ICOT Technical Report: TR-0866

TR-0866

A Legal Reasoning System on a Deductive Object-Oriented Database

by
C. Takahashi (JIPDEC) & K. Yokota

March, 1994

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191~5

Institute for New Generation Computer Technology

A Legal Reasoning System on a Deductive Object-Oriented Database

Chie Takahashi
JIPDEC *

Kazumasa Yokota
ICOT †

Abstract

A legal reasoning system is a large-scale knowledge information processing system into which many technologies such as artificial intelligence, natural language processing, and databases are integrated. From a database point of view, this application features many kinds of data and knowledge, and provides many research topics for next generation databases: features of very large databases and knowledge-bases, their classification, treatment of partial information, query processing containing high-level reasoning, and so on. Further, it suggests ideas for the boundaries between databases and applications. In this paper, we explain our experimental legal reasoning system that is based on the deductive object-oriented database system QUIXOTE, and show how the effectiveness of the extended features for next generation databases.

1 Introduction

Recently, legal reasoning has attracted much attention from researchers in the field of artificial intelligence, with great expectations for its *big* application. Legal reasoning systems are very important applications, whose development, like that of theorem provers, dates back to before artificial intelligence was proposed, (for example, see [3]). In fact, laws are related not only to the judicial world but also to all social activities. To support legal interpretation and reasoning in a wide range of situations, many systems have been developed, including those capable of planning tax-saving strategies, negotiation of payment of damages, making contract documents, predicting judgements and supporting legislation. Many works on expert systems for such applications have been published, while powerful legal database systems have not yet been reported.

In the Japanese FGCS (Fifth Generation Computer System) project, legal reasoning systems were considered quite critical and two prototype legal reasoning systems were developed: HELIC-II [5] and TRIAL [6, 11, 9].

For the above systems, we provide database and knowledge-base management facilities: QUIXOTE[10] and Kappa-P [9]. QUIXOTE is a deductive object-oriented database (DOOD) language and knowledge representation language, used for describing and classifying complex legal data and knowledge, while Kappa-P is a parallel nested relational database management system, used to store large volumes of legal data. Especially, all data and knowledge in TRIAL is written in QUIXOTE and some advanced query processing facilities, such as hypothetical reasoning and hypothesis generation (abductive reasoning), are provided for legal reasoning by QUIXOTE.

In this paper, we report on the experimental system, TRIAL, and illustrate the effectiveness of the advanced features of the DOOD system. Based on our practical experience, we discuss the roles we expect databases to play in legal applications, and the features that should be provided for next generation databases. This paper presents new applications of DOOD languages to knowledge information processing by presenting an overview of the TRIAL system in QUIXOTE as an example and discussing the features that will be required by next generation database systems. In Section 2, we briefly explain the kinds of features needed for legal reasoning. By way of example, we consider the case of worker's compensation law. In Section 3, we describe some of the features of QUIXOTE, based on the above example. In Section 4, we consider the entire example database used in TRIAL. Lastly, we discuss the features demanded of next generation databases, especially those derived from legal applications.

2 Legal Reasoning

2.1 Basic Model

The analytical legal reasoning process is considered as consisting of three steps: *fact finding*, *statutory interpretation*, and *statutory application*. Although fact finding is very important as a starting point, it is beyond the

*Japan Information Processing Development Center (JIPDEC), 3-5-8, Shibakoen, Minato-ku, Tokyo 108, JAPAN, e-mail: j-takaha@icot.or.jp.

†Institute for New Generation Computer Technology (ICOT), 21F., Mita-Kokusai Bldg., 1-4-28, Mita, Minato-ku, Tokyo 108, JAPAN, e-mail: kyokota@icot.or.jp.

capabilities of current technologies. So, we assume new cases to already be represented in an appropriate form for our system. Statutory interpretation is a particularly interesting theme from an artificial intelligence point of view. Our legal reasoning system, TRIAL, focuses on statutory interpretation as well as statutory application.

Although there are many approaches to statutory interpretation, we follow the procedure below:

- *analogy detection*
  Given a new case, similar precedents to that case are retrieved from an existing precedent database.
- *rule transformation*
  Precedents (interpretation rules), extracted by analogy detection, are abstracted until the new case can be applied to them.
- *deductive reasoning*
  Apply the new case, in a deductive manner, to abstract interpretation rules transformed by rule transformation. This step may include statutory application because it is used in the same manner.

Among these steps, a strategy enabling analogy detection is essential to legal reasoning for *more efficient* detection of *better* precedents, which ultimately determines the quality of the results of legal reasoning. As the primary objective of TRIAL is to investigate the possibilities of *Quixote* in this area and develop a prototype system, we focus only on a small target. That is, to what extent should interpretation rules be abstracted for a new case, to get an answer with a plausible explanation, but not for a general abstraction mechanism.

2.2 Example

In this paper, we consider a simplified example related to "*karōshi*" (death from overwork) to discuss the applicability of *Quixote* to legal reasoning. A new case, *new-case*, is as follows:

> *Mary*, a *driver*, employed by a company, "*S*", died from a *heart-attack* while taking an *intermission* between jobs. Can this case be applied to the *worker's compensation law*?

3 Features of *Quixote*

Quixote is based on several concepts: object identity, subsumption relation, subsumption constraint, property inheritance, module, submodule relation, and rule inheritance. From a database point of view, the language is a DOOD language while, from a logic programming point of view, it is thought of as an extended constraint logic programming language based on subsumption constraints. There are some differences from the new F-logic[4, 2]: the representation of object identity, the introduction of subsumption constraints, update semantics, and query processing. In this section, we explain some of its features, used in the above example. See the details of *Quixote* in [9, 8, 10].

3.1 Object Identity and Subsumption Relation

Objects in *Quixote* are identified by extended terms called *object terms*, that correspond to object identifiers (oids). An object term consisting of an atomic symbol is classified as *basic*, while an object term in the form of a tuple is referred to as *complex*. In the above example, *mary*, *driver*, *heart-attack* and *intermission* are basic, while $org[name = \text{"}S\text{"}]$ is complex. Generally, an object term is a variable or a term having the following form:

$$o[l_1 = t_1, \cdots, l_n = t_n] \ (0 \leq n)$$

where $o, l_1, \cdots, l_n$ are basic and $t_1, \cdots, t_n$ are object terms. $l_1, \cdots, l_n$ are called *labels*.

Object terms are related to each other by a *subsumption relation* (a kind of is_a relation). Given partial order between the basic object terms, it is extended between complex object terms as usual:

$$\begin{aligned} org[name = \text{"}S\text{"}] &\sqsubseteq org. \\ org[name = X, president = Y] &\sqsubseteq org[name = X, president = X] \end{aligned}$$

As the construction of a lattice from a partially ordered set, like that in [1], is well-known, we can assume that a set of object terms with *top* and *bot* constitutes a lattice, without losing generality. The meet and join operations of o_1 and o_2 are denoted by $o_1 \downarrow o_2$ and $o_1 \uparrow o_2$, respectively.

3.2 Subsumption Constraints and Property Inheritance

The property of an object is represented as a subsumption constraint. The pair constituted by an object term o and a label l, denoted $o.l$, is called a *dotted term*, which plays the role of a variable ranging over the domain of the object terms. Furthermore, a pair constituted by a dotted term and a label is, itself, also a dotted term. If t_1, t_2 is an object term or dotted term, then a *subsumption constraint* is defined as follows:

$$t_1 \sqsubseteq t_2.$$

That is, a *property* of an object o is a subsumption constraint with a dotted term starting with o. In other words, it is defined as a triple constituted by a label, a subsumption relation, and a value. For example, the *result* of the *new-case* is represented by the following subsumption constraint: $new\text{-}case.result \cong heart\text{-}attack$.

The syntactic construct for representing an object term with subsumption constraints is called an *attribute term*. Let o be an object term, C a set of subsumption constraints, then $o|C$ is an attribute term. For example, the following attribute term represents that the *new-case* is that *Mary* died from a *heart-attack* while taking an *intermission*:

$$\begin{aligned} new\text{-}case|\{&new\text{-}case.who \cong mary,\\ &new\text{-}case.while \cong intermission,\\ &new\text{-}case.result \cong heart\text{-}attack\} \end{aligned}$$

There are some syntax sugars in QUIXOTE:

$$o|\{o.l \sqsubseteq t\} \Leftrightarrow o/[l \rightarrow t] \quad o|\{o.l \sqsupseteq t\} \Leftrightarrow o/[l \leftarrow t] \quad o|\{o.l \cong t\} \Leftrightarrow o/[l = t]$$

Thus, the above attribute term can be represented as follows:

$$new\text{-}case/[who = mary, while = intermission, result = heart\text{-}attack]$$

Notice that there are two kinds of properties: those that appear in object terms and those as subsumption constraints. The former are *intrinsic* for an object, while the latter are *extrinsic* for an object. Regarding the extrinsic properties, the following constraint solver is applied to a set of subsumption constraints:

$$\begin{array}{rcl} o = o & \Longrightarrow & \text{eliminated} \\ o_1 \sqsupseteq o_2 & \Longrightarrow & o_2 \sqsubseteq o_1 \\ o_1 \sqsubseteq o_2, o_1 \sqsubseteq o_3 & \Longrightarrow & o_1 \sqsubseteq o_2 \downarrow o_3 \end{array} \qquad \begin{array}{rcl} o_1 \sqsubseteq o_2, o_2 \sqsubseteq o_3 & \Longrightarrow & o_1 \sqsubseteq o_3 \\ o_1 \sqsubseteq o_3, o_2 \sqsubseteq o_3 & \Longrightarrow & o_1 \uparrow o_2 \sqsubseteq o_3 \end{array}$$

Any set of subsumption constraints will produce a unique solution by applying the above rules[7].

It is natural to assume that extrinsic properties are inherited by object terms with respect to $\sqsubseteq$-ordering. Consider the following example:

$$\begin{array}{c} myocardial\text{-}infarction \sqsubseteq heart\text{-}attack \\ heart\text{-}attack/[arteriosclerosis \rightarrow yes] \end{array}$$

Since *myocardial-infarction* is a kind of *heart-attack* and *heart-attack* has a property $[arteriosclerosis \rightarrow yes]$, *myocardial-infarction* has the same property by inheritance from *heart-attack*.

Property inheritance between objects is defined by the following rule:

$$\begin{array}{rcl} o_1 \sqsubseteq o_2 & \Longrightarrow & o_1.l \sqsubseteq o_2.l \\ o_1[l = t] & \Longrightarrow & o_1.l = t \end{array}$$

According to the rules, we obtain downward and upward inheritance, multiple inheritance, and exception as follows:

$$\begin{array}{rcl} o_1 \sqsubseteq o_2, o_2/[l \rightarrow t] & \Longrightarrow & o_1/[l \rightarrow t] \\ o_1 \sqsubseteq o_2, o_1/[l \leftarrow t] & \Longrightarrow & o_2/[l \leftarrow t] \\ o_1 \sqsubseteq o_2, o_2 \sqsubseteq o_3, o_2/[l = t] & \Longrightarrow & o_1/[l \rightarrow t], o_3/[l \leftarrow t] \\ o_1 \sqsubseteq o_2, o_1 \sqsubseteq o_3, o_2/[l \rightarrow t_2], o_3/[l \rightarrow t_3] & \Longrightarrow & o_1/[l \rightarrow t_1 \downarrow t_2] \\ o_1 \sqsupseteq o_2, o_1 \sqsupseteq o_3, o_2/[l \leftarrow t_1], o_3/[l \leftarrow t_2] & \Longrightarrow & o_1/[l \leftarrow t_1 \uparrow t_2] \\ o_1/[l \rightarrow t_2] & \Longrightarrow & o_1[l = t_1]/[l = t_1] \end{array}$$

3.3 Rule and Module

A rule is defined as in constraint logic programming, as follows:

$$a_0 \Leftarrow a_1, \cdots, a_n || D$$

where $a_0, a_1, \cdots, a_n$ are attribute terms and D is a set of subsumption constraints. a_0 is called a *head*, $a_1, \cdots, a_n, D$ is called a *body*, and a_i is called a *subgoal*. A rule means that if the body is satisfied then the head is satisfied. If a body is empty, then the rule is called a *fact*.

For example, the following is a rule for judgement.

$$\begin{aligned} judge[case = X]/[judge \rightarrow insurance] \\ \Leftarrow judge[case = X]/[judge \rightarrow job\text{-}causality], \\ judge[case = X]/[judge \rightarrow job\text{-}execution] \\ ||\{X \sqsubseteq case\}. \end{aligned}$$

It means that if the judgement of some case, $judge[case = X]$ where $X \sqsubseteq case$, is *job-causality* and *job-execution*, then the judgement is *insurance*.

A module corresponds to a part of the world (situation) or a local database. The module concepts play an important role in classifying knowledge, modularizing a program or a database, assumption-based reasoning, and dealing with any inconsistency in a database in $\mathcal{QUIXOTE}$.

A *module* is defined as a set of rules as follows:

$$m :: \{r_1, \cdots, r_n\}$$

where m is an object term called a *module identifer* (mid) and $r_1, \cdots, r_n$ are rules. m is sometimes used instead of a module itself, if there is no confusion.

The definition of rules is extended for external reference of objects:

$$m_0 :: \{a_0 \Leftarrow m_1 : a_1, \cdots, m_n : a_n || D\}$$

where $m_0, m_1, \cdots, m_n$ are mids. It means that the module m_0 has a rule such that if a_i and D are satisfied in the module m_i for all $1 \leq i \leq n$, then a_0 is satisfied in the module m_0. As an attribute term can be separated into an object term and a set of constraints, the rule can be rewritten as follows:

$$m_0 :: \{o_0 | C_0 \Leftarrow m_1 : o_1, \cdots, m_n : o_n || C\}$$

where $a_i = o_i | C_i (0 \leq i \leq n)$ and $C = C_1 \cup \cdots \cup C_n \cup D$.

Importing and exporting rules are done by *rule inheritance*, defined in terms of the binary relation (written $\sqsupseteq_S$) between modules, called a *submodule relation* as follows:

$$m_1 \sqsupseteq_S m_2, \;\; m_1 :: R_1, \;\; m_2 :: R_2 \quad \Longleftrightarrow \quad m_1 :: R_1 \cup R_2, \;\; m_2 :: R_2$$

where m_1, m_2 are modules and R_1, R_2 are sets of rules. The right hand side of $\sqsupseteq_S$ in a submodule definition may be a formula of mids with set operations. For example, if we have

$$\begin{array}{llll} m_1 :: \{r_{11}, r_{12}, r_{13}\} & m_4 :: \{r_{41}, r_{42}\} & m_2 :: \{r_{21}, r_{22}\} & m_4 \sqsupseteq_S m_1 \cup m_3 \\ \{m_2, m_3\} :: r_{31} & m_5 \sqsupseteq_S m_2 \backslash m_3 & & \end{array}$$

then m_4 has $\{r_{11}, r_{12}, r_{13}, r_{31}, r_{41}, r_{42}\}$ and m_5 has $\{r_{21}, r_{22}\}$.

It is possible for inconsistent knowledge to co-exist by making use of the module mechanism. For example, consider that it cannot be said for certain whether *mary* has *arteriosclerosis*. The following shows how such a problem is handled:

$$\begin{aligned} new\text{-}case_1 :: mary/[arteriosclerosis \rightarrow yes]. \\ new\text{-}case_2 :: mary/[arteriosclerosis \rightarrow no]. \end{aligned}$$

where $new\text{-}case_1$ and $new\text{-}case_2$ are not related by submodule relation.

A *database* or a *program* is defined as the triple (S, M, R) of a finite set of subsumption relations S, a set of submodule relations M, and a set of rules R.

3.4 Query Processing

Query processing basically corresponds to resolution and constraint solving in constraint logic programming. One of the main features of data and knowledge in knowledge information processing, such as legal reasoning, is that the information is partial, that is to say, sufficient information need not necessarily be given. For example, a new case might lack some important facts. So, query and an answer is extended for treating partial information.

A *query* is defined as the pair (A, P) (written $?\text{-}A;;P$) of a set of attribute terms A and a program P, where A is referred to as the *goal* and P as a *hypothesis*.

Consider a database DB. A query $?\text{-}A;;P$ to DB is equivalent to a query $?\text{-}A$ to $DB \cup P$ (If $DB = (S_1, M_1, R_1)$ and $P = (S_2, M_2, R_2)$ then $DB \cup P = (S_1 \cup S_2, M_1 \cup M_2, R_1 \cup R_2)$). That is, P is inserted into DB before A is processed. In other words, P works as a hypothesis for $?\text{-}A$. As hypotheses are incrementally inserted into a database, nested transactions are introduced to control such insertions. See the details in [10].

An *answer* is defined as the triple (D, V, E) of a set of subsumption constraints D that cannot be solved during query processing, a set of variable constraints V that are bounded during query processing, and the corresponding derivation flow E. D lacks information in the database, obtained by abduction, V is an answer in the sense of constraint logic porgramming and E is the explanaiton.

3.5 *Quixote* System

A *Quixote* system consists of a *client* as a user interface in C on UNIX™ and a *server* as a knowledge-base engine in a parallel logic programming, KL1, which was designed and developed by ICOT to run under UNIX. A server and clients are connected by the TCP/IP protocol. One of the user interfaces is *Qmacs* using *GNU-Emacs*, while others are windows that are implemented using *X-Window* and which display some figures graphically. The overall architecture is shown in Figure 1, while Figure 2 shows an example of a graphical figure.

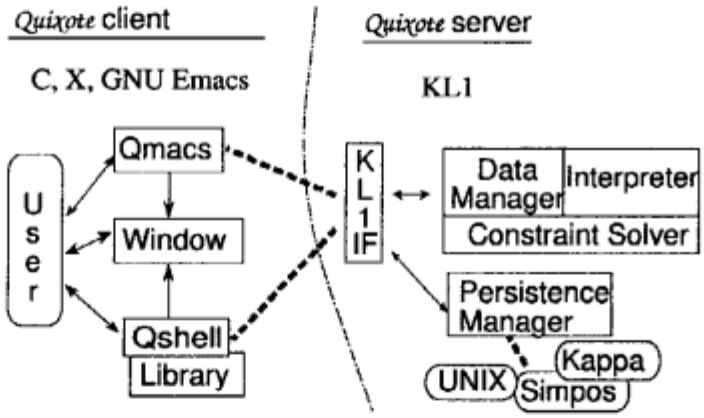

Figure 1: Architecture of *Quixote* system

4 Legal Reasoning on *Quixote*

In this section, we explain our legal reasoning system, TRIAL, written as a *Quixote* system. The overall architecture of the system, written in KL1, is shown in Figure 3. *Quixote* supports the functions of rule transformation and deductive reasoning as native functions besides the database component, while TRIAL supports analogy detection besides the interface component. All data and knowledge in the database component is written in *Quixote*.

A new case, *new-case*, (in the New Case Database), is represented as the module *new-case* in *Quixote* as follows:

$$
\begin{aligned}
\mathit{new\text{-}case} :: \{&\mathit{new\text{-}case}/[\mathit{who} = \mathit{mary},\\
&\quad \mathit{while} = \mathit{intermission},\\
&\quad \mathit{result} = \mathit{heart\text{-}attack}];;\\
&\mathit{relation}[\mathit{state} = \mathit{employ}, \mathit{employee} = \mathit{mary}]\\
&\quad /[\mathit{affiliation} = \mathit{organization}[\mathit{name} = \text{“}S\text{”}],\\
&\quad\ \mathit{job} \rightarrow \mathit{driver}]\}
\end{aligned}
$$

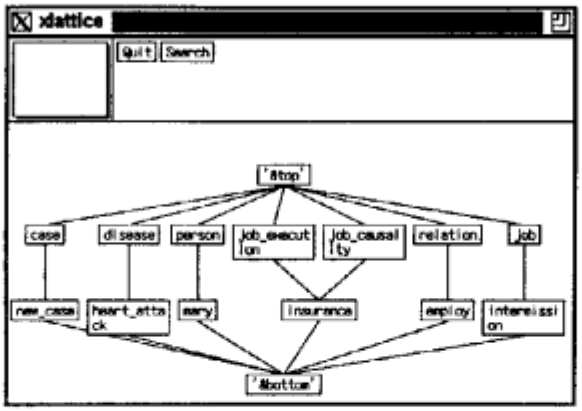

Figure 2: A window display of a lattice

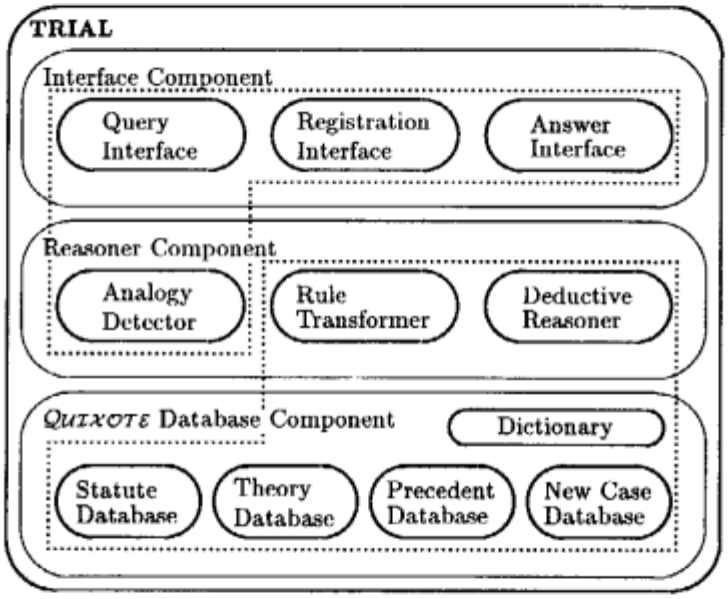

Figure 3: Architecture of TRIAL

where ";;" is a delimiter between rules. Assume that there are two *abstract* precedents [1] of *job-causality* and *job-execution*:

$$
\begin{aligned}
case_1 :: judge[case = X]/[judge \rightarrow job\text{-}execution] \\
\Leftarrow relation[state = Y, employee = Z]/[cause = X], X \\
||\{X \sqsubseteq \mathtt{parm}.case, Y \sqsubseteq \mathtt{parm}.state, Z \sqsubseteq \mathtt{parm}.employee\};; \\
case_2 :: judge[case = X]/[judge \rightarrow job\text{-}causality] \\
\Leftarrow X/[while = Y, result = Z], \\
||\{X \sqsubseteq \mathtt{parm}.case, Y \sqsubseteq \mathtt{parm}.while, Z \sqsubseteq \mathtt{parm}.result\}.
\end{aligned}
$$

Note that variables X, Y, and Z in both rules are restricted by the properties of an object parm. That is, they are already abstracted by parm and their abstract level (the range of variables) is controlled by parm's properties. Such precedents are retrieved from the precedent database by analogy detection and are abstracted by rule transformation.

We must consider the *labor-law* (in the statute database) and a *theory* (in the theory database) as follows:

[1] In this paper, we omit the rule transformation step and assume that abstract interpretation rules are given.

$$
\begin{aligned}
&\mathit{labor\text{-}law} :: \mathit{organization}[\mathit{name}=X] \\
&\qquad /[\mathit{responsible} \rightarrow \mathit{compensation}[\mathit{object}=Y, \mathit{money}=\mathit{salary}]] \\
&\qquad\quad \Leftarrow \mathit{judge}[\mathit{case}=C]\ /[\mathit{judge} \rightarrow \mathit{insurance}], \\
&\qquad\quad\ \ \mathit{relation}[\mathit{state}=Z, \mathit{employee}=Y] \\
&\qquad\quad\ \ \ \ /[\mathit{affiliation}=\mathit{organization}[\mathit{name}=X]] \\
&\qquad\quad\ \ ||\{C \sqsubseteq \mathit{case}\}. \\
&\mathit{theory} :: \mathit{judge}[\mathit{case}=X]/[\mathit{judge} \rightarrow \mathit{insurance}] \\
&\qquad\quad \Leftarrow \mathit{judge}[\mathit{case}=X]/[\mathit{judge} \rightarrow \mathit{job\text{-}causality}], \\
&\qquad\quad\ \ \mathit{judge}[\mathit{case}=X]/[\mathit{judge} \rightarrow \mathit{job\text{-}execution}] \\
&\qquad\quad\ \ ||\{X \sqsubseteq \mathit{case}\}.
\end{aligned}
$$

Furthermore, we must define the parm object as follows:

$$
\begin{aligned}
\mathit{parm} :: \mathtt{parm}/[&\mathit{case} = \mathit{case}, \mathit{state} = \mathit{relation}, \mathit{while} = \mathit{job}, \\
&\mathit{result} = \mathit{disease}, \mathit{employee} = \mathit{person}].
\end{aligned}
$$

This object results from abstracting precedents and is used for the control of predicting judgements.

To use parm for case_1 and case_2, we define the following submodule relation:

$$\mathit{parm} \sqsupseteq_S \mathit{case}_1 \cup \mathit{case}_2.$$

This information is dynamically defined during rule transformation, because the choice of precedents is experimental.

Furthermore, we must define the subsumption relations:

case	$\sqsupseteq$	*new-case*	*person*	$\sqsupseteq$	*mary*
relation	$\sqsupseteq$	*employee*	*job-causality*	$\sqsupseteq$	*insurance*
disease	$\sqsupseteq$	*heart-attack*	*job-execution*	$\sqsupseteq$	*insurance*
job	$\sqsupseteq$	*intermission*			

Then, we can ask some questions with a hypothesis to the above database:

1) If *new-case* inherits *parm* and *theory*, then what kind of *judgment* do we obtain?

$$
\begin{aligned}
&?\text{-}\ \mathit{new\text{-}case} : \mathit{judge}[\mathit{case}=\mathit{new\text{-}case}]/[\mathit{judge}=X];; \\
&\mathit{new\text{-}case} \sqsupseteq_S \mathit{parm} \cup \mathit{theory}.
\end{aligned}
$$

We get three answers, in which the first is returned unconditionally, while the latter two are answers with assumptions:

- $X \sqsubseteq \mathit{job\text{-}causality}$
- if $\mathit{new\text{-}case} : \mathit{judge}[\mathit{case}=\mathit{new\text{-}case}]$ has $\mathit{judge} \sqsubseteq \mathit{job\text{-}execution}$, then $X \sqsubseteq \mathit{insurance}$
- if $\mathit{new\text{-}case} : \mathit{relation}[\mathit{state}=\mathit{employ}, \mathit{employee}=\mathit{mary}]$ has $\mathit{cause}=\mathit{new\text{-}case}$, then $X \sqsubseteq \mathit{insurance}$

2) If *new-case* inherits *labor-law* and *parm*, then what kind of responsibility should the organization to which Mary is affiliated have?

$$
\begin{aligned}
&?\text{-}\ \mathit{new\text{-}case} : \mathit{organization}[\mathit{name}=\text{“}S\text{”}]/[\mathit{responsible}=X];; \\
&\mathit{new\text{-}case} \sqsupseteq_S \mathit{parm} \cup \mathit{labor\text{-}law}.
\end{aligned}
$$

We get two answers with assumptions:

- if $\mathit{new\text{-}case} : \mathit{judge}[\mathit{case}=\mathit{new\text{-}case}]$ has $\mathit{judge} \sqsubseteq \mathit{job\text{-}execution}$,
  then $X \sqsubseteq \mathit{compensation}[\mathit{obj}=\mathit{mary}, \mathit{money}=\mathit{salary}]$
- if $\mathit{new\text{-}case} : \mathit{relation}[\mathit{state}=\mathit{employ}, \mathit{employee}=\mathit{mary}]$ has
  $\mathit{cause}=\mathit{new\text{-}case}$,
  then $X \sqsubseteq \mathit{compensation}[\mathit{obj}=\mathit{mary}, \mathit{money}=\mathit{salary}]$

For analogy detection, the parm object plays an essential role in determining how to abstract rules, as in case_1 and case_2, which properties or objects are to be abstracted using the properties of parm, and which values are to be given for the properties of parm. In this experimental system, which adds to the basic functions of QUIXOTE, we have experimented with not only hypothetical reasoning and abduction, but also such abstraction, that is, analogy detection.

For TRIAL's user interface, QUIXOTE returns explanations (derivation graphs) with corresponding answers, if necessary. The TRIAL interface displays this graphically according to a user's request. By judging an answer on the basis of the validity of the assumptions and the corresponding explanation, the user can update the database or change the abstraction strategy.

The TRIAL system was implemented for four months by two persons. This highlights the productivity of QUIXOTE for such knowledge information processing systems.

5 Concluding Remarks

Knowledge information processing applications will play an important role in future information processing systems, including future generation databases. Legal reasoning systems, one such application, show typical requirements for future database systems.

From the experiences of TRIAL, we can list several requirements for next generation database systems, which need not necessarily be only for legal applications:

- *Processing partial information*
  As data and knowledge are often not given in a perfect form, unlike conventional applications, we must consider ambiguous or erroneous data as well as null values and logically incomplete information such as negation and disjunction. In *Quixote*, we use subsumption constraints to handle ambiguous and partially lacking properties. They are useful not only to knowledge databases but also to scientific databases.

- *Realizing an environment for thinking experiments*
  Answers are not necessarily given uniquely in knowledge information processing, but are refined by repeating trial-and-error querying with the users. In this sense, the features of hypothetical reasoning and hypothesis generation are very important.

- *Framework of very large database and knowledge-base*
  Classification mechanisms are very important for storing large databases and knowledge-bases. Subsumption and submodule hierarchies contribute to such classification. Especially, a framework that allows inconsistent data and knowledge to co-exist is needed.

- *Integration of heterogeneous data and knowledge*
  Even if we consider only one application, we can find various kinds of data and knowledge within it. For example, legal data includes large amounts of text data as the primary data, and abstracted data or knowledge (including rules) as the higher level data. Although we have not integrated such data and knowledge into TRIAL, the integration of such heterogeneous data and knowledge will become very important.

- *Knowledge discovery in databases*
  To classify large databases and knowledge-bases, two kinds of knowledge discovery will be needed: how to find erroneous and lacking information; how to find new knowledge (abstracted rule in the above example).

- *Integration of technologies in related areas*
  Database technologies have been spreading: for example, we now have deductive databases, database programming languages, deductive object-oriented databases, and very large knowledge-bases. More technologies in many areas such as artificial intelligence, programming languages, and operating systems, should be embedded into database systems.

Quixote and *μ-Quixote* systems, which run under UNIX, have been released as ICOT free software. We have been extending the features of the systems as next generation database systems and a framework of heterogeneous, distributed, cooperative problem solvers to enable their application to a wider range of knowledge information processing applications such as natural language processing and genetic information processing systems.

Acknowledgments

The authors wish to thank Nobuichiro Yamamoto (Hitachi, Ltd.) for designing and implementing the TRIAL system, and all the members of the *Quixote* project for their valuable advice.

References

[1] H.Aït-Kaci. "An Algebraic Semantics Approach to the Effective Resolution of Type Equations", *Theoretical Computer Science*, no.45, 1986.

[2] A.J. Bonner and M. Kifer, "Transaction Logic Programming", *Proc. Int'l Logic Programming*, 1993.

[3] L.O. Kelso, "Does the Law Need a Technological Revolution?", *Rocky Mt. Law Rev.*, vol.18, pp.378-392, 1946.

[4] M. Kifer, G. Lausen, and J. Wu, "Logical Foundations of Object-Oriented and Frame-Based Languages", *SUNY TR 93/06*, June, 1993.

[5] K. Nitta, Y. Ono, T. Chino, T. Ukita, and S. Amano, "HELIC-II: A Legal Reasoning System on the Parallel Inference Machine", *Proc. Int. Conf. on FGCS*, ICOT, Tokyo. June 1-5, 1992.

[6] N. Yamamoto, "TRIAL: a Legal Reasoning System (Extended Abstract)", *Joint French-Japanese Workshop on Logic Programming*, Renne, France, July, 1991.

[7] H. Yasukawa and K. Yokota, "Labeled Graph as Semantics of Objects", *Proc. Joint Workshop of SIGDBS and SIGAI of IPSJ*, Nov., 1990.

[8] H. Yasukawa, H. Tsuda, and K. Yokota. "Objects, Properties, and Modules in *QUIXOTE*", *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, June 1-5, 1992.

[9] K. Yokota and H. Yasukawa, "Towards an Integrated Knowledge-Base Management System — Overview of R&D on Databases and Knowledge-Bases in the FGCS Project", *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, June 1-5, 1992.

[10] K. Yokota, H. Tsuda, and Y. Morita, "Specific Features of a Deductive Object-Oriented Database Language *QUIXOTE*", *Workshop on Combining Declarative and Object-Oriented Databases, (ACM SIGMOD'93 Workshop)*, Washington DC, May 29, 1993.

[11] K. Yokota and M. Shibasaki, "Can Databases Prodict Legal Judgements?", *Joint Workshop of IPSJ SIGDSS and IEICE SIGDE (EDWIN)*, Nagasaki, July 21-23, 1993. (in Japanese)