Knowledge Information Processing Software

Katsumi Nitta, Kazumasa Yokota, Akira Aiba and Masato Ishikawa

Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
{nitta, kyokota, aiba, ishikawa}@icot.or.jp

Abstract

In the FGCS Follow-on project, the second research department has developed knowledge processing software for knowledge representation and applications. For knowledge representation software, we have improved the functions of a knowledge representation language, *Quixote*, and developed a heterogeneous distributed cooperative system, *Helios*. *Helios* solves real complex problems by communicating with several problem solvers. As application software, we developed new function modules for genetic information processing and legal reasoning. The genetic information processing group succeeded in creating a useful tool for sequence analysis, and has worked on constructing a biological knowledge base and on structure analysis. The legal reasoning group developed a software tool which consists of an argumentation function and a debate function. Overviews of this research are introduced briefly.

1 Introduction

In the FGCS project, knowledge processing groups developed basic knowledge processing technologies such as knowledge representation and high level inference, and developed several application systems.

Among the basic knowledge processing technologies, a parallel constraint logic programming language, GDCC, which solves algebraic constraint problems, and a knowledge representation language, *Quixote*, are based on sound theories and have been given high evaluations as infrastructures for developing intelligent systems.

Among application programs, the frontiers have been genetic information processing and legal reasoning. These application groups have conducted collaborative work with human experts, and by running these systems on the Parallel Inference Machines, they showed a remarkable increase in speed.

Based on the FGCS project research, we attempted to consolidate basic technologies of knowledge processing, show the effectiveness of parallel inference in application systems, and develop frontier application systems.

However, to develop highly intelligent systems, much research remains to be done. For knowledge representation technologies, for instance, we looked at real complex problems such as genome analysis and legal reasoning. To solve these problems, we have to use various databases and various programs to analyze them, and have to arrange them using domain knowledge. In such cases, we need a new paradigm which solves complex and difficult problems by communicating with several problem solvers.

In addition to basic technologies, genetic information processing and legal reasoning have many areas for improvement. For example, programs of amino acid sequence analysis are expected to be more practical when they can apply genetic algorithm and knowledge based technologies. Programs of structure analysis will be improved by developing new technologies for Hidden Markov Modes. And a legal reasoning system will be more useful if it also has a function which generates interpretations of the law. In order to make these application systems be more practical, they must be evaluated by human experts. Therefore, to improve portability is also an important task.

Taking into these considerations, in the FGCS Follow-on project, we set the following targets for the second research department.

1. To develop a new paradigm for solving large problems.
2. To develop more intelligent functions for genetic information processing and legal reasoning.
3. To improve the portability of systems by developing in KLIC.

In this paper, we introduce results of researches in the FGCS Follow-on project. In Section 2, we introduce research into knowledge representation technologies, and in Section 3, we explain the new functions for genetic information processing and legal reasoning. In Section 4, we show the collaborative works of these research groups.

2 Knowledge Representation Technologies

2.1 Overview

In the FGCS project and its Follow-on project, we have worked on various knowledge information processing applications, such as natural language processing, genetic information processing, and legal reasoning. In such an environment, our major interest from a knowledge representation point of view has been how to support these applications with by domain-independent languages and systems [Yokota and Yasukawa 1992, Yokota and Aiba 1994].

We have two research objectives: representing and managing data and knowledge efficiently, and providing an environment where multiple languages and systems can work effectively for a single purpose. For the first, we proposed a framework of a deductive object-oriented database (DOOD) and have designed and developed the *Quixote* language [Yokota *et al.* 1993]. As the second target, we proposed a multi-agent based framework for a heterogeneous, distributed, cooperative problem solving system, *Helios*[Yokota and Aiba 1994], which is now under development.

From a database point of view, *Quixote* is an upper layer of *Kappa*[Kawamura *et al.* 1992] that makes representation and management of higher-level data and knowledge possible, while Helios can be thought an extended multidatabase[Yokota 1994a], where multiple heterogeneous database management systems can process queries cooperatively. From a knowledge representation point of view, *Quixote* is a flexible knowledge representation language with both logic and object-orientation features, while *Helios* makes it possible to combine various representation languages and constraint solvers in a single problem solver.

Helios is an attempt to provide an environment for combining problem solvers such as constraint solvers, databases, and application programs written in various programming languages with various data representations in one system working in a distributed environment. Each problem solver in Helios utilizes the other solvers' functions, which cooperate and negotiate with each other to solve complicated problems.

Besides those research activities, we also implemented a parallel constraint logic programming language GDCC [Terasaki *et al.* 1992], and its parallel constraint solvers on top of KLIC system. GDCC is a language by which problems can be represented declaratively in terms of a set of constraints, and was originally developed in the FGCS project by using KL1 language running on PIMs. It utilizes a parallelized Buchberger algorithm to handle algebraic equality constraints, and a parallelized modified Buchberger algorithm to handle boolean constraints. Along with the development of KLIC, aiming for extending the environment for using KL1 from PIMs to UNIX based workstation, we also reimplemented the GDCC language processor for UNIX machines. By using a GDCC language processor for KLIC, the GDCC facility can be used on UNIX based workstations. This version of GDCC on KLIC will be released as *ICOT Free Software* until the end of March 1995.

In the following subsection, we give an overview of *Quixote* and Helios.

2.2 *Quixote*

Quixote is an object-oriented logic programming language with database features and is considered a DOOD language. The main specific characteristics of *Quixote* are as follows:

1. Integration of logic and object-orientation features by *subsumption constraints*.
2. Knowledge classification and inheritance by a *module* concept.
3. Query processing for *partial information databases*: abduction of subsumption constraints and hypothetical reasoning.
4. Support of *database management system features* such as concurrency control, nested transactions, and persistence.

We have developed applications such as legal reasoning, genetic information processing, and natural language processing in *Quixote*, and have shown the effectiveness and efficiency of the language.

After many efforts to build knowledge representation languages and extensions of deductive databases in the FGCS project, we started to design the *Quixote* language in 1990 and have since implemented the system. The *Quixote* system has been released as *ICOT Free Software* (the third version in June, 1993 and the fourth version in December, 1994). The system can work under UNIX, MS-DOS, and Macitosh environments.

2.2.1 Basic Features of *Quixote* Objects

An object in *Quixote* consists of an object identifier (oid) and a set of properties. Each property can be considered a method, as usual, and the implementation of a method is written in the body of a rule. The subsumption relation among oids makes property inheritance possible.

(1) Object Identity and Subsumption Constraints

An oid is in the form of a tuple called an *object term*. For example,

$apple$,

$apple[color = red]$, and

$cider[alcohol = yes,$
$\quad product = process[source = apple,$
$\quad\quad process = ferment]]$

are object terms, where the first term is *basic*, but the latter two are *complex*.

Given a *subsumption relation* (partial order) $\sqsubseteq$ among basic object terms, the relation is extended among complex object terms as usual. For example,

$$apple \sqsupseteq apple[color = red].$$

Congruence relation $o_1 \cong o_2$ is defined as $o_1 \sqsubseteq o_2 \wedge o_1 \sqsupseteq o_2$. We assume that the subsumption relation among object terms constitutes a lattice without loss of generality because it is easy to construct a lattice from a partially ordered set. Meet and join operations on object terms are denoted by $\downarrow$ and $\uparrow$, respectively.

Properties are defined as a set of subsumption constraints of an oid and used with the oid as follows:

$apple|\{apple.species \sqsubseteq rose,$
$\quad apple.area \sqsupseteq_H \{aomori, nagano\}\},$

where *apple* is an object term and the right hand side of | is a set of properties: $apple.species \sqsubseteq rose$ means that *apple*'s *species* is (subsumed by) *rose* and $apple.area \sqsupseteq_H \{aomori, nagano\}$ means that there are *aomori* and *nagano* in *apple*'s production *areas*. Here a relation between sets is defined as a Hoare ordering based on subsumption relations:

$$S_1 \sqsubseteq_H S_2 \stackrel{def}{=} \forall e_1 \in S_1, \exists e_2 \in S_2,\ e_1 \sqsubseteq e_2.$$

Although Hoare ordering is not partial, we assume it as a partial order because the representative of an equivalence class modulo $\sqsubseteq_H$ is easily defined as a set where no element is subsumed by other elements in the same set.

(2) Property Inheritance

For *property inheritance*, we assume the following rule:

if $o_1 \sqsubseteq o_2$,
then $o_1.l \sqsubseteq o_2.l$ and $o_1.l' \sqsubseteq_H o_2.l'$,

where o_1 and o_2 are object terms, l and l' are labels, and l and l' take a single value and a set value, respectively. According to the rule, we get, for example,

if $apple.species \sqsubseteq rose$,
then $apple[color = red].species \sqsubseteq rose$,
and
if $apple[color = red].area \sqsupseteq_H \{fukushima\}$,
then $apple.area \sqsupseteq_H \{fukushima\}$.

That is, $apple.species \sqsubseteq rose$ is downward inherited from *apple* to $apple[color = red]$, while $apple[color = red].area \sqsupseteq_S \{fukushima\}$ is upward inherited from $apple[color = red]$ to *apple*.

Note that there are two kinds of properties: properties in an object term and properties in the form of constraints. The former are called *intrinsic* and the latter are called *extrinsic*. Only extrinsic properties (subsumption constraints) are inherited according to the (extended) subsumption relation among object terms.

Intrinsic properties interrupt property inheritance as follows:

> Even if *apple* has $apple.color \cong green$,
> $apple[color = red]$ does not inherit $color \sqsubseteq green$
> because the intrinsic property $color = red$ with
> the same label *color* rejects the extrinsic property $color \sqsubseteq green$.

This corresponds to *exception* of property inheritance.

Multiple inheritance is defined as the merging of subsumption constraints. Such constraints are reduced as follows:

if $p.l \sqsubseteq a$ and $o.l \sqsubseteq b$, then $o.l \sqsubseteq a \downarrow b$,
if $a \sqsubseteq o.l$ and $b \sqsubseteq o.l$, then $a \uparrow b \sqsubseteq o.l$,
if $o.l \sqsubseteq_H s_1$ and $o.l \sqsubseteq_H s_2$, then $o.l \sqsubseteq_H s_1 \cup s_2$,

and

if $s_1 \sqsubseteq_H o.l$ and $s_2 \sqsubseteq_H o.l$,
$\quad$ then $\{x \downarrow y | x \in s_1, y \in s_2\} \sqsubseteq_H o.l$,,

Note that the least upper bound of the two sets, s_1 and s_2, is defined as $s_1 \cup s_2$ under Hoare ordering, because $s_1 \cup s_2 \sqsubseteq_H \{e_1 \uparrow e_2 |\ e_1 \in s_1, e_2 \in s_2\}$. In the above example, the merging of $apple.area \sqsupseteq_H \{aomori, nagano\}$ and $apple.area \sqsupseteq_H \{fukushima\}$ is reduced to $apple.area \sqsupseteq_H \{aomori, nagano, fukushima\}$.

(3) Intentional Objects

An object can be defined intentionally in the form of a *rule*:

$$o_0|C_0 \Leftarrow o_1|C_1, \cdots, o_n|C_n \parallel C,$$

where, for $0 \leq i \leq n$, o_i is an object term and C_i is a set of the related subsumption constraints, and C is a set of constraints (variable constraints). An object $o_0|C_0$ is intentionally or conditionally defined by the rule. $o_0|C_0$ is a *head* and $o_1|C_1, \cdots, o_n|C_n \parallel C$ is a *body*. Intuitively, 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*. In a sense, an object is defined as a set of rules with the same object term. There is one important restriction in C_0: C_0 may not contain subsumption relations among object terms. The reason for this is to avoid destruction of the lattice during query processing.

Note that an object term plays the role of an oid. That is, two facts,

$$o|\{o.l \sqsubseteq a\} \Leftarrow \text{and}$$
$$o|\{o.l \sqsubseteq b\} \Leftarrow$$

can be merged as follows:

$$o|\{o.l \sqsubseteq a \downarrow b\} \Leftarrow .$$

In cases where an object term with variables is in the head of a rule, the object is defined when the variables are instantiated during query processing. That is, subsumption constraints in a head are merged after evaluation of all the related rules.

When the constraints of a head are empty, *Quixote* is an instance of CLP(X) [Jaffar and Lassez 1987]:

$$o_0 \Leftarrow o_1|C_1, \cdots, o_n|C_n \parallel C$$
$$\iff \quad o_0 \Leftarrow o_1, \cdots, o_n \parallel C_1 \cup \cdots \cup C_n \cup C$$

From a programming language point of view, the existence of head constraints in a rule makes *Quixote* an extension of CLP(X) and, in the procedural semantics, the possibility of merging head constraints must be checked at every OR node of the resolution tree.

2.2.2 Databases and Query Processing

(1) Modules and Databases

A set of rules can be defined as a *module*:

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

This means that a module identified by a *module identifier* (mid) m has rules $r_1, \cdots, r_n$. We use 'module m' instead of 'a module identified by mid m' for simplicity. Here, we define the *submodule relation* between modules. For example, consider two submodule relations:

$$m_1 \sqsupseteq_S m_2 + m_3, \text{ and}$$
$$m_2 \sqsupseteq_S m_4.$$

The definitions mean that m_1 inherits all rules in m_2 and m_3, and m_2 inherits all rules in m_4. We call such inheritance *rule inheritance*, where exception, locality, and overriding are also defined [Yokota *et al.* 1993]. The submodule relation is an acyclic directed graph, in which modules can be nested. Rules without a mid are inherited by all modules. For example, consider the following three definitions:

(1) $m_1 :: \{r_{11}, \cdots, r_{1n}\}$
$m_2 :: \{r_{21}, \cdots, r_{2m}\}$
$\{m_1, m_2\} :: \{r_{31}, \cdots, r_{3l}\}$

(2) $m_1 :: \{r_{11}, \cdots, r_{1n}\}$
$m_2 :: \{r_{21}, \cdots, r_{2m}\}$
$r_{31}, \cdots, r_{3l}$

(3) $m_1 :: \{r_{11}, \cdots, r_{1n}\}$
$m_2 :: \{r_{21}, \cdots, r_{2m}\}$
$common :: \{r_{31}, \cdots, r_{3l}\}$
$m_1 \sqsupseteq_S common$
$m_2 \sqsupseteq_S common$

In any of these definitions, m_1 has $r_{11}, \cdots, r_{1n}, r_{31}, \cdots, r_{3l}$ and m_2 has $r_{21}, \cdots, r_{2m}, r_{31}, \cdots, r_{3l}$.

A module can be referenced from a subgoal in a rule in other modules. The definition of a rule is extended as follows:

$$m_0 :: o_0|C_0 \Leftarrow m_1 : o_1|C_1, \cdots, m_n : o_n|C_n \parallel C$$

which means that there is a rule in module m_0 such that if $o_i|C_i$ and C are satisfiable in module m_i for all $1 \leq i \leq n$, then $o_0|C_0$ is satisfiable in a module m_0. There might be some discussion about why a rule is not defined as follows:

$$m :: m_0 : o_0|C_0 \Leftarrow m_1 : o_1|C_1, \cdots, m_n : o_n|C_n \parallel C.$$

According to the definition, module m knows some knowledge in another module, that is, the rule corresponds to a kind of *brief*. However, this causes serious semantic problems.

The objectives of a module are as follows:

1. Classification of data and knowledge under certain criteria.
2. Coexistence of inconsistent data and knowledge.
3. Modular programming style.

These features are also very useful for constructing very large knowledge-bases (VLKB).

A *Quixote database* or a *Quixote program* is defined as a triple (S, M, R) of a set S of subsumption relations among basic objects, a set M of submodule relations among mids, and a set R of rules. Intuitively, a database can be also considered as a set of modules or as a set of objects.

(2) Query Processing

One of the major characteristics in knowledge information is the partiality of information. That is, sufficient information is not necessarily given, as for example in business applications. The introduction of an object identity is essential for representing such partial information. Partiality should be considered not only in representation but also in query processing:

1. What information is lacking in the database for this query?
2. If some information is inserted into the database, what answer will be given for this query?

Further, as derivation becomes more complicated, we want to know why a particular answer is returned.

In logic programming, finding a lack of information or unsatisfiable subgoals corresponds to abduction, that is, hypothesis generation. Remember that a rule in *Quixote* can be represented as follows:

$$o_0|C_0 \Leftarrow o_1, \cdots, o_n \parallel C_1 \cup \cdots \cup C_n \cup C,$$

where oids $o_1, \cdots, o_n$ are considered as existence checks of the corresponding objects, while $C_1 \cup \cdots \cup C_n \cup C$ is considered to be a satisfiability check of subsumption and variable constraints. In the current implementation of *Quixote*, only subsumption constraints in $C_1 \cup \cdots \cup C_n \cup C$ are taken as assumptions. That is, even if body constraints are not satisfied, they are taken as a conditional part of an answer. For example, consider a database consisting of three objects:

$$o_1 \Leftarrow o_2 \| \{o_2.l \sqsubseteq a\}.$$
$$o_2.$$
$$o_3 \Leftarrow o_4.$$

For a query ?-o_1, the answer is that if $o_2.l \sqsubseteq a$, then yes, while, for a query ?-o_3, the answer is no.

Further, the derivation process of an answer is also returned as an explanation with the answer. That is, each answer is in the following form:

if *assumptions* then *answer* because *explanation*,

where both *assumptions* and *answer* are in the form of a set of constraints.

On the other hand, hypothetical reasoning corresponds to the insertion of hypotheses into a database. A query is in the following form:

if *hypotheses* then ?-*query*
(written as ?-*query*;;*hypotheses*).

For example, consider the database used above. For queries ?-o_1;;$o_2|\{o_2.l \sqsubseteq a\}$ and ?-o_3;;o_4, both answers are yes without any assumptions. That is, if, for a query ?-q, the answer is 'if H then A', then, for a query ?-q;;H, the answer is simply A. Note that, as a database consists of a triple of definitions of subsumption relations, submodule relations, and rules, hypotheses can also consist of such a triple. A query ?-q;;(S_H, M_H, R_H) to a database (S, M, R) is equivalent to a query ?-q to a database $(S \cup S_H, M \cup M_H, R \cup R_H)$.

Hypotheses are incrementally inserted into a database, that is, a query ?-q;;H to a database DB updates the database to $DB \cup H$. To control such repetitive insertions of hypotheses, nested transactions are introduced. Users can declare *begin_transaction*, *abort_transaction*, or *end_transaction* at any time among queries. A top-level commit operation (an outermost transaction from *begin_transaction* to *end_transaction*) makes insertions persistent. On the other hand, a *roll-back* operation (caused by *abort_transaction*) recovers the before image of the corresponding *begin_transaction*. Hypothetical reasoning is useful in the construction of a knowledge-base or in *thought experiment* or *trial-and-error* type query processing.

(3) Other Features

Here, we list some more features of *Quixote*:

1. *Assertion* and *deletion* of extensional objects and properties during query processing are supported. These are controlled by the same uniform nested transaction logic used in hypothetical reasoning. However, note that the subsumption relation and submodule relation may not be updated during query processing, because the change in their inheritance might destroy the soundness of the derivation, although they may be inserted as hypotheses.

2. All objects in *Quixote* except temporarily created objects during query processing are persistent. Persistent objects in a database are stored through a uniform logical interface into other database management systems or file systems via TCP/IP protocol. Such objects are invoked when the corresponding database is opened.

3. A *Quixote* system consists of a client as a user interface and a server as a knowledge-base engine. Servers and clients are connected also by TCP/IP protocol and servers control multi-user access.

2.3 HELIOS

HELIOS [Aiba *et al.* 1994] is a framework for constructing heterogeneous distributed cooperative problem solving systems from various problem solvers such as constraint solvers, databases, and application programs implemented in various programming languages using various data representations.

To construct advanced and complicated problem solving systems in knowledge processing, three kinds of heterogeneity should be considered: model heterogeneity, spacial heterogeneity, and temporal heterogeneity. That is, multiple model, paradigm, or representation, distributed system, and extension should be considered. For this reason, a system for those problems should have corresponding three flexibilities. HELIOS is a framework having those three flexibilities for constructing those problems.

The main characteristics of HELIOS are as follows:

1. Each problem solver is encapsulated by a *capsule* for providing methods and converting its local representation to a common one. This encapsulated problem solver is called an *agent*. A problem solver in an agent is called a *substance*.

2. Agents are placed in a common space called an *environment*. An environment provides a global information including common representation of data and controls communication between agents.

3. An environment with its agents can be considered as a problem solver, and it can itself be an agent if it is encapsulated by a capsule. This allows us to construct a hierarchical agent-environment structure.

4. Communication between agents is realized by message passing between them. Those messages have flexibility on designating destinations.

After considerable work on knowledge representation technologies, such as designing, implementing and applying Quixote and GDCC, and deep consideration of huge applications such as legal reasoning, genetic information processing, and natural language understanding in the FGCS project and its Follow-on project, we started to design HELIOS in 1993. We implemented the first experimental version of HELIOS in the beginning of 1994, and the first version that works on UNIX workstations by using C language in mid-summer of 1994. The second version was implemented in November 1994 on UNIX workstations connected by Ethernet. The second version of HELIOS will be released as *ICOT Free Software* until the end of March 1995.

2.3.1 Agents, Environments, and Messages

A problem solver is encapsulated by a *capsule* which provides methods that are definitions of functionalities of the problem solver that can be used from other agents.

All communication between agents is realized by message passing. All messages are handled by an environment, and they are strongly typed. A type system used to type messages is called a *common type system*, and defined in an environment. Each agent's capsule translates messages between the common type system and the type system used in its substance.

By encapsulating an environment and its agents, a new problem solver can be constructed. If this is wrapped by a capsule, a new agent is defined. This type of agent is called a *complex agent*, while an agent with no internal structure is called a *simple agent* (Figure 1).

An agent has the following two functions: solving problems that are provided by other agents, and asking other agents to solve problems that can not be solved locally. An agent have only the former functions is called a *passive agent*, and an agent have both functionalities is called an *active agent*. Almost all problem solvers can be substances of passive agents without modification, but extensions of problem solvers are required to be substances of active agents.

An environment collects information on the agents in it, and takes care of delivery of all messages. That is, a message produced by an active agent or a message produced outside and passed from a capsule of the environment is sent to the environment, which then delivers it to agents that have a possibility of solving the problem in the message. If the environment cannot find agents that have a possibility of solving the problem, then the message is sent outside the environment through the capsule. In this sense, a user can be considered as the outermost environment of HELIOS. Thus, problems that no agent in HELIOS can answer are sent to the user.

There are many ways to designate the destination of a message. All of those ways are handled by the environment.

2.3.2 HELIOS: Its languages and Functions

The message protocol used in HELIOS includes the following information:

1. Message identifier
   An identifier for a message given by the capsule of the sender of the message. This is unique in an environment.

2. Transaction identifier
   An identifier is an identifier for controlling messages updating contents of substances.

3. Message type
   As described, a message sends either problems or answers to other agents. The former are called *query messages*, and the later are called *reply messages*. The message type is used to distinguish those types of messages.

4. Source
   An identifier of the agent that produces the message.

5. Destination of the message (Destination)
   The destination of the message can be designated in the following ways:

   Agent identifier

   In this case, the destination agent is explicitly designated. An environment maintains a directory consisting of agent identifiers and their physical address. This directory is called an *agent directory*.

   Function and process

   The functions of an agent are defined in its capsule. An environment collects them and makes a directory consisting of functions and agent identifiers. This directory is maintained in the environment. This directory is called a *function directory*. If a function is used to designate the destination of a message, then the environment refers the directory to obtain identifiers of agents that have the function designated in the message. The environment then delivers the message to those agents. The process determines the aggregate function to obtain an answer to the sender from collected answers.

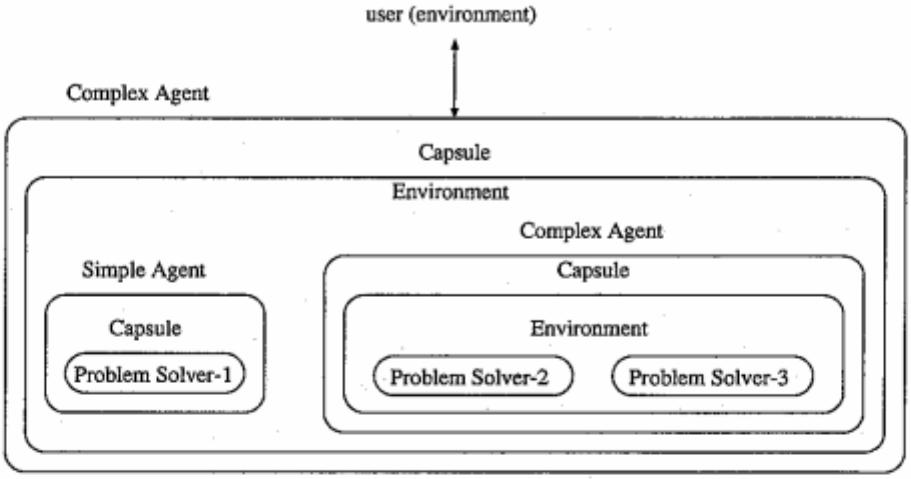

Figure 1: Basic Model for HELIOS

Function, Method, and Process

When agents with functions have methods with identical or similar function but different names, these methods can not be invoked by the above method. To deal with this situation, an environment uses a directory consisting of a pair of function and method, and a list of pairs of an agent identifier and method. This directory is called a *method directory*.

By using these functions for designating destination of messages, many protocols for cooperating and negotiating agents can be realized, such as contract net protocol, distributed constraint satisfaction, distributed constraint relaxation, etc.

To describe information and definitions required for constructing capsules and environments, two languages are provided. One is a language for capsules, called *CAPL* (CAPsule Language), and the other is a language for environments, called *ENVL* (ENVironment Language).

The following is an example of a CAPL program 3 and an ENVL program 2 by using CAPL and ENVL of the second version of HELIOS.

```
&environment n-queens;
     &common_type common-type-file;
     &agent_dir   queeen1, queen-2, queen-3,
                  queen-4, queen-5, queen-6,
                  queen-7, queen-8;
```

Figure 2: ENVL program for 8-Queens

In this ENVL program, `&environment` gives an identifier of the environment, `&common_type` gives a file name for common type declaration, and `&agent_dir` gives the agent directory. In this case, agents queen$N$ ($1 \leq N \leq 8$) are almost identical except N. In such case, a *parametric agent* can be used. A parametric agent can be defined by `&parametric_agent queen;`. By this definition, a template for agents named `queen` is defined. When the user gives commands for expanding this template with given parameters, 1 to 8 in this case, then the agents from `queen(1)` to `queen(8)` are generated. By using this facility, user is not required to write similar CAPL programs more than once.

The Figure 3 shows the main part of a CAPL program for a capsule of an agent for solving the N-Queen problem.

In the CAPL program in Figure 3, an agent defined by this capsule is a simple agent named `queen1`. `&parameter` defines a local constant, `ID`, that is used for invoking a predicate in the substance. `&env` declares that the environment of this agent is `n-Queen.` , and that the ENVL program for the environment is stored in the file `nQ.env`. The `&inside` field designates a command to wake up the substance. In this case, the command `prolog` is used to wake up the substance. The program for the substance is stored in the file named `/app/nQueen/lib/queen-pr.pl`. The language used to implement the substance is declared in the field of `&substance_type`. This capsule and the substance are connected by `pipe`.

The next field defines import methods for this agent. This agent receives a method named `solve_n_queens` with one parameter of the type int, and returns a value of the type list-of-positions. Both types belong to the common type system. The right hand side of `=>` means the invocation of a predicate in the substance.

```
&type       simple;
&parameter  ID:int = 1;
&agent      queen1;
&env        n-Queen, nQ.env;
&inside     prolog
            &sub /app/nQueen/lib/queen-pr.pl;
&substance_type prolog
&connection pipe;
&import_method
  solve_n_queens @ #1:int -> #2:[positoin]
    =>
    n_queens @ #ID:INT, #1:INT, #2:[POSITION];
                              ...

&export_method
   get_my_domain @ #1:position -> #2:positions
   <=
   bag_of!get_others_positions @ #1:POSITION, #
                                    3:INT,
                                [#2]:[POSITIONS];

                              ...

&convert
  (id=#1,position=#2):position
                  <-> [#1,#2]:POSITION;
  (id=#1,positions=#2):positions
                  <-> [#1,#2]:POSITIONS;

&self_model
  &function 4queens, 5queens, 6queens,
            7queens, 8queens;
```

Figure 3: CAPL program for 8-Queens

The following lines defines the export methods, type conversion rule, and self model are defined.

2.3.3 HELIOS as a huge system for knowledge processing

By the research and development, we provided a comprehensive testbed for heterogeneous distributed cooperative problem solving for dealing with three heterogeneity: model heterogeneity, spatial heterogeneity, and temporal heterogeneity. By making capsules and environments programmable by CAPL and ENVL, HELIOS provides flexible and extensible common space and communications. The resulting system using HELIOS as a huge system for knowledge processing from issued of sharing knowledge and reusability of knowledge/programs. Reusability of programs as agents is for temporal heterogeneity, and introducing capsule and environment are for model and spatial heterogeneity. When very large knowledge bases are considered, considering the above three kinds of heterogeneity is crucial.

3 Application Programs

3.1 Overview

In the FGCS project, genetic information processing and legal reasoning are the most successful fields in developing application systems. Both fields have large public databases and require proper knowledge bases by which we can make large-scale problem solving systems. Fortunately, academics in both fields are cooperatively active because they had succeeded in getting Grants-in-aid of Scientific Research on Priority Areas from the Ministry of Education, Science and Culture of Japan. We conducted our researches receiving their suggestion.

3.1.1 Genetic Information Processing

In the proceedings of FGCS'92, we discussed the availability of parallel inference machines in the field of genetic information processing. We reported on a parallel sequence analysis system and a parallel protein folding system. Both of them worked efficiently on PIM model-m. Preliminary studies on the knowledge representation and retrieval were also made.

The systems seemed useful for biologists. The evaluation by biologists, however, didn't go well, because the systems worked only on PIMs, which are experimental parallel machines for computer scientists. We thought that new systems should be established using UNIX-based computer resources, and that more intensive studies of biological knowledge are required to develop useful computer systems for biologists.

3.1.2 Legal Reasoning

Legal reasoning is a thinking process of lawyers to apply legal rules to a new case.

In the FGCS project, we have developed a legal reasoning system, HELIC-II [Nitta *et al.* 1992]. This system consists of two inference engines: a rule base reasoner and a case base reasoner, and it solved several criminal problems.

However, HELIC-II has following problems. (1) Though it has a function for generating arguments, it lacks in a function which selects the best one. (2) As it runs only on PIMs, it is difficult for lawyers to use it.

To resolve these problems, we started the research on the *new HELIC-II* in the FGCS Follow-on project.

3.2 Genetic Information Processing

After FGCS'92, we focused on sequence analysis problems to prove the applicability of parallel computing in genetic information processing. We intend to develop a

UNIX-based sequence analysis environment that would be convenient for biologists.

We intensively studied the presentation of biological knowledge by constructing some experimental systems. Some of them were deductive object oriented databases written in *Quixote* language. The others, written in C, learned hidden Markov models.

3.2.1 Parallel Applications

The fundamental technique for analyzing genetic sequence data by computer is to examine similarities among sequences. This usually requires large amounts of computation to find the similarities, since there are a lot of sequences in the database to be examined. The computational problem can be partly solved with parallel implementation [Coulson *et al.* 1987, Kawai *et al.* 1991].

After noticing that the iterative improvement method [Gotoh 1993] was suitable for parallel computing of genetic sequence data, we developed some parallel iterative aligners [Ishikawa *et al.* 1992, Ishikawa *et al.* 1994b]. In the following paragraphs, we discuss the most sophisticated iterative aligner and an alignment workbench featuring the parallel iterative aligners.

Parallel Aligner by Genetic Algorithm

Genetic algorithms (GAs) simulate the survival of the fittest in a population of solutions which represent points in a search space (Figure 4). In our GA alignment system [Ishikawa *et al.* 1993], each solution, though usually represented by a binary string, corresponds to a possible multiple sequence alignment. A fitness function corresponds to the alignment score. The number of solutions in a population is the same as the number of available PEs, because each PE manages a solution.

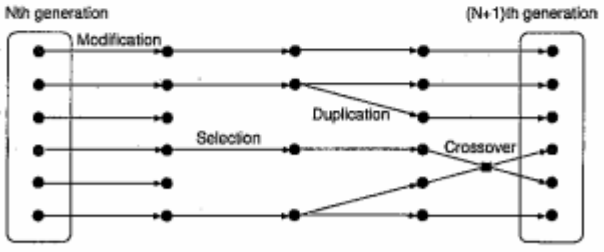

Figure 4: **Mechanism of Genetic Algorithm (GA)** — GA is a stochastic search algorithms based on the biological evolution process, whose operators consist of modification, selection, duplication, and crossover.

The modification operator changes certain parts of a solution. Although modification means a random perturbation under the orthodox concept of a genetic algorithm, it is considered an iterative cycle of improvement in our definition.

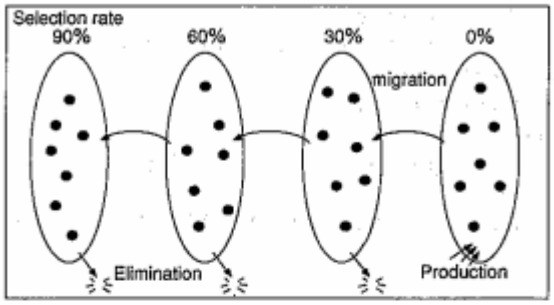

Figure 5: **Multi-Group GA Strategy** — A high-score solution migrates to a high-selection-rate group in order to do a concentrated search around its vicinity, and solutions captured in local optima surrender their computational resources to a distributed search in the lowest-selection-rate group.

The multi-group GA technique can dynamically change search strategies by migration during a GA process [Mühlenbein and Schlierkamp 1993]. We have devised a multi-group GA as shown in Figure 5 using our GA alignment system. We focused on the selection rate of our GA system. By changing the selection rate, we can create fairly different search strategies. When the system has the highest selection rate, the search becomes narrow and concentrated. Without selection, on the other hand, the search becomes wide and distributed. Our multi-group GA strategy forms a hybrid search which features the advantages of both narrow and wide searches.

Sequence Alignment Workbench

Our alignment workbench, which features the parallel iterative aligners, realizes alignment which is not only fast and high-quality, but also constraint-based [Ishikawa *et al.* 1994a]. When a user has some biological knowledge which indicates that some characters might be aligned in a column, a constraint can be defined for those characters. The constraint set is considered simultaneously in each iteration cycle of parallel alignment. Then appropriate multiple alignment is generated by the aligner and displayed in full color on the display.

The alignment workbench also contains the following characteristic sequence analysis modules: a phylogenetic tree drawer, a motif-database matcher, and a stem region specifier. The matcher identifies sequence motifs in a protein sequence alignment, retrieving motif data from the Prosite database [Bucher and Bairoch 1994]. The drawer constructs an evolutional tree, dependent on the current editing alignment. The stem specifier indicates some possible stacking regions in an RNA sequence alignment [Ishikawa *et al.* 1994c].

The iterative aligners and the alignment workbench (Figure 6) are open to the public via the Internet for biologists' use.

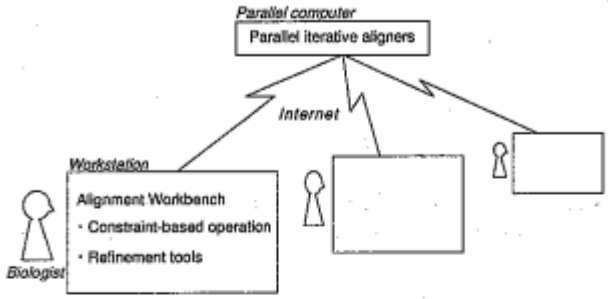

Figure 6: **Alignment Environment** — The parallel aligners, working on a PIM or a UNIX-based parallel computer such as CM5, provide the alignment workbench with aligned sequence data via the Internet.

3.2.2 Knowledge Representation

Biological knowledge representation (KR) requires very large databases, meta-databases and database flexibility. Biologists discover new facts constantly, adding them to the existing databases, and often building new databases. In order to get the latest data as quickly as possible, meta-databases, federated databases or integrated interfaces for heterogeneous databases are required [Letovsky and Berlyn 1994] [Perrière *et al.* 1994]. Encapsulation and polymorphism in the object-oriented model suggest a reasonable solution.

Biologists often request additional attributes or schema reconstruction, for they discover novel criteria or viewpoints. This is why a schema evolution facility or database flexibility is required. Several genome mapping databases employ logic programming environments such as Prolog [Yoshida *et al.* 1992] or interval logic [Cui 1994]. There are also biochemical databases in Lisp [Karp and Paley 1994] and Prolog [Kazic 1994]. The deductive object oriented database (DOOD) model satisfies both flexibility and federation of databases [Tanaka 1992] [Tanaka 1993a] [Hirosawa *et al.* 1993] [Goto *et al.* 1994].

The biological KR should also treat varieties or errors. Prosite, one of the main biological database, uses enumerative representation like regular expressions, and will soon employ another enumerative representation called *profile* [Bucher and Bairoch 1994]. Many recent researchers, however, are interested in stochastic representation such as hidden Markov models (HMM) or stochastic grammars [Asai *et al.* 1993] [Haussler *et al.* 1993] [Sakakibara *et al.* 1994] [Mamitsuka and Abe 1994].

Deductive Object Oriented Database Model

Hierarchical structures and exceptions are very common features of knowledge. Object-oriented models usually support both features by foreign object reference mechanisms such as inheritance. We have tried to build a *motif database* in an object-oriented model [Hirosawa *et al.* 1993].

Motifs are characteristic patterns of a sequence set. For instance, cytochrome c has a motif "CxxCH" in a specific position, where "CxxCH" represents a sequence of Cysteine–(any 2 amino acids)–Cysteine–Histidine. The knowledge of motifs accelerates multiple alignment, while multiple alignment clarifies the motifs.

Biochemical knowledge requires hierarchical network representation. Experiments clarify global reactions first and proceed to individual reactions. The DOOD model supports such network shape modification, but lacks a global viewpoint, because it handles networks as an aggregation of nodes and arcs. It is indispensable to supplement visualization tools of total network shapes. We have developed an experimental KB of biochemical cascades in a DOOD model (*Quixote*) with a visualization tool (Figure 7) [Hirosawa *et al.* 1995].

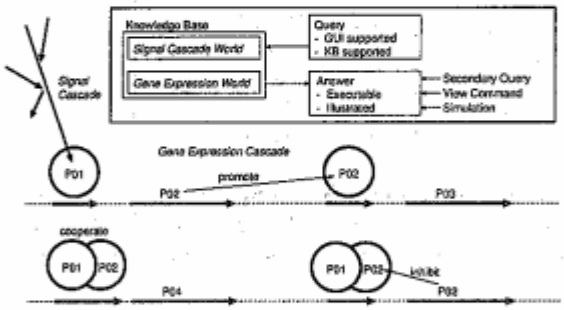

Figure 7: **Knowledge Base of Biochemical Cascades** — configuration of the knowledge base (above); signal and gene expression cascades (below); The knowledge base supports users' queries and provides visual and executable answers.

Hidden Markov Model

Multiple sequence alignment is used to clarify the conserved part and variation of a specific protein group. Profile is a mean to represent them enumeratively. The hidden Markov model (HMM), however, which is used in speech recognition, represents them stochastically [Asai *et al.* 1992] [Krogh *et al.* 1994]. Iterative sequence aligners [Ishikawa *et al.* 1992] and HMM Viterbi learning are very closely related to each other [Tanaka *et al.* 1993b].

Protein (secondary) structures were insufficiently labeled such as alpha-helix or beta-strand. MSSD, a statistical representation, classifies the local structure of

peptides [Onizuka *et al.* 1993]. MSSD enables modeling of protein structures in HMMs. Since protein structure HMMs and protein sequence HMMs are comparable, the relationship between protein structures and their sequences will be clarified.

An HMM requires a proper network configuration. There are several researches for configuration learning algorithms [Fujiwara *et al.* 1994]. We made experiments on two automatic configuration methods: a successive state splitting (SSS) algorithm for protein sequences and structures (Figure 8), and a genetic algorithm (GA) for the poly(A)-signal of DNA sequences.

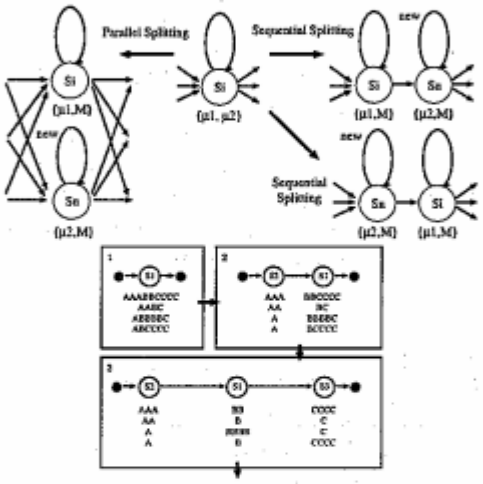

Figure 8: **Successive State Splitting** — state splitting (above); initial steps of the SSS (below); The Hidden Markov Network grows larger by splitting the state of the most *varied*.

Poly(A)signal was studied in [Yada *et al.* 1994a] and had a result in an enumerative manner as Figure 9. It appears to be represented well in stochastic manner. We developed an automatic configuration mechanism using genetic algorithms (GA) [Yada *et al.* 1994b].

3.3 Legal Reasoning

In the FGCS Follow-on project, we reexamined a model of legal reasoning which can simulate the thinking process of attorneys. While our old model has only a function for generating arguments, a new model is enhanced by adding two functions : selecting argumentation by value judgment, and debating. Based on a new model, we developed a new version of *HELIC-II*.

3.3.1 A Model of Legal Reasoning

Lawyers solves many kinds of legal problems, such as legislating, consulting, finding facts, arguing in the court,

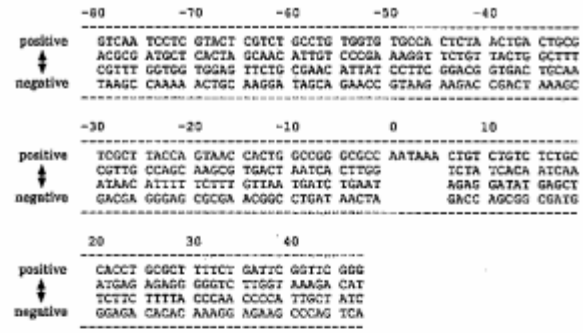

Figure 9: **Poly(A)signal around AATAAA** — a tendency around the sequence AATAAA is shown at each column (base).

and judging. Among them, we focus on arguing about application of rules to facts, and on decision-making by the judge.

When a legal problem occurs, the plaintiff (prosecutor) focuses on some important events in the problem, and generates a legal consequence (goal) which he wants to achieve. Then, the plaintiff constructs an argument which supports the goal. Then the defendant tries to make a counter-argument, and the debating process occurs.

These actions can be divided into three parts: making arguments, selecting arguments and debating. We call the functions of making and selecting arguments the argumentation function. In the old HELIC-II system, we realized only the function for making arguments.

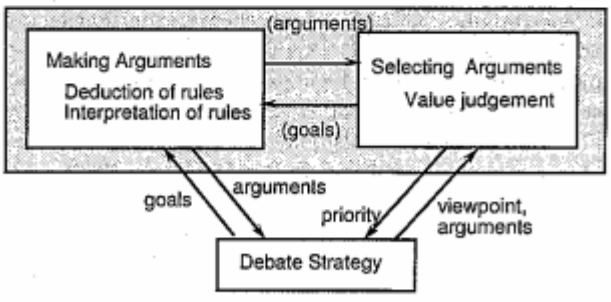

Figure 10: Basic components of legal reasoning

Our new legal reasoning model consists of seven components: facts, knowledge base, procedure for making arguments, procedure for selecting arguments, procedure for debating, and agents. We will explain each of them.

Figure 11 shows the relations of the components.

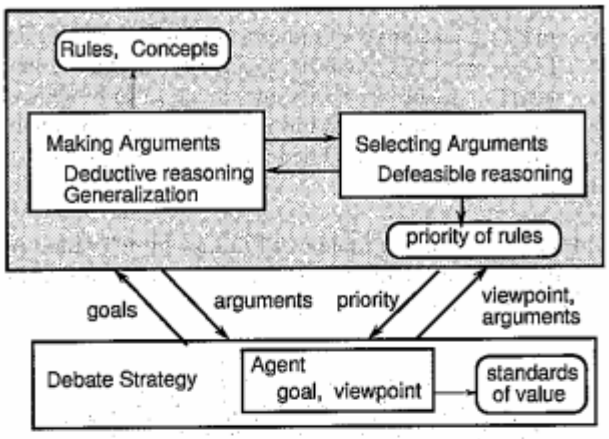

Figure 11: Legal Reasoning Model of the new HELIC-II

3.3.2 Argumentation Function

To realize the argumentation function, we have to deal with a variety of knowledge, such as legal rules, precedents, standards of value and personal viewpoints. To represent this knowledge, we developed a new language.

Here, we will show how legal knowledge is described, and how it is used for reasoning.

(1) Description of Legal Knowledge

Concepts :
Any concept appearing in the knowledge base must be defined in the concept dictionary. A concept is defined as a type definition and a subsumption relation. For example,"hitting other people is an act of violence" and "father is a male" are represented as follows.

$$hit < do\_violence$$
$$father[sex : male]$$

Facts :
A new case is represented as a set of facts.

$$act1 :: hit(agent = tom, object = bill) \mid \#act.$$
$$act2 :: with\_criminal\_intent(a\_object = \#act).$$
$$act3 :: injured(a\_object = bill, cause = \#act).$$

Statutory rules, Legal theories and Precedents:
As most legal rules take the form of "if - then - unless - rule", they are easily represented as rules. The following is an example.

$$penal36 :: punishable(a\_object = @act) \leftarrow act(agent = X/person, object = Y/person) \mid @act, not\ self\_defense(a\_object = @act).$$

When a rule is applied to facts, condition parts of some rules may be replaced by more abstract predicates. This operation corresponds to a widening interpretation.

Standard of value and viewpoint
When lawyers select one from conflicting interpretations of legal rules, they evaluate each interpretation based on their viewpoints. We describe the personal viewpoint as priority relation between standards of value.

Standards of value are represented by priority name and priority between factors of value.

$$lex\_posterior := \{case\_of\_90s > case\_of\_80s\}.$$
$$focus\_on\_economy := \{economy > polution\}.$$
$$lex\_superior := \{case\_of\_supreme > case\_of\_localCourt\}.$$
$$focus\_on\_PublicDiscipline := \{morals > freedom\_of\_press\}.$$

The relation between rules and factors of value is represented as follows. The first example means that rules,r_1, r_4, r_7 have a factor "economy".

$$economy := \{r_1, r_4, r_7\}.$$
$$polution := \{r_2, r_6\}.$$
$$case\_of\_supreme := \{r_1, r_2, r_7\}.$$
.....

A *viewpoint* is priority relation of standards of value. Different people have different viewpoints.

$$v1 := \{ focus\_on\_economy > focus\_on\_PublicDiscipline, focus\_on\_consistency\_of\_interpretation > focus\_on\_flexibility\_of\_law \}.$$

(2) Defeasible Reasoning

Defeat Relation
Let Arg_1 be an argument which supports a goal G, and let Arg_2 be an argument which supports a goal C. If C and B conflict, then Arg_2 is a counter-argument to Arg_1 (Figure 12).

Let r_1 and r_2 be top default rules included in Arg_1 and Arg_2, and let r_1 have priority over r_2. Then, if r_2 is the only default rule included in Arg_2, or if Arg_2 is not defeated by another counter counter argument, then Arg_1 is defeated by Arg_2.

An argument can be classified into three categories - a *defeated* argument, a *justified* argument and a *merely plausible* argument [Sartor 1993].
A defeated argument is an argument which is defeated by some counter argument. A justified argument is an argument which defeats any counter arguments. A merely plausible argument is an argument which is neither a defeated one nor a justified one. Justified arguments and

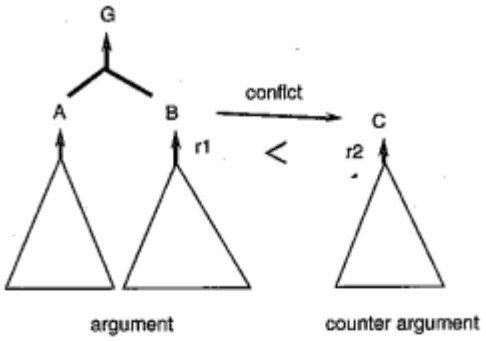

Figure 12: Defeat relation

merely plausible ones are called *plausible* ones.

Query Mode

After we give this knowledge to the *new HELIC-II*, we focus on an event in a new case, and input a query as follows.

$$?-punish(a\_object = \#hit,\ goal = crime\_of\_inflicting\_injury).$$

Then, the plausible arguments which support a goal "*crime_of_inflicting_injury*" are obtained.

3.3.3 Debating Function

When both parties debate in the court, there are two kinds of issues - issues of finding facts and issues of selecting interpretation by value judgment. The *new HELIC-II* deals with both of them. We model the debating function as follows.

1. Initially, two parties have different facts, different rules and different viewpoints. They don't know the rules and viewpoints which the opposite side may have.

2. As both parties have different viewpoints, they have different priority relation of rules. Therefore, if the plaintiff insists $r_1 > r_2$, and the defendant insists $r_2 > r_1$, we cannot decide the defeat relation between the two arguments. In this case, the system changes the issue point.

3. Both parties pose their claims. There are several possible claims, as follows: (1) To make a new argumentation for a given goal. (2) To find issues in arguments posed by the opponent. (3) To select one issue and make a counter-argument for it. (4) To decide if an argument can be defeated by a counter-argument or not. (5) To modify one's own viewpoint to defeat a counter argument. (6) To change issues.

4. During the debate process, the current viewpoint of each party may be enhanced by attaching new priority relations of standards of value. For example, let $v_1 := \{p_1 > p_2\}$ be a current viewpoint, and let $p_1 > p_3$ be a priority by which an argument of this side defeats a counter argument of opponent side. Then, v_1 is modified and becomes $v_2 := \{p_1 > p_2,\ p_1 > p_3\}$.

Figure 13 is an example of a window of debating. As debating is conducted by two parties, two windows appear in the screen.

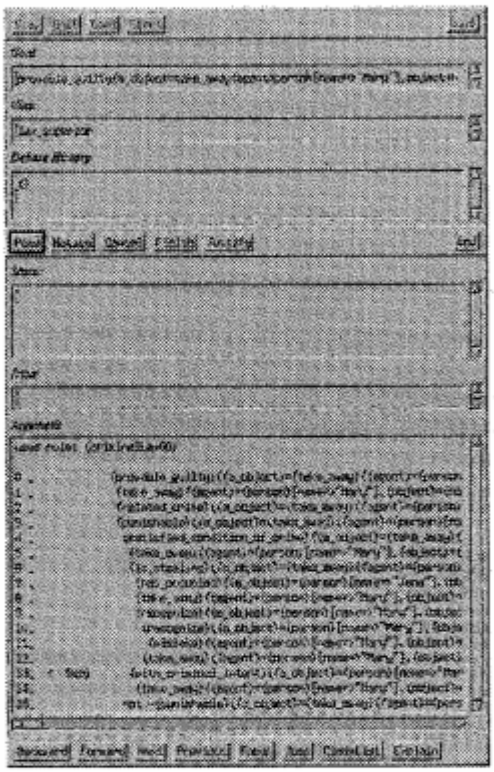

Figure 13: Debating Interface

3.3.4 Subtasks

In addition to the development of the *new HELIC-II*, the legal reasoning group has conducted following researches as subtasks.

1. Representation of legal rules of the penal code:
   We mainly focused on processes of argumentation and debate in the *new HELIC-II*, and we took AI approach to model them. However, it doesn't mean that analysis of logical aspect of legal reasoning is not important. We also had a small project which focuses on a logical model of legal reasoning. We compared and categorized legal rules of penal code systematically, and described them as rules. This

research includes the design of a new knowledge representation language for legal knowledge [Shibasaki and Nitta 1994].

2. Classification of predicates which represent events: We showed how legal knowledge is represented in the *new HELIC-II*. However, it is troublesome to input a new case because we must describe facts of a new case in detail [Tojo and Nitta 1994]. We focused that events cause various pattern of change of status, and categorized events into several groups. By using this information, even if the user omit to input some facts, the system may make up with omitted facts. This research will be used to develop a user interface of the *new HELIC-II* in the future.

4 Towards Large Scale Knowledge Applications

In the previous two sections, we showed the results of research groups in the second research department. Both sets of research were conducted collaboratively. In this section, we will how each subject is viewed from another group considering perspective for integrating each result.

Knowledge Representation Technologies
From viewpoint of knowledge representation, genetic information processing and legal reasoning have a common feature that not only a knowledge base but scientific databases play important role to solve a complex problem. For example, genetic information processing uses databases of DNA sequences, RNA sequences, protein sequences, protein structures, and so on. And legal reasoning uses a database of judicial precedents. As data types of these databases are complex, these domains are good application fields of a DOOD language *Quixote* . Furthermore, genetic information processing uses various databases each of which is developed in different laboratories. Therefore, from viewpoint of *Helios*, genetic information processing is regarded as an example of multi databases.

Genetic Information Processing
As we introduced in chapter 3.2.2, we represented biological knowledge using *Quixote* . Knowledge of molecular biology consists of sequences, local structure, global structure, chemical reaction, physical rules, and so on. To analyze biological data, biologists have to refer to several databases and combine the results of sequence analysis and structure analysis programs. *Quixote* is a useful tool to unify this, because it is a DOOD language which can deal with various types of data easily.

Our sequence alignment workbench also made progress by referring to motif data which represented by *Quixote* .

In the future, programs of genetic information processing will be unified into a single large system using *Helios*.

Legal Reasoning
Many legal reasoning systems which have been developed outside ICOT are focused on the process of making arguments. *Quixote* is a powerful tool to realize this process, because its functions of module and subsumption constraint fit representing legal knowledge. The *Quixote* group described legal rules of an international treaty (CISG) in *Quixote* with the help of the legal reasoning group and showed they are represented naturally [Takahashi and Yokota 1994] [Takahashi and Yokota 1995].

Though *Quixote* is a powerful tool for the process of making arguments, we designed a new language to develop the *new HELIC-II* because the research target of this group was not only a module for making arguments, but modules for value judgment and debate strategy. Though the new language is less powerful for the process of making arguments, it covers both the processes of making arguments and value judgment, and has an interface to communicate with the debating module. To design the new language, this group looked to the *Quixote* group for guidance.

5 Summary

We showed research results of the second research department in the FGCS Follow-on project.

As knowledge representation technologies, we showed the features of a knowledge representation language, *Quixote* , and a new paradigm of problem solving, *Helios*. Because *Helios* regards various problem solvers as agents and can treat them uniformly, we can easily make a total system by combining them. The research of *Helios* is closely related to distributed artificial intelligence, which is one of the most promising topics.

The genetic information processing group made progress in developing practical sequence analysis technologies. As they are convenient tool which runs on the Unix environment, the number of users is increasing. This group has also conducted ambitious research - constructing a biological knowledge base and analyzing the higher structures of proteins. As it will take more time to be a practical tool, our approach is one of the most promising one.

The legal reasoning group developed a software tool which has functions for both argumentation and debate. As this is the first attempt ever to combine these functions, the AI and Law worlds have given considerable attention to the *new HELIC-II*.

References

[Aiba *et al.* 1994] A. Aiba, K. Yokota, and H. Tsuda.

Heterogeneous Distributed Cooperative Probmem Solving System HELIOS. In *Proc. Int. Symposium on FGCS 94*, 1994.

[Aït-Kaci *et al.* 1986] H. Aït-Kaci and R. Nasr. LOGIN: A Logic Programming Language with Built-In Inheritance. In *Journal of Logic Programming*, 1986. pp. 185-215.

[Asai *et al.* 1992] K. Asai, S. Hayamizu, K. Onizuka, and K. Handa. Continuous Speech Recognition Techniques for Protein Structure Prediction Systems. In *Proc. Genome Informatics Workshop III*, 1992. pp. 97-100.

[Asai *et al.* 1993] K. Asai, S. Hayamizu, and K. Handa. Secondary Structure Prediction by Hidden Markov Model. In *Comput. Applic. Biosci.*, Vol. 9, 1993. pp. 141-146.

[Branting 1991] K. Branting. Integrating Rules and Precedents for Classification and Explanation: Automatic Legal Analysis, Ph D. thesis, Univ. Texas, 1991.

[Bucher and Bairoch 1994] P. Bucher, and A. Bairoch. A Generalized Profile Syntax for Biomolecular Sequence Motifs and Its Function in Automatic Sequence Interpretation. In *Proc. 2nd ISMB*, 1994. pp. 53-61.

[Coulson *et al.* 1987] A.F.W. Coulson, J.F. Collins, and A. Lyall. Protein and nucleic acid sequence database searching: a suitable case for parallel processing. In *Comput. J.*, Vol. 30. 1987. pp. 420-424.

[Cui 1994] Z Cui. Using Interval Logic for Order Assembly. In *Proc. 2nd ISMB*, 1994. pp. 103-111.

[EDR] User's manuals of conceptual dictionary, EDR, 1994.

[Fujiwara *et al.* 1994] Y. Fujiwara, M. Asogawa, and A. Konagaya. Stochastic Motif Extraction Using Hidden Markov Model. In *Proc. 2nd ISMB*, 1994. pp. 121-129.

[Gordon 1993] T. Gordon. The Pleading Game - An Artificial Intelligence Model of Procedural Justice, Ph D. thesis, GMD, 1993.

[Goto *et al.* 1994] S. Goto, N. Sakamoto, and T. Takagi. A Deductive Object-Oriented Language for Integrated Genome Databases. In *Proc. 27th HICSS*, 1994. pp. 108-112.

[Gotoh 1993] O. Gotoh. Optimal Alignment between Groups of Sequences and its Application to Multiple Sequence Alignment. In *Comput. Applic. Biosci.*, Vol. 9, 1993. pp. 361-370.

[Haussler *et al.* 1993] D. Haussler, A. Krogh, I.S. Mian, and K. Sjölander. Protein Modeling using Hidden Markov Models: Analysis of Globins. In *Proc. 26th HICSS*, 1993. pp. 792-802.

[Hirosawa *et al.* 1993] M. Hirosawa, R. Tanaka, and M. Ishikawa. Application of Deductive Object-Oriented Knowledge Base to Genetic Information Processing. In *Proc. Int. Symp. on Next Generation Database Systems and Their Applications*, 1993. pp. 116-122.

[Hirosawa *et al.* 1995] M. Hirosawa, R. Tanaka, H. Tanaka, M. Akahoshi, and M. Ishikawa. Toward Simulation-like Representation of the Cell. In *Proc. Health Sciences, Physiological and. Pharmacological Simulation Studies*, SCS, 1995.

[Ishikawa *et al.* 1992] M. Ishikawa, M. Hoshida, M. Hirosawa, T. Toya, K. Onizuka, and K. Nitta. Protein Sequence Analysis by Parallel Inference Machine. In *Proc. FGCS'92*, 1992. pp. 294-299.

[Ishikawa *et al.* 1993] M. Ishikawa, T. Toya, Y. Totoki, and A. Konagaya. Parallel Iterative Aligner with Genetic Algorithm. In *Proc. AI and Genome Workshop in 13th IJCAI*, 1993. pp. 13-22.

[Ishikawa *et al.* 1994a] M. Ishikawa, Y. Totoki, R. Tanaka, and M. Hirosawa. Multiple Sequence Alignment Editor Featured by Constraint-based Parallel Iterative Aligner. In *Proc. 3rd Int'l. Conf. Bioinformatics and Genome Research*, 1994.

[Ishikawa *et al.* 1994b] M. Ishikawa, Y. Totoki, T. Toya, M. Hoshida, and M. Hirosawa. Protein Sequence Analysis by the Parallel Iterative Improvement Method. In *Trans. Inf. Process. Soc. Jap.*, 1994.

[Ishikawa *et al.* 1994c] M. Ishikawa, T. Toya, Y. Totoki, and R. Tanaka. Multiple RNA-Sequence Alignment Considering Stem Regions. In *Proc. Genome Informatics Workshop V*, 1994.

[Jaffar and Lassez 1987] J. Jaffer, and J.-L. Lassez. Constraint Logic Programming. In *Proc. 4th IEEE. Symp. on Logic Programming*, 1987.

[Karp and Paley 1994] P. Karp, and S.M. Paley. Representation of Metabolic Knowledge: Pathway. In *Proc. 2nd ISMB*, 1994. pp. 203-211.

[Kawai *et al.* 1991] M. Kawai, A. Kishino, and K. Naito. Rapid Analysis Methodology for Gene Sequences Using a Parallel Processor. In *FUJITSU Scientific and Technical Journal*, Vol. 27, 1991. pp. 270-277.

[Kawamura *et al.* 1992] M. Kawamura, H. Sato, K. Naganuma, and K. Yokota. Parallel Database Management System: Kappa-P. In *Proc. FGCS'92*, 1992.

[Kazic 1994] T. Kazic. Biochemical Database Klotho. (http://ibc.wustl.edu/klotho/.).

[Krogh *et al.* 1994] A. Krogh, M. Brown, I.S. Mian, K. Sjölander, and D. Haussler. Hidden Markov Models in Computational Biology, Application to Protein Modeling. In *J. Mol. Biol*, Vol. 235, 1993. pp. 1501-1531.

[Letovsky and Berlyn 1994] S.I. Letovsky, and M.B. Berlyn. Issues in the Development of Complex Scientific Databases. In *Proc. 27th HICSS*, Vol. 5, 1994. pp. 5-14.

[Loui 1992] R. Loui. Computing Specificity, Research Report, WUCS-92-46, Washington Univ., 1992.

[Mamitsuka and Abe 1994] H. Mamitsuka, and N. Abe. Predicting Location and Structure of Beta-Sheet Regions Using Stichastic Tree Grammars. In *Proc. 2nd*

ISMB, 1994. pp. 276-284.

[Mühlenbein and Schlierkamp 1993] H. Mühlenbein, and D. Schlierkamp-Voosen. Predictive Models for the Breeder Genetic Algorithm: Continuous Parameter Optimization. In *Evolutionary Computation*, Vol. 1, 1993. pp. 25-49.

[Nishioka *et al.* 1994] T. Nishioka, K. Yokota, C. Takahashi, and S. Tojo. Constructing a Legal Knowledge-base with Partial Information. In *Proc. ECAI '94 WORKSHOP on Artificial Normative Reasoning*, Amsterdam, Aug. 8, 1994.

[Nitta *et al.* 1992] K. Nitta *et al.* HELIC-II: A Legal Reasoning System on Parallel Inference Machine. In *Proc. Int. Conf. FGCS'92*, 1992. pp. 1115-1124.

[Nitta *et al.* 1993a] K. Nitta *et al.* HELIC-II: A Legal Reasoning System on Parallel Inference Machine. In *New Generation Computing*, Vol. 11, 1993. pp. 423-448.

[Nitta *et al.* 1993b] K. Nitta *et al.* A Computational Model for Trial Reasoning. In *Proc. Int. Conf. on Artificial Intelligence and Law*, 1993. pp. 20-29.

[Onizuka *et al.* 1993] K. Onizuka, K. Asai, M. Ishikawa, and S.T.C. Wong. A Multi-Level Description Scheme of Protein Conformation. In *Proc. 1st ISMB*, 1993. pp. 301-310.

[Perrière *et al.* 1994] G. Perrière, F. Chevenet, F. Dorkeld, T. Vermat, and C. Gautier. Building Integrated Systems for Data Representation and Analysis in Molecular Biology. In *Proc. 27th HICSS*, Vol. 5, 1994. pp. 89-98.

[Prakken 1993] H. Prakken. Logical Tools for Modeling Legal Argument, Ph D. thesis, Vrije Universiteit, 1993.

[Rissland *et al.* 1987] E.L. Rissland *et al.* A Case-Based System for Trade Secrets Law. In *Proc. Int. Conf. on Artificial Intelligence and Law*, 1987. pp. 60-66.

[Sakakibara *et al.* 1994] Y. Sakakibara, M. Brown, R. Underwood, S. Mian, and D. Haussler. Stochastic Context-Free Grammars for Modeling RNA. In *Proc. 26th HICSS*, 1994. pp. 284-293.

[Sartor 1993] G. Sartor. A Simple Computational Model for Nonmonotonic and Adversarial Legal Reasoning. In *Proc. Int. Conf. on AI and Law*, 1993. pp. 192-201.

[Shibasaki and Nitta 1994] M. Shibasaki, and K. Nitta. Defeasible Reasoning in Japanese Criminal Jurisprudence. In *Proc. FGCS workshop on Application of Logic Programming to Legal Reasoning*, 1994.

[Takahashi and Yokota 1994] C. Takahashi and K. Yokota. A Legal Reasoning System on a Deductive Object-Oriented Database. In *Proc. 5th Int. Hong Kong Computer Society Database Workshop on New Generation Database Systems*, Hong Kong, 1994.

[Takahashi and Yokota 1995] C. Takahashi and K. Yokota. Constructing a Legal Database on *Quixote*. In *Proc. the Sixth Australasian Database Conference (ADC'95)*, Adelaide, Australia, 1995.

[Tamaki and Sato 1986] H. Tamaki and T. Sato. OLD Resolution with Tabulation. In *Proc. Int. Conf. on Logic Programming*, 1986.

[Tanaka 1991] H. Tanaka. Protein Function Database as a Deductive and Object-Oriented Database. In *Proc. Database and Expert Systems Applications*, 1991. pp. 481-486.

[Tanaka 1992] H. Tanaka. Integrated System for Protein Information Processing. In *Proc. FGCS'92*, 1992. pp. 321-329.

[Tanaka 1993a] H. Tanaka. A Private Knowledge Base for Molecular Biological Research. In *Proc. 26th HICSS*, 1993. pp. 844-852.

[Tanaka *et al.* 1993b] H. Tanaka, M. Ishikawa, K. Asai, and A. Konagaya. Hidden Markov Models and Iterative Aligners: Study of their Equivalence and Possibilities. In *Proc. 1st ISMB*, 1993. pp. 395-401.

[Tanaka *et al.* 1993c] H. Tanaka, K. Onizuka, and K. Asai. Classification of Proteins via Successive State Splitting of Hidden Markov Network. In *Proc. AI and Genome Workshop in 13th IJCAI*, 1993.

[Terasaki *et al.* 1992] S. Terasaki, D. Hawley, H. Sawada, K. Satoh, S. Menju, T. Kawagishi, N. Iwayama, and A. Aiba. Parallel Constraint Logic Programming Language GDCC and its Parallel Constraint Solvers. In *Proc. Conf. FGCS'92*, 1992. pp. 330-346.

[Tojo and Nitta 1994] S. Tojo and K.Nitta, Automated Generation of Temporal RElations in a Legal Case, In *Proc. Workshop on Legal Application of Logic Programming*, 1994, pp. 33-47.

[Yada *et al.* 1994a] T. Yada, M. Ishikawa, Y. Totoki, and K. Okubo. Statistical Analysis of Human DNA Sequences in the Vicinity of Poly(A) Signal. In *Proc. 3rd Int. Conf. on Bioinformatics and Genome Research*, 1994.

[Yada *et al.* 1994b] T. Yada, M. Ishikawa, H. Tanaka, and K. Asai. DNA Sequence Analysis using Hidden Markov Model and Genetic Algorithm. In *Proc. Genome Informatics Workshop V*, 1994.

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

[Yokota 1994a] K. Yokota. Multi-agent Based Extentions of Multidatabases. In *Joint Workshop of SIGDBS of IPSJ and SIGDE of IEICE*, 1994. (in Japanese)

[Yokota 1994b] K. Yokota. Legal Reasoning on a Deductive Object-Oriented Database and its Extension. In *Workshop on Knowledge Representation for Legal Reasoning*, Boston, USA, 1994; ICOT-TR, 1994.

[Yokota and Aiba 1994] K. Yokota, and A. Aiba. A New Framework of Very Large Knowledge Bases. In *Knowledge Building and Knowledge Sharing*, eds K. Fuchi and T. Yokoi, Ohmsha and IOS Press, 1994

[Yokota *et al.* 1993] K. Yokota, H. Tsuda, and Y. Morita. Specific Features of a Deductive Object-Oriented Database Language *Quixote*. In *Proc. ACM SIGMOD Workshop on Combining Declarative and Object-Oriented Databases*, Washington DC, USA, 1993.

[Yokota *et al.* 1994] K. Yokota, T. Nishioka, H. Tsuda, and S. Tojo. Query Processing for Partial Information Databases in *Quixote*. In *6th IEEE Int. Conf. on Tools with Artificial Intelligence*, New Orleans, 1994.

[Yokota and Yasukawa 1992] 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. In *Proc. FGCS'92*, 1992.

[Yoshida *et al.* 1992] K. Yoshida, C. Smith, T. Kazic, G. Michaels, R. Taylor, D. Zawada, R. Hagstrom, and R. Overbeek. Toward a Human Genome Encyclopaedia. In *Proc. FGCS'92*, 1992. pp. 307-320.