attempt to
bring computational assistance to bear in managing and interacting with these normative
systems. A prototype expert system is described that utilizes deontic rules for reasoning
about normative constraints in organizations and other social systems. Applications to bu-
reaucracies and electronic contracting systems are discussed. DX can be used to model
regulations and policies in organizations, such as library regulations of universities, resource
access policies, etc. Also, regulatory aspects in interorganizational systems, e.g. EDI, can be
effectively modeled and managed by DX. Automated regulation and policy reasoning help
manage normative rules and reduce complexity in their applications.

Key Words and Phrases: deontic logic; expert systems; performatives; normative systems;
bureaucratic systems; electronic contracting systems
1. INTRODUCTION: THE UBIQUITY OF SOCIAL NORMS

Social norms are the basis of every civilization. They are the rules and conventions that guide the conduct of members of a society: how we dress, what and how we eat, where and how we travel, our recreation and the way we work. A primary purpose of social norms is to ensure equity. Some norms, such as styles of dress or table manners, are mainly conventional. Other norms have a more serious purpose of ensuring social equity. Thus we have norms forbidding violence, theft, fraud, etc. where one individual takes unfair advantage of another. Still other norms serve the function of coordinating social activity. An example is traffic regulations, which dictate which side of the street to drive on, speed limits, when to stop, yield, and so on, in order to coordinate traffic flow. Similarly, other norms regulate commercial activity, transport, the postal service, telecommunications, and other types of services where a large number of individuals, companies and institutions must interact cooperatively. Still other kinds of social norms are for purposes of social management, serving both equity and coordination objectives. Examples include taxation, military obligations, immigration and naturalization, import/export regulations, entitlements such as unemployment and housing benefits, public health services, retirement benefits, etc. All of these norms have the common feature that they regulate human behavior.

Social norms vary in the degree they are articulated or made explicit. For example, it is a norm in Western societies that men wear darker, more conservative colored clothes than women, though this is seldom stated explicitly. At the other extreme, certain norms are codified in the form of laws and published for the attention of all members of the society. Intermediate are organizational norms, the duties and conventions of the work place. Often these are communicated through informal on the job training; in more bureaucratic organizations the norms may be stated explicitly in the form of a job description.

For every norm, there is an enforcement or sanction imposed on violators. When informal norms are violated, e.g. bad table manners or outlandish dress, the sanction is typically disapproval—a raised eyebrow, a sarcastic remark. For civil and criminal laws, the sanction is more severe: a fine, a jail term, perhaps even execution. Again, organizational norms are an intermediate case. Most organizational norms are enforced by disapproval, perhaps a formal reprimand. More severe violations may lead to denial of a pay raise, demotion, or termination.

As the society grows more complex, with more specialization of roles, and a wider range of activities both in work and leisure, the system of social norms also grows more complex. For instance, the EEC, attempting to regulate and coordinate all the diversity of the common market countries, is said to produce some 15,000 pages of reports and memoranda per day! (Most of which is translated to each of the nine languages of the member countries, multiplying the paper produced by nearly an order of magnitude.) Much of this documentation has to do with the adjustment of social norms. As the system of norms grows more complex,
it also becomes more confusing, both to the people who must abide by them, as well as to
the administrators who need to modify them. This has two undesirable effects. One, the so-
cial system becomes encumbered by bureaucratic overhead and so becomes less efficient.
Two, the ability to make changes and adjustments to the system becomes increasingly more
difficult, leading to bureaucratic rigidity.

This is a situation where technological support is warranted. What we have in mind is a
type of expert system whose rules represent the system of norms, and which is able to ad-
vise users of the rights, duties and privileges that pertain to their specific circumstances.
Furthermore, such an expert system should also provide aids to administrators in helping to
adapt the rule-base and determine the consequences of changes. The system would be made
accessible to members of the organization or society through a computer network. Thus,
changes in normative rules would be instantly available. As a practical consideration, such a
system would only be workable for norms that are fairly explicit, thus subject to formaliza-
tion as rules, e.g. in areas such as income tax regulations, unemployment and other welfare
benefits, health insurance, immigration and naturalization, etc.

A problem arising from this proposal is that reasoning about norms involves a different
kind of logic than is typically employed in expert systems. Norms are not true or false in the
usual sense; for instance a policy of no smoking does not guarantee that no one will smoke.
What is needed is a different kind of rule representation and inferencing mechanism.

In the logic literature, normative reasoning is the subject of deontic logic, which formal-
izes such concepts as obligation, permission and prohibition. Borrowing from this, we pro-
pose the notion of a deontic expert system, which incorporates deontic concepts in its rules
and inferencing. Specifically, we will present a prototype deontic expert system, called DX,
and demonstrate its operation and applicability. In later sections we discuss extensions to
this basic model and its potential application to large scale normative systems.

2. COMPUTING DEONTICS

2.1. Standard Deontic Logic

Deontic logic has its origin in the classical philosophy of ethics. The modern development of
deontic logic was initiated in the early 1950’s by von Wright [35] who coined the term, based
on the Greek ‘δεντικος’ meaning “as it should be” or “duly.” Deontic logic is a logic of nor-
mative concepts. Its major application, outside of ethics, has been to the philosophy of law.
The practical relevance of deontic logic in administrative contexts is to provide automatic
inference in, say, contract arbitration or the interpretation of bureaucratic regulations. Such
applications are useful in complex cases where the chain of connections would otherwise be
difficult to follow. Thus, the axioms and inference rules of deontic logic take on practical
importance for normative systems that are complex yet explicit, amenable to formalization.
The first axiomatization for deontic logic was proposed by von Wright [35, 36]. A basic concept is captured by the operator:

\[ O\phi \]

read that “\( \phi \) is obliged.” Based on this, a notion of permission can be defined as its logical dual:

\[ P\phi =_{\text{def}} \neg O\neg \phi \]

that is, “\( \phi \) is permitted” if and only if “it is not obliged not to do \( \phi \).” A related concept of prohibition was defined as:

\[ F\phi =_{\text{def}} O\neg \phi \]

that is, “\( \phi \) is forbidden” if and only if “it is obliged not to do \( \phi \).” For completeness, we also add a notation for waiver (of an obligation):

\[ W\phi =_{\text{def}} \neg O\phi \]

that is, “\( \phi \) is waived” if and only if “it is not obliged to do \( \phi \).”

Various axiomatic systems of deontic logic have been proposed. In an introductory survey, Føllesdal and Hilpinen [12] present what they call the standard system of deontic logic. Based on propositional logic, this serves as a more or less consensually accepted core on which to base further discussion. The standard system assumes elementary generic actions (in the sense of von Wright [36]). Assuming \( \phi \) and \( \psi \) to be actions of this type, the standard system has the following axioms:

\begin{align*}
\text{[DA1]} & \quad O\phi \rightarrow \neg O\neg \phi \quad \text{(or equivalently } O\phi \rightarrow P\phi ) \\
& \quad \text{If } \phi \text{ is obliged, then } \phi \text{ is permitted.} \\
\text{[DA2]} & \quad O(\phi \land \psi ) \leftrightarrow O\phi \land O\psi \\
& \quad \phi \text{ and } \psi \text{ are together obliged if and only if they are obliged separately.} \\
\text{[DA3]} & \quad O(\phi \lor \neg \phi ) \\
& \quad \text{It is obliged to either do or not do } \phi .
\end{align*}

Interesting theorems resulting from the above axioms and predicate calculus are:

\begin{align*}
\text{[DT1]} & \quad F\phi \rightarrow W\phi \\
\text{[DT2]} & \quad O(\phi ) \land O(\phi \rightarrow \psi ) \rightarrow O(\psi ) \\
\text{[DT3]} & \quad P(\phi ) \land O(\phi \rightarrow \psi ) \rightarrow P(\psi ) \\
\text{[DT4]} & \quad F(\psi ) \land O(\phi \rightarrow \psi ) \rightarrow F(\phi ) \\
\text{[DT5]} & \quad W(\psi ) \land O(\phi \rightarrow \psi ) \rightarrow W(\phi )
\end{align*}

In fact, replacing the axiom [DA2] by the theorem [DT2] results in the same deontic logic system [12].

November 21, 1994
The deontic formulation $O(\phi \rightarrow \psi)$, which is often symbolized as $C(\phi, \psi)$, is called the commitment expression [35], whose behavior is expressed in [DT2–5]. A commitment expression is used to transform a deontic statement to another. For example, in an agency relation, duties and discretions of the principal are transferred to the agent; that is, an agency relation may be represented by commitment expressions.

The sense of the deontic operators obviously relies on what is meant by an action. Von Wright [36: p. 16] comments:

A few words should be said about the reading of the formulae. In my first construction of a system of deontic logic the variables were treated as schematic names of actions. … According to this conception, “$Pp$” could be read “It is permitted to $p$.” This conception, however, is connected with difficulties and inconveniences. It is, first of all, not clear whether the use of truth-connectives for forming compound names of actions is logically legitimate. …

It now seems to be better to treat the variables as schematic sentences which express propositions. … Instead of “proposition” we can also say “possible state of affairs.” According to this conception, “$Pp$” may be read “it is permitted that (it is the case that) $p$.”

Against this reading, however, it may be objected that it does not accord very well with ordinary usage. Only seldom do we say of a state of affairs that it is permitted, obligatory, or forbidden. Usually we say this of actions. But it is plausible to think that, when an action is permitted, etc., then a certain state of affairs is, in a ‘secondary’ sense, permitted, etc. too. This is the state which, in a technical sense $p$ can be called the result of the action in question. … The formula “$Pp$” is thus read “it is permitted to see to it that (it is the case that) $p$.”

In the axioms, $\phi$ and $\psi$ are regarded as propositional variables. However, they are not exactly propositions in the usual sense of referring to a static state of affairs, e.g. the window is closed. Rather, as names for generic actions, they refer to someone’s causing a certain state to occur, e.g. closing the window. Thus, in this form of deontic logic, the concept of truth value is replaced by one of performance value [36], i.e. whereas a proposition is either true or false, an action is either performed or not performed. Further, the actions controlled by these deontic operators presume an aspect of human agency. We don’t obligate or permit natural phenomena such as the sun rising.

### 2.2. Computing Process for Deontic Reasoning

For our purposes, we would like to develop a computational version of deontic reasoning. This presents several challenges. First, the practical needs of an expert system for deontic reasoning require that we be able to distinguish how deontic status varies for different individuals, e.g. in a contract, that Smith has an obligation to deliver a certain good, while Jones has an obligation to pay a certain amount of money. This leads us to pursue first-order deontic reasoning as propositional logic and its extension to predicate logic (see e.g. [8: Ch. 12]). A second issue is that there is very little literature, and no readily available software, for computing modal operators. A third consideration, aside from computability, is the availability of a suitable implementation environment that facilitates programming and user interface development. These aspects taken together lead us to attempt to couch the deontic
concepts within the framework of program clause logic, computable through various logic programming languages, such as Prolog. Logic programming [3, 24] is a subset of first order logic with functions, where rules are expressed in the form:

$$\phi \leftarrow \psi_1, \psi_2, \ldots, \psi_n$$

where $\phi$ and $\psi_i$ ($i = 1, 2, \ldots, n$) are literals and commas between literals denote conjunction. Disjunction is expressed by having multiple rules for the same conclusion.

In short, we adopt aspects of deontic logic and develop a first order program clause deontic reasoning system. The deontic reasoning system, implemented as DX, is not a deduction system for deontic logic; instead it is an expert system that provides certain reasoning features adopted from deontic logic.

The application of logic programming for deontic reasoning requires proper treatments of deontic operators. We introduce one-place predicates and a two-place predicate:

- oblig($\alpha$) (corresponding to $O(\alpha)$)
- permit($\alpha$) (corresponding to $P(\alpha)$)
- forbid($\alpha$) (corresponding to $F(\alpha)$)
- waive($\alpha$) (corresponding to $W(\alpha)$)
- commit($\alpha, \beta$) (corresponding to $C(\alpha, \beta)$)

where $\alpha$ and $\beta$ are terms denoting actions. Note that we do not allow compound terms as arguments of the above predicate symbols. That is, for example, we do not allow an expression oblig($\alpha \& \beta \lor \gamma$). The definitional relationships between deontic predicates are:

1. [DR1] $\text{oblig}(\alpha) = \neg \text{waive}(\alpha)$ (from definitions of deontic operators)
2. [DR2] $\text{permit}(\alpha) = \neg \text{forbid}(\alpha)$ (from definitions of deontic operators)

Further, from axioms and theorems, we have:

1. [DR3] $\text{permit}(\alpha) \leftarrow \text{oblig}(\alpha)$ (from [DA1])
2. [DR4] $\text{waive}(\alpha) \leftarrow \text{forbid}(\alpha)$ (from [DT1])
3. [DR5] $\text{oblig}(\beta) \leftarrow \text{oblig}(\alpha), \text{commit}(\alpha, \beta)$ (from [DT2])
4. [DR6] $\text{permit}(\beta) \leftarrow \text{permit}(\alpha), \text{commit}(\alpha, \beta)$ (from [DT3])
5. [DR7] $\text{forbid}(\alpha) \leftarrow \text{forbid}(\beta), \text{commit}(\alpha, \beta)$ (from [DT4])
6. [DR8] $\text{waive}(\alpha) \leftarrow \text{waive}(\beta), \text{commit}(\alpha, \beta)$ (from [DT5])

Given a program $P$ and a query $\phi$, we define a state:

$$\sigma_0 = \langle \phi, \emptyset \rangle$$

where $\emptyset$ is the empty substitution. Let us define possible transformal states of $\sigma_0$ as follows:

$$\sigma_1 = \langle \psi, \emptyset \rangle$$
where \( \phi = \psi \) from a definitional relationship between deontic predicates (i.e., \([\text{DR1}] \) or \([\text{DR2}] \)),

\[
\sigma_1^2 = \langle (\psi), \emptyset \rangle \text{ or } \sigma_2^2 = \langle (\psi_1, \psi_2), \emptyset \rangle
\]

where \( \phi \leftarrow \psi \) or \( \phi \leftarrow \psi_1, \psi_2 \) from a relationship of deontic predicates due to axioms and theorems of deontic logic (i.e., \([\text{DR3}–8] \)), or

\[
\sigma_3^3 = \langle (\psi_1 \theta_1, \psi_2 \theta_1, \ldots, \psi_n \theta_1), \theta_1 \rangle
\]

for every \( \psi \leftarrow \psi_1, \psi_2, \ldots, \psi_n \in P \) such that \( \phi \theta_1 = \psi \theta_1 \) where \( \theta_1 \) is a substitution:

\[
\theta_1 = \{x_1 = t_1, x_2 = t_2, \ldots, x_k = t_k\}
\]

where \( x_i \) (\( i = 1, 2, \ldots, k \)) are variables in \( \phi \) and \( \psi \) and \( t_j \) (\( j = 1, 2, \ldots, k \)) are terms in \( \phi \) and \( \psi \) such that \( x_i \neq t_j \) (for all \( i \) and \( j \)). Note that matching \( \phi \) with \( \psi \) by \( \theta_1 \) is called unification [3, 24]. We non-deterministically choose one of the above \( \sigma_i^j \) as the next transformal state. Say that the following is \( \sigma_1 \):

\[
\sigma_1 = \langle (\psi_1, \psi_2, \ldots, \psi_n), \theta_1 \rangle
\]

We further continue the above state transformation for each of \( \psi_i \) (\( i = 1, 2, \ldots, n \)), until we obtain:

\[
\sigma_m = \langle \emptyset, \theta_1 \theta_2 \cdots \theta_m \rangle
\]

where \( \theta_1 \theta_2 \cdots \theta_m \) is a composition of substitutions (as defined in [3, 24]). Then, \( \theta_1 \theta_2 \cdots \theta_m \) is given as an answer to the query. Generations of \( \sigma_n \) for all non-deterministic choices of state transformations are done via the backtracking techniques [3, 24].

The computing process in this section is obtained by program clause logic programming of selected deontic logic axioms and theorems. It can be summarized as follows. By not allowing compound terms as arguments of deontic operators, we have a first order reasoning system. By expressing deontic axioms and theorems as program clauses, we made logic programming of deontic features possible.

2.3. Toward a Deontic Expert System

An expert system is an interactive computer system [6] that can “help solve complex, real-world problems” using “large bodies of domain knowledge ... gleaned from human experts” [7: p. 79]. The term ‘expert system’ is frequently used in two senses: one as an expert system application (e.g. MYCIN [31]) providing domain-specific knowledge, the other as a knowledge engineering system (e.g. EMYCIN [34]) for developing expert system applications of particular subject domains, which we call an expert system shell. An expert system shell provides a language for representing knowledge in the form of a certain data structure and controlling knowledge. The underlying knowledge representation of an expert system may
be one of various types, but the production-rule knowledge representation has been widely accepted by many researchers [7].

The IF/THEN construct of a production-rule is very similar to that of a program clause of logic programming. The control strategy for production-rules, backward chaining, is similar to the resolution of logic programming. However, a principal difference is the way negation is treated. One of typical characteristics of expert systems is that they provide a “consultation mode” for the purpose of the interaction with users [6]. Therefore, the interpretation of unproved facts is not necessarily in most expert systems. The deontic expert system of this paper treats negation as explicit negation, rather than negation as failure [11, 30]. Negated information is either given as facts, derived from rules, or asserted from the users via an interactive consultation mode.

The deontic expert system shell developed in this paper, as the name indicates, is to build applications in the domain of deontic worlds. Therefore, it is necessary to provide some closed (or pre-defined) vocabulary to capture deontic concepts, as well as closed vocabulary to construct IF/THEN rules and open vocabulary to capture entities and concepts in particular subject domains. The vocabulary for deontic concepts includes the notions of obligation, permission, forbearance, and waiver that are introduced and defined as modal operators in deontic logic. However, our principal departure from deontic logic is to regard these notions as predicates. In modal deontic logic, actions to which modal operators are applied are propositions. As in modal deontic logic, the arguments of deontic predicates are actions; however, there we take a somewhat different ontological perspective in regarding actions as logical individuals rather than propositions.

In brief, the deontic expert system in this paper is an interactive rule-based system with deontic reasoning based on general program clause logic programming without the negation as failure rule [11, 30].

3. DX: A DEONTIC EXPERT SYSTEM SHELL

We operationalize the concepts of deontic logic programming and expert system in the form of a rule-based deontic expert system shell, called DX. We mainly describe the syntax and operational mode of DX, together with several examples.

3.1. Syntax of DX

The basic syntax of DX is based on the IF/THEN structure common to expert systems. The rule structure for DX is

\( \langle \text{condition} \rangle \text{IF} \langle \text{conditions} \rangle . \)

or equivalently,

\( \text{IF} \langle \text{conditions} \rangle \text{THEN} \langle \text{condition} \rangle . \)
Notice that the period at the end of the above expressions is a part of the syntactic structure. Rules may also be unconditional, in which case they are expressed as the predicate ⟨condition⟩ by itself without the IF/THEN. The ⟨conditions⟩ may include the connectives AND and OR, where AND has the more immediate binding. When a rule includes OR, it may be decomposed to rules that do not include OR. That is, a rule:

⟨condition⟩ IF ⟨condition1⟩ OR ⟨condition2⟩.

is equivalent to rules:

⟨condition⟩ IF ⟨condition1⟩.
⟨condition⟩ IF ⟨condition2⟩.

A ⟨condition⟩ may be a simple predicate expression, e.g. \texttt{STAFF(X)}, \texttt{DEPT-OF(X,Y)}. In DX, these predicate names are regarded as part of the open vocabulary, introduced as appropriate for the problem domain. Also, there are five primitive deontic conditions that have special interpretations in the system:

\begin{verbatim}
OBLIG( ⟨action⟩ )
PERMIT( ⟨action⟩ )
FORBID( ⟨action⟩ )
WAIVE( ⟨action⟩ )
COMMIT( ⟨action⟩, ⟨action⟩ )
\end{verbatim}

Additionally, a condition may be negated, using the symbol ‘~’. In the DX rule interpreter, since negation is treated as explicit negation, if a condition cannot be proven, the interpreter will ask the user whether it is true or false. The ⟨action⟩ is represented as:

⟨agent⟩ : ⟨condition⟩

read that ⟨agent⟩ brings about ⟨condition⟩. An ⟨agent⟩ is an expression of term, which is either a constant, variable, or a function. These terms are regarded as part of the open vocabulary as predicates for conditions are. The combined syntax for the DX rules is shown in Figure 1.

Note that deontic conditions, e.g. \texttt{PERMIT( ⟨action⟩ )}, etc., themselves may be qualified as conditions. Thus, this notation allows the recursive expression of deontic predicates. For example,

\texttt{PERMIT(X:PERMIT(Y:PERMIT(Z:A)))}

is read that “\texttt{X} is permitted to bring about the permission of \texttt{Y} to bring about the permission of \texttt{Z} to bring about the condition \texttt{A}.”

---

DX: A Deontic Expert System

November 21, 1994
Facts may also be included in rule-base as predicates. The difference between facts and unconditional rules is that facts are ground, e.g. predicates without variables. These ground predicates are also regarded as part of the open vocabulary dependent on the problem domain.

Another type of assertion is sometimes included in the rule-base, to specify that certain conditions are mutually exclusive. This enables the system to infer the truth values of certain predicates without having to ask. The format is

\[ \text{EXCLUSIVE}(\{ \text{list-of-conditions} \}) \]

where

\[
\text{list-of-conditions} := \text{condition} \\
\text{list-of-conditions} := \text{condition}, \text{list-of-conditions}
\]

For example,

\[ \text{EXCLUSIVE}([\text{STUDENT}(X), \text{STAFF}(X), \text{FACULTY}(X)]) \]

states that “\(X\) is exclusively either a student, a staff member, or a faculty member.”

A comment on the notation: for readability, both predicate constants and variables are written in upper case, variables being just a single letter while predicate constants are longer words. Individual constants begin with a capital letter followed by lower case letters or digits. For example,

\[ \text{FACULTY}(X) \]
\[ \text{CONTRACT}(\text{Smith}, \text{Jones}) \]

3.2. Implementation of DX

DX consists of two components, interpreter and dialog, and provides data structures for rules and facts as defined in the previous section. The interpreter consists of a knowledge control strategy, which is based on the resolution of deontic logic programming with explicit negation, and a knowledge base modifier that allows the users to update the knowledge base of a DX application.

The dialog of DX provides four types of user commands:

- **Query** — Prove a condition.
- **Assess** — Find the deontic status of a given action.
- **Update** — Change a condition through the performance of an action.
- **Search** — Find courses of action to achieve a specified condition.

**Query** is a command that invokes the DX knowledge control strategy to prove a given condition. The **Assess** command finds deontic conditions (obligation, permission, prohibition, and/or waiver) of a given action through the knowledge control strategy. DX conditions are
modified by the **Update** command. The **Update** command asserts or retracts facts as results of actions. **Search** is a command that repeatedly applies the knowledge control strategy to find one or more courses of action that can achieve a given goal.

The dialog of DX also provides an interface that asks the users to enter a truth value of an unknown condition. Note that in a procedure of the DX knowledge control, truth values of conditions are explicitly determined by facts, rules, or assertions through the window; thus no unproved conditions appear in the procedure.

### 3.3. DX Query Procedures

The **Query** command in DX performs factual deductions much like Prolog. That is, DX extends the standard backward chaining mechanism of expert systems by allowing variables in expressions. Given a query $\phi$, the DX query processor maintains a state:

$$\sigma_0 = \langle (\phi), \emptyset \rangle$$

where $\emptyset$ is an empty substitution. When a rule:

$$\phi_j \text{ IF } \psi_1 \text{ AND } \psi_2 \text{ AND } \ldots \text{ AND } \psi_n$$

is in the rule-base such that:

$$\phi_\theta_1 = \phi_j, \theta_1$$

where $\theta_1$ is a substitution:

$$\theta_1 = \{ x_1 = t_1, x_2 = t_2, \ldots, x_k = t_k \}$$

where $x_i$ ($i = 1, 2, \ldots, k$) are variables in $\phi$ and $\phi_j$, and $t_j$ ($j = 1, 2, \ldots, k$) are terms in $\phi$ and $\phi_j$ such that $x_i \neq t_j$ (for all $i$ and $j$), the DX query processor transforms $\sigma_0$ to:

$$\sigma_1 = \langle (\psi_1, \psi_2, \ldots, \psi_n), \theta_1 \rangle.$$  

The truth value of an unknown fact is interactively given by the user of DX. Further transformations of states by the DX query processor may result in:

$$\sigma_m = \langle (), \theta_1, \theta_2, \ldots, \theta_m \rangle$$

where $\theta_1, \theta_2, \ldots, \theta_m$ is a composition of substitutions. Then, the DX query processor gives $\theta_1, \theta_2, \ldots, \theta_m$ as an answer. When no such state $\sigma_m$ with the empty list of literals is reachable and the rule-base contains another rule:

$$\phi_j \text{ IF } \pi_1 \text{ AND } \pi_2 \text{ AND } \ldots \text{ AND } \pi_n$$

where $\phi$ and $\phi_j$ are unifiable, that is:

$$\phi_\tau_1 = \phi_j, \tau_1,$$

the DX query processor backtracks the state transformation and continues the state transformation. The DX query procedures repeat until no further state transformation results in:
In a state:

$$\sigma_m = \langle(),\ldots\rangle.$$ 

where $$\Theta = \theta_1\theta_2\cdots \theta_\mu$$, if there exists no rule whose consequence is unifiable with $$\psi_1$$, the DX query processor asks users about its truth value.

As an example problem area, we consider deontic rules relating to the management of the information systems department of a university. Every faculty member in this department teaches computers. This is expressed in the following rule and query. (See Figure 2.)

query: \text{TEACHES-COMPUTER}(X)

rule: \text{TEACHES-COMPUTER}(X) \text{ IF} \\
\text{FACULTY}(X) \text{ AND} \\
\text{DEPT-OF}(X, \text{Information-Systems}).

fact: \text{FACULTY}(\text{Lee}).

fact: \text{DEPT-OF}(\text{Lee}, \text{Information-Systems}).

response: $$X = \text{Lee}$$
Unlike Prolog, all negation is explicit. For example, we may have a rule that all faculty not on leave are on the payroll:

query: \( \text{ON-PAYROLL}(X) \)
rule: \( \text{ON-PAYROLL}(X) \) IF \\
\( \text{FACULTY}(X) \) AND \\
\( \neg \text{ON-LEAVE}(X) \).

fact: \( \text{FACULTY}(\text{Lee}) \).
fact: \( \neg \text{ON-LEAVE}(\text{Lee}) \).
fact: \( \text{FACULTY}(\text{Elam}) \).
fact: \( \text{ON-LEAVE}(\text{Elam}) \).
response: \( X = \text{Lee} \)

3.4. Deontic Deduction

The Assess command is used to determine the deontic status of a specific action, following the computing process for deontic reasoning in Section 2.2. When an Assess command of an action, say \( X: \text{DO-SOMETHING} \), is given, DX tries to prove \( \text{OBLIG}(X: \text{DO-SOMETHING}) \) OR \( \text{PERMIT}(X: \text{DO-SOMETHING}) \) OR \( \text{FORBID}(X: \text{DO-SOMETHING}) \) OR \( \text{WAIVE}(X: \text{DO-SOMETHING}) \). For example, we will examine a possible set of deontic rules for controlling access to the departmental photocopy machine.

assess: \( \text{Lee:USE-COPIER} \)
rule: \( \text{PERMIT}(X: \text{USE-COPIER}) \) IF \\
\( \text{FACULTY}(X) \).
fact: \( \text{FACULTY}(\text{Lee}) \).
response: \( \text{PERMIT}(\text{Lee:USE-COPIER}) \)

Students may use the copier if they are teaching assistants. (See Figure 3.)

assess: \( \text{Chen:USE-COPIER} \)
rule: \( \text{PERMIT}(X: \text{USE-COPIER}) \) IF \\
\( \text{STUDENT}(X) \) AND \\
\( \text{TEACHING-ASSISTANT}(X) \).
fact: \( \text{STUDENT}(\text{Chen}) \).
fact: \( \text{TEACHING-ASSISTANT}(\text{Chen}) \).
response: \( \text{PERMIT}(\text{Chen:USE-COPIER}) \)
Staff may use the copier if they have special copy privileges.

assess: Fletcher:USE-COPIER
rule: PERMIT(X:USE-COPIER) IF
      STAFF(X) AND
      COPY-PRIVILEGE(X).
fact: STAFF(Fletcher).
fact: COPY-PRIVILEGE(Fletcher).
response: PERMIT(Fletcher:USE-COPIER)

3.5. DX Update Procedures: Changing Deontic Constraints

The Update command in DX changes deontic constraints by performing an action. These changes may be physical, such as a change in location, or they may be changes in non-physical conditions, e.g. becoming a student, getting married. We refer to the latter as *deontic status conditions*. The kinds of conditions appearing in deontic rules are mainly of this type; all previous examples have been this type. When such deontic status conditions change, new types of deontic conclusions apply. Physical conditions may change due to natural causes, e.g. aging, the weather, or as the result of some human action, e.g. driving a car. Deontic status conditions, by contrast, are only changed by some human action. In DX, only human actions are considered.

In order to support changing deontic constraints, DX uses the following rule

\[
\text{rule} := \text{FROM } \langle \text{conditions-1} \rangle \text{ IF } \langle \text{action} \rangle \text{ TO } \langle \text{conditions-2} \rangle.
\]

It means if \langle \text{conditions-1} \rangle are satisfied and \langle \text{action} \rangle is permitted, the performance of \langle \text{action} \rangle asserts \langle \text{conditions-2} \rangle. DX uses the Query processor to check if \langle \text{conditions-1} \rangle are satisfied and the Access processor to check if \langle \text{action} \rangle is permitted; and then asserts the \langle \text{conditions-2} \rangle.

For example, the chairperson may grant copier privileges to staff.

update: Dyer:COPY-PRIVILEGE(Brown)
rule A: PERMIT(X:COPY-PRIVILEGE(Y)) IF
       CHAIR(X) AND
       STAFF(Y).
rule B: FROM TRUE
       IF X:COPY-PRIVILEGE(Y)
       TO PERMIT(Y:USE-COPIER).
fact: CHAIR(Dyer).
fact: STAFF(Brown).
response: UPDATE ACCEPTED
When the Update command of $\text{Dyer:COPY-PRIVILEGE(Brown)}$ is issued, DX first checks if it is permitted. Since $\text{Dyer}$ is a chair and $\text{Brown}$ is a staff member, $\text{Dyer:COPY-PRIVILEGE(Brown)}$ is permitted by rule A. As the result, $\text{PERMIT(Brown:USE-COPIER)}$ is asserted by rule B.

assess: $\text{Brown:USE-COPIER}$  
response: $\text{PERMIT(Brown:USE-COPIER)}$

Due to the previous Update, the Assess to $\text{Brown:USE-COPIER}$ is answered ‘permitted.’

update: $\text{Lee:COPY-PRIVILEGE(Quintus)}$  
response: $\text{UPDATE REJECTED}$

However, the Update command of $\text{Lee:COPY-PRIVILEGE(Quintus)}$ is denied because its permission is not granted by rule A.

3.6. DX Search Procedures: Goal Seeking with Deontic Constraints

When a given condition is not directly permitted by the deontic rules, it may nonetheless be achievable by a certain sequence of deontic actions. The command Search will find these various action sequences. For example, suppose that Quintus, who is on the staff, is not presently permitted to use the copier.

search: $\text{PERMIT(Quintus:USE-COPIER)}$  
action: $\text{Dyer:COPY-PRIVILEGE(Quintus)}$.

In this case, the system has suggested that the department chair, Dyer, grant Quintus copy privileges. The explanation is as follows. (See Figure 4.)

| rule A: | $\text{PERMIT(X:USE-COPIER)}$ IF  
|        | $\text{STAFF(X) AND}$  
|        | $\text{COPY-PRIVILEGE-GRANTED(X)}$. |

| rule B: | $\text{FROM TRUE}$  
|        | $\text{IF Y:COPY-PRIVILEGE(X)}$  
|        | $\text{TO COPY-PRIVILEGE-GRANTED(X)}$. |

| rule C: | $\text{PERMIT(X:COPY-PRIVILEGE(Y)) IF}$  
|        | $\text{CHAIR(X) AND}$  
|        | $\text{STAFF(Y)}$. |

| fact:  | $\text{STAFF(Quintus)}$. |

| fact:  | $\text{CHAIR(Dyer)}$. |

| therefore: | $\text{PERMIT Dyer:COPY-PRIVILEGE(Quintus)}$ |

| action:  | $\text{Dyer:COPY-PRIVILEGE(Quintus)}$ |

| therefore: | $\text{PERMIT Quintus:USE-COPIER}$ |
To achieve the goal seeking with deontic constraints, we adopt a simple search processor. Given a goal $\gamma$ by a Search command, the DX Search processor maintains a state:

$$\sigma_0 = \langle (\gamma), () \rangle$$

If there exists a rule:

$$\text{FROM } \tau_1 \text{ AND } \ldots \text{ AND } \tau_n \text{ IF } \alpha_1 \text{ TO } \gamma_1 \text{ AND } \ldots \text{ AND } \gamma_k$$

where $\gamma \theta_1 = \gamma, \theta_1$ ($\theta_1$ is an substitution) for some $i$, a new state is obtained as follows:

$$\sigma_1 = \langle (\tau_1 \theta_1, \ldots, \tau_n \theta_1, \text{PERMIT}(\alpha_1 \theta_1)), (\alpha_1 \theta_1) \rangle$$

If $\tau_1 \theta_1$ is given as a fact, the following state is obtained:

$$\sigma_2 = \langle (\tau_2 \theta_1, \ldots, \tau_n \theta_1, \text{PERMIT}(\alpha_1 \theta_1)), (\alpha_1 \theta_1) \rangle$$

Otherwise if there is a rule:

$$\tau \text{ IF } \tau'_1 \text{ AND } \ldots \text{ AND } \tau''_m$$

where $\tau_1 \theta_1 = \tau_2 \theta_1$, then the state is transformed to:

$$\sigma_2 = \langle (\tau'_1 \theta_2, \ldots, \tau''_m \theta_2, \tau_2 \theta_1 \theta_2, \ldots, \tau_n \theta_1 \theta_2, \text{PERMIT}(\alpha_1 \theta_1 \theta_2)), (\alpha_1 \theta_1 \theta_2) \rangle$$

Otherwise, the DX Search processor finds another rule:

$$\text{FROM } \upsilon_1 \text{ AND } \ldots \text{ AND } \upsilon_m \text{ IF } \alpha_2 \text{ TO } \delta_1 \text{ AND } \ldots \text{ AND } \delta_l$$

where $\tau_1 \theta_1 \theta_2 = \delta_1 \theta_2$ for some $i$, and transforms $\sigma_1$ to:

$$\sigma_2 = \langle (\upsilon_1 \theta_2, \ldots, \upsilon_m \theta_2, \tau_2 \theta_1 \theta_2, \ldots, \tau_n \theta_1 \theta_2, \text{PERMIT}(\alpha_1 \theta_1 \theta_2), \text{PERMIT}(\alpha_2 \theta_2)), (\alpha_1 \theta_1 \theta_2, \alpha_2 \theta_2) \rangle$$

Further transformations of the state and refutations of PERMIT(---) by deontic reasoning in Section 2.2 may result in a state:

$$\sigma = \langle (), (\alpha_1 \Theta_1, \alpha_2 \Theta_2, \ldots) \rangle$$

where $\Theta_i$ (for all $i$) are compositions of substitutions. Then, the sequence of actions $\alpha_1 \Theta_1, \alpha_2 \Theta_2, \ldots$ is returned as an answer.

### 4. APPLICATIONS: PERFORMATIVE NETWORKS

As illustrated in the previous examples, typical applications for the deontic expert system are to represent bureaucratic rules and regulations within organizations. These would be especially beneficial in cases where the rules are complex and/or subject to frequent changes. In such bureaucratic organizations, the deontic expert system would be part of the organization’s computer network, thus ensuring that everyone had access to the same rule set.
Deontic expert systems may also be used to model inter-organizational activities and other activities in the society at large. Williamson [37, 38] offers a perspective of social organization in terms of governance structures. In this framework, organizations are one type of governance structure, based on the notion of a hierarchy where every individual has just one supervisor. However, for some organizations, where for instance production and marketing concerns compete for priority, a matrix kind of organization may be appropriate. For other types of resource allocation situations, a market structure may be the more suitable governance structure. For Williamson, this ‘markets vs. hierarchies’ trade-off is determined by transaction cost economics: markets have lower transaction costs for standardized types of contracting exchanges, whereas hierarchies have advantages when more specialized cooperation is needed between the parties.

The kinds of governance structures described by Williamson are distinguished by their number and scope: hierarchies consist of a single, monolithic governance structure, whereas markets comprise a large number of relatively small governance structures, i.e. the separate contracts. From our perspective, however, what all these governance structures have in common is that they are deontic systems; whether hierarchy or contract, the purpose is to regulate or control the behavior of certain individuals to achieve economic goals.

This suggests that the use of deontic expert systems might also have applications external to the organization, facilitating the contracting process. We refer to these types of applications as ‘electronic contracting’ [20, 21]. Unlike internal applications, where a computer network is often already in place for data processing applications, electronic contracting applications may require that a separate, inter-organizational network be established. Such networks are indeed the trend, however, as more and more companies develop electronic data interchange (EDI) applications. In terms of the kinds of data exchanged and the inferencing employed, EDI is rather like the external counterpart to data processing applications internally. A potential role of deontic expert systems is to extend EDI to include reasoning about the legal rules and regulations affecting commercial transactions.

A requirement of these applications, whether for bureaucratic rule-bases or electronic contracting, is the ability to actually perform actions via the network, a goal made possible because the principal actions of concern are linguistic rather than physical, namely deontic actions. These are what the philosopher Austin [4] called performative communications. In our earlier examples, the conferring of copy privileges and copy discretion were performative.

In electronic contracting, performative communications alter the legal status of the parties by conferring rights, imposing obligations, and prohibiting or waiving certain acts; that is, they change the deontic state of the parties. For example, issuing a letter of credit entitles the named beneficiary to payment upon compliance with the stated terms and conditions. Similarly, issuing a negotiable bill of lading gives the legal holder of the document title to
the goods if the document directs that they are to be delivered to the bearer or to the order of a named person.

The combination of these two aspects—performative communications and deontic states—leads to the concept of a performative network: a computer network accessible to individuals, households, companies and governmental agencies, providing the capability to perform deontic transactions. But what makes the network performative? This is not a technical feature, but rather a policy made about the network’s use, what we call a performative assumption. In essence, this assumption is that communications made through the network are legally (deontically) binding. In electronic contracting applications, the network obtains legal force through an umbrella contract, a membership agreement signed by all parties using the network. In bureaucratic applications, the performative assumption may be made as a matter of organizational policy.

An important advantage of performative networks is that the computer can act as a witness to deontic transactions. Assuming that all transactions are logged, performative networks can provide an audit trail of negotiations and contract performance, providing complete and exact evidence in cases of dispute. An even more significant advantage, however, is that performative networks can actually enforce deontic constraints. This was illustrated earlier with the Update command in DX: when the user attempts to make a deontic action for which he/she is not permitted, the system rejects the action. Similarly, a performative network for electronic contracting could prevent users from forming contracts that violate trade regulations.

Using the analog of the Search command in DX, a performative network could also aid users in finding ways to achieve their goals. For example, a performative network serving welfare recipients could help users find all the support services for which they are entitled. Conceivably, such a network could be much more thorough and helpful than the human staff that presently performs such functions, also helping to reduce the high overhead of these organizations.

5. FUTURE DIRECTIONS

The DX shell provides a basic model for computer-assisted deontic reasoning. However, to become useful for practical applications, a few extensions are needed. In this section, we present two directions of extensions: defeasible reasoning and temporal reasoning.

5.1. Deontic Dilemmas and Rule Precedence

Deontic dilemmas, or deontic conflicts, are often observed in normative systems and their resolution is important from a practical standpoint [16: p. vii]:

It has often been argued that unlike “natural” necessities, obligations can conflict with one another, and the resolution of such normative conflicts is an important part of moral discourse.
A deontic dilemma arises when two or more deontic rules imply conflicting conclusions for a given situation. A classical example is Kierkegaard’s analysis of Abraham’s dilemma whether or not to kill his son, Isaac. God had commanded Abraham to kill his son, yet ethical principles said he should not. We distinguish two categories of dilemmas, ‘mild’ and ‘deadlock.’ A mild dilemma occurs when deontic rules conflict, but there is still a course of action available that avoids violating either rule. There are two main cases:

\[
\text{FORBID}(X:A) \& \text{PERMIT}(X:A) \\
\text{OBLIG}(X:A) \& \text{WAIVE}(X:A)
\]

In the first case, where the same action is both forbidden and permitted, the party \(X\) may abstain from the forbidden action to avoid violation. In the second case, where the same action, \(A\), is both obliged and waived, the party \(X\) may do the obliged action to avoid violation. Deadlock dilemmas are more difficult. In these cases, the individual has no course of action that avoids a violation. The principal pattern of a deadlock dilemma is the following:

\[
\text{OBLIG}(X:A) \& \text{FORBID}(X:A).
\]

That is, an action \(A\) is both obliged and forbidden. Or more generally:

\[
\text{OBLIG}(X:A) \& \text{OBLIG}(X:B).
\]

where \(A\) and \(B\) are mutually exclusive actions, represented in DX with the assertion:

\[
\text{EXCLUSIVE}([A,B]).
\]

Example (Mild Dilemma). The university library has a policy that students with outstanding fines may not use copiers on campus.

assess: Chen:USE-COPIER  
rule: PERMIT (X:USE-COPIER) IF STUDENT (X).  
rule: FORBID (X:USE-COPIER) IF STUDENT (X) AND OUTSTANDING-FINE (X).  
fact: STUDENT (Chen).  
fact: OUTSTANDING-FINE (Chen).  
response: PERMIT (Chen:USE-COPIER)  
FORBID (Chen:USE-COPIER)

Example (Deadlock Dilemma). Teaching assistants must make use of the copier to prepare exam materials for class.

assess: Chen:USE-COPIER  
rule: OBLIG (X:USE-COPIER) IF TEACHING-ASSISTANT (X) AND
REQUESTED-COPYING-EXAM(X).

rule: FORBID(X:USE-COPIER) IF
      STUDENT(X) AND
      OUTSTANDING-FINE(X).

fact: TEACHING-ASSISTANT(Chen).

fact: REQUESTED-COPYING-EXAM(Chen).

response: OBLIG(Chen:USE-COPIER)
           FORBID(Chen:USE-COPIER)

One of the ways to systematically avoid deontic dilemmas is hierarchical reasoning of rules [1] by default or defeasible reasoning [10, 26, 27]. Often we indicate that certain deontic rules apply as defaults; when a more specific situation occurs, defaults are defeated. We are currently developing the DDX shell (i.e., defeasible deontic expert system shell), as an extension to the DX shell. We adopt Nute’s specificity-based defeasible reasoning [27] to establish and reason about hierarchies of rules.

5.2. Temporal Reasoning

One of key functions that is missing in the current implementation of DX is temporal inferencing. This, we believe has, a fundamental role in deontic reasoning. An aspect of temporal inferencing is the marking of transactions (i.e. actions) with the date and time when they were performed. Our notations for temporal markings are [22]:

(\text{action}) \text{ON} (\text{date})
(\text{action}) \text{BY} (\text{date})

The first notation is used to record the time when historical actions were performed. The second is to mark deontic constraints with a \textit{deadline} when the action is to be performed. Another aspect of temporal inferencing is the sequencing of events, whether one event precedes another, or whether they occur concurrently. To achieve this, we adopt Petri nets\textsuperscript{5} [28, 29], augmented with the action notation described above.

This is presently being developed as a generalization of the DX shell, called CASE/EDI. As its name suggests, CASE/EDI is designed to be a computer aided system engineering tool for developing prototype EDI models. A special feature of this system is the use of direct manipulation graphics for the input and display of the Petri nets. CASE/EDI allows graphical modeling of procedures and reasoning about their deontic status during their execution. It also allows the users to retrieve (or generate) procedures (i.e. courses of action) to achieve a certain goal.
6. CONCLUDING REMARKS

The DX shell, proposed in this paper, is a system for computer-assisted deontic reasoning. Its principal characteristics are as follows:

- DX reasoning is based on program clause logic programming.
- DX is an interactive system. The DX consultation mode includes a menu of accessible DX commands, through which the users query and update conditions.
- DX provides deontic deduction capabilities.

In Sections 2 and 3, we focused our attention on the syntactic aspect of DX, which includes the procedural/operational model of DX reasoning. The other aspect of logic modeling, not addressed in this paper, is semantics that includes the specification of the domain of DX, mapping syntactic components to those in the domain, and soundness/completeness issues of DX reasoning. Especially, issues of soundness and completeness are important in that they are a means of justifying correctness of DX reasoning. We believe DX reasoning is at least sound and complete with some restrictions, because it is founded on standard logic programming in which the resolution is proved to be sound and complete [3, 24]. However, the considerations of soundness and completeness of DX reasoning require further investigations; we will leave these as a future research issue.

In Section 5, we proposed extensions to DX in two directions: defeasible reasoning and temporal reasoning extensions. Another research issue to be considered addresses managing the evolution of the rule-base. For managing the evolution of the rule-base, we would like to model how deontic rules could be used as meta-rules to control how the rule-base is modified, for instance, a manager cannot give himself a promotion. Currently DX provides for changes in single conditions only. An extension is to allow for changes, additions, and deletions in the rules, too. That is, rather than modifying the rules using an ordinary text editor, all insertions, deletions, and changes would pass through deontic controls. The principal difficulty occurs when different authorities require mutually exclusive actions. Various heuristics, such as the relative power of the two authorities, the recency of the rule, and the preference of the users, may be used to decide which rule should take precedence.

All features of DX presented in this paper have been implemented in Prolog. The implementation in Macintosh® platform (with LPA™ Mac-Prolog 3.0) provides additional features of menu-driven graphical user interface. A sample DX rule-base of University of Texas General Library Lending Codes is currently being developed, to demonstrate the applicability of DX.
ACKNOWLEDGEMENT

The authors appreciate anonymous reviewers and the Editor-in-Chief of the *Journal of Management Information Systems* for useful comments, criticism, and suggestions that contributed to a significant improvement of the paper.

REFERENCES


26. Nute, D. A nonmonotonic logic based on conditional logic. Report No. 01-007, Advanced Computational Methods Center, University of Georgia, 1985


FOOTNOTES

1 (p. 4) For the proofs of theorems, refer to [17], [35], and [36].

2 (p. 6) See e.g. [9] or [33]. For a more general treatment of logic and artificial intelligence, see [13]. For a theoretical foundation of logic programming, see [3] and [24].

3 (p. 8) Here we use the Backus-Naur Form (BNF) commonly used in computer science for presenting programming language syntax. The angle brackets are non-terminal syntactic constituents. The ‘::=’ symbol is for syntactic definitions. All other symbols are terminal symbols in the DX rule syntax.

4 (p. 17) For implementations, functions, trends, and other issues of EDI, see [2], [6], [15], [18], and [32].

5 (p. 20) For variations of Petri nets and advanced issues, see [14], [19], [23], and [25].
Figure 1: BNF Syntax for DX Rules

\[
\begin{align*}
\langle \text{rule} \rangle &::= \langle \text{condition} \rangle. \\
\langle \text{rule} \rangle &::= \text{IF } \langle \text{conditions} \rangle \text{ THEN } \langle \text{condition} \rangle. \\
\langle \text{rule} \rangle &::= \langle \text{condition} \rangle \text{ IF } \langle \text{conditions} \rangle. \\
\langle \text{rule} \rangle &::= \text{FROM } \langle \text{conditions} \rangle \\
&\quad \text{ IF } \langle \text{action} \rangle \\
&\quad \text{ TO } \langle \text{conditions} \rangle. \\
\langle \text{conditions} \rangle &::= \langle \text{condition} \rangle \\
\langle \text{conditions} \rangle &::= \langle \text{condition} \rangle \text{ AND } \langle \text{conditions} \rangle \\
\langle \text{conditions} \rangle &::= \langle \text{condition} \rangle \text{ OR } \langle \text{conditions} \rangle \\
\langle \text{condition} \rangle &::= \langle \text{predicate} \rangle \\
\langle \text{condition} \rangle &::= \text{OBLIG}(\langle \text{action} \rangle) \\
\langle \text{condition} \rangle &::= \text{PERMIT}(\langle \text{action} \rangle) \\
\langle \text{condition} \rangle &::= \text{FORBID}(\langle \text{action} \rangle) \\
\langle \text{condition} \rangle &::= \text{WAIVE}(\langle \text{action} \rangle) \\
\langle \text{condition} \rangle &::= \text{COMMIT}(\langle \text{action} \rangle, \langle \text{action} \rangle) \\
\langle \text{condition} \rangle &::= \sim \langle \text{condition} \rangle \\
\langle \text{action} \rangle &::= \langle \text{agent} \rangle : \langle \text{condition} \rangle \\
\langle \text{agent} \rangle &::= \langle \text{term} \rangle
\end{align*}
\]
Figure 2. A Sample **Query** Session
(a) DX Menu

<table>
<thead>
<tr>
<th>DX Wins</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Syntax check switch</td>
<td>✔</td>
</tr>
<tr>
<td>Compile</td>
<td></td>
</tr>
<tr>
<td>Query</td>
<td>✔Y</td>
</tr>
<tr>
<td>Assess</td>
<td>✔A</td>
</tr>
<tr>
<td>Update</td>
<td>✔U</td>
</tr>
<tr>
<td>Search</td>
<td></td>
</tr>
</tbody>
</table>
Figure 2. A Sample Query Session
(b) The Query option lists templates of all available Query terms.

Select goal condition:

Select:

\text{teaches\_computer(’H’)}
Figure 2. A Sample Query Session
(c) A specific Query term is generated from the selected template.
(Here, ‘X’ in the template is replaced by a variable X.)
Figure 2. A Sample Query Session
(d) The Output Screen

Σ Output Window

`teaches_computer(Lee) is true because ...
RULE: teaches_computer(X) if
    faculty(X) and dept_of(X, Information Systems).
FACT: faculty(Lee).
THEREFORE, teaches_computer(Lee).`
Figure 3. A Sample Assess Session
(a) The **Assess** option lists templates of all available **Assess** terms.

<table>
<thead>
<tr>
<th>Enter action to be assessed:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Select:</strong></td>
</tr>
<tr>
<td>'H':<strong>use_copyer</strong></td>
</tr>
<tr>
<td>'H':<strong>copy_privilege('Y')</strong></td>
</tr>
</tbody>
</table>
Figure 3. A Sample Assess Session
(b) A specific Assess term is generated from the selected template. (Here, ‘X’ in the template is replaced by a constant ‘Chen’.)
Figure 3. A Sample **Assess** Session

(c) DX asks the user to enter a truth value of an unknown condition.

![Image showing a dialog box asking if teaching_assistant(Chen) is true with options: Yes, No, Abort.](image)
Figure 3. A Sample Assess Session
(d) The Output Screen

Σ Output Window

ACTION Chen: use_copyer IS PERMITTED BECAUSE ...
RULE: permit(X: use_copyer) if
  student(X) and teaching_assistant(X).
FACT: student(Chen).
FACT: teaching_assistant(Chen).
THEREFORE, permit Chen: use_copyer.

No (more) solutions.
Figure 4. A Sample Search Procedure

Search: \textsc{PERMIT(Quintus:USE-COPIER)} by rule A

\textsc{STAFF(Quintus) ?}

Yes

\textsc{COPY-PRIVILEGE-GRANTED(Quintus) ?}

No

Check rule B

Update: \textsc{Dyer:COPY-PRIVILEGE(Quintus)} by rule C

Permitted

Action: \textsc{Dyer:COPY-PRIVILEGE(Quintus)} by rule B

\textsc{COPY-PRIVILEGE-GRANTED(Quintus) ?}

Yes