PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

SPHINX -- A HYBRID KNOWLEDGE REPRESENTATION SYSTEM

Sangki Han and Jung Wan Cho

Computer Science Department
Korea Advanced Institute of Science and Technology
P. O. Box 150, Cheongryang, Seoul 131-650, Republic of Korea
skhan@casun.kaist.ac.kr (Internet) ..mcvax!hallalsorak!casun!skhan (Usenet)

ABSTRACT

In this paper, a new hybrid knowledge representation system called Sphinx is presented. It consists of two major reasoners for terminological and assertional reasoning. The former is based on classification and uses a frame-based description language while the latter is a theorem prover based on Horn logic and uses an extended logic programming language. Besides a new hybrid reasoning scheme, a new knowledge base maintenance mechanism based on the negation as failure inference rule is also presented, so that Sphinx can support incremental assertions and retractions. In addition, Sphinx provides explanation capability to help the user in developing and debugging his/her knowledge bases.

1 INTRODUCTION

As a number of intelligent systems have been developed, it is recognized that most of them require various kinds of knowledge and reasoning. However, one uniform knowledge representation formalism cannot serve all representational needs required in far-flung domains. Therefore, it is naturally required that a system allows several types of knowledge to exist simultaneously and supports adequate and effective reasoning for each type of knowledge.

Recently, a new approach called *hybrid knowledge representation* scheme has been studied to meet this requirement. In typical hybrid systems, two or more fundamentally different types of representational formalism and reasoning are provided in such a way that each component acts a complementary one to one another and each reasoner makes an inference in accordance with other reasoners. Each compnent has own inference mechanism and language. In usual, an appropriate set of inferences over sentences and complex terms is supported by most of hybrid systems.

With this approach, the overall semantics for the hybrid representation formalism becomes clear and the interaction between different kinds of knowledge can be well understood and maintained. Examples of hybrid knowledge representation systems are KRYPTON (Brachman et al. 1983), KL-TWO (Vilain 1984), BACK (von Luck et al. 1987), and Loom (Mac Gregor and Bates 1987).

In this paper, a new hybrid knowledge representation system called **Sphinx** is presented. Sphinx is based on a restricted theorem prover which reasons over a subset of the first-order logic in a simple and efficient way. Like other hybrid systems, the theorem prover is augmented with a special-purpose reasoner for terminological reasoning.

Sphinx also consists of two major subcomponents each of which is termed **TBox** and **ABox**, respectively, following to KRYPTON. The TBox is for terminological reasoning and based on frames, while the ABox for assertional reasoning is a theorem prover based on Horn logic. Viewed from a different angle, Sphinx is a tightly-coupled integration of logic programming technique and classification-based reasoning (Schmolze and Israel 1983). In fact, the assertional subcomponent of Sphinx can be regarded as a kind of an executor for logic programs.

Interfacing between two components is accomplished by unifying terms in the TBox and predicates in the ABox as in other hybrid systems. Therefore, terminological knowledge and analytical inferences reflecting the structural relations in the TBox are available in the assertional reasoner.

Being allowed to express ABox queries for knowledge base (KB) in the first-order sentence, more expressiveness at least in inspecting a KB is possible, even though the KB is maintained in a

†This research is supported partially by the Ministry of Science and Technology in Korea as a national project for next generation computer systems , under contact No. N04050.

simple and limited form. The query laguage also permits the constituent (wh-) questions as well as yes-no questions.

Sphinx adopts *Nagation As Failure* (NAF) rule (Clark 1978) for inferring negative information. It offers elegant and simple inference algorithms in proving a query and revising KB. In particular, it also supports a kind of nonmonotonic reasoning which simplifies the revision of KB greatly.

Although the ABox is basically a theorem prover, incremental assertions and retractions of factual knowledge are permitted in Sphinx. When a fact is acquired or deleted, Sphinx preserve consistency by revising its KB itself. Since Sphinx records only premises and infers logical consequences whenever they are to be proved, an addition of a new fact does not cause any contradiction.

When a fact is retracted, which is equivalent to an addition of its negation owing to the NAF rule, dependency-directed backtracking is performed. As the logical consequences of a premise is not recorded but inferred, they are not inferred any longer if the premise is retracted.

Instead of recording all the dependency relations between propositions, Sphinx uses the terminological KB in the TBox which shows the dependency relations between terms which are applied to all the extensions of terms. Consequently, by referring to dependency relations between terms, Sphinx can finds out which premises have influences on the deduction of a given fact and at least one of these premises is negated if we want to retract that fact.

Besides maintenance capability, Sphinx also provides a capability to explain its deduction steps. It helps user to develop, debug, and understand his knowledge base.

Sphinx is implemented on Sun-3/160* 4.2 BSD UNIX† workstation using Quintus PROLOG‡. To demonstrate its usefulness, a sample system called SDNKB is developed for describing the status information about nodes and network configuration of our domestic computer network called SDN (System Development Network) in Korea (Chon et al. 1984). In this paper, most of examples are taken from SDNKB.

† UNIX is a trademark of Bell Laboratories.
* Sun is a trademark of Sun Microsystems, Incorporated.
‡ Quintus PROLOG is a trademark of Quintus Computer Systems, Incorporated.

2 TERMINOLOGICAL REASONING IN SPHINX

Definitions in the terminological component (TBox) serve to define the terms which consist of the vocabulary to represent knowledge in the world being modelled. Like other hybrid systems, terms in the TBox correspond to predicates in the ABox.

Like KL-TWO and KRYPTON, two major categories are used to classify terms in Sphinx: *Concept* and *Role*. A *Concept* is a representation of the unary predicate or relation representing the object in the world, while a *Role* represents the relationship between Concepts.

Sphinx also distinguishes between *primitive* and *defined* terms. A term is primitive if no definition can be given for it. Terms for natural kinds such as animal, bird, chair, and etc. are typical examples of primitive terms. Fig. 1 shows TBox operations to create the primitive terms.

(1) **defConcept**(*Concept*).
(2) **defConcept**(C_1,C_2).
(3) **defRole**(*Role*,*Domain*,*Range*).

FIGURE 1. TBox operations for primitive terms

The first and second operations are used to declare primitive Concepts. In particular, the second operation denotes the primitive specialization between two primitive Concepts.

The third operation creates a primitive Role with declarations of domain and range of the Role, respectively. The semantics of this operation can be defined as follows:

$$\forall x.\ \forall y.\ (Role(x,y) \rightarrow (Domain(x) \wedge Range(y)))$$

As is easily seen, domain and range state necessary conditions about a Role and, as a result, they function as integrity constraints.

On the other hand, defined terms are terms to which a set of necessary and sufficient conditions is given. Fig. 2 shows five different operations and their semantics to define complex terms out of primitive or defined ones.

The first operation defines a conjunction of Concepts given in the list. For example, a *unix_workstation* can be defined as a conjunction of *workstation* and *unix_machine*:

defConcept(unix_workstation,[workstation,unix_machine]).

(1) **defConcept**$(C,[C_1, \cdots ,C_n])$.
$\forall x.\ (C(x) \leftrightarrow C_1(x) \wedge \cdots \wedge C_n)$

(2) **defConcept**(C,C_1,R,C_2).
$\forall x.\ (C(x) \leftrightarrow C_1(x) \wedge \forall y.\ (R(x,y) \rightarrow C_2(y)))$

(3) **defConcept**$([C_1, \cdots ,C_n],C)$, where $n \geq 2$.
$\forall x.\ (C_i(x) \rightarrow C(x)) \wedge \forall x.\ (C_i(x) \rightarrow \sim C_j(x))$,
where $2 \leq i,\ j \leq n$ and $i \neq j$

(4) **defConcept**$(C,[C_1, \cdots ,C_m],$
$[R_1 : RC_1, \cdots ,R_n : RC_n])$.
$\forall x.\ (C(x) \leftrightarrow (C_1(x) \wedge \cdots \wedge C_m(x)) \wedge$
$\forall y_1.\ (R_1(x,y_1) \wedge RC_1(y_1)) \wedge$
$\forall y_2.\ (R_2(x,y_2) \wedge RC_2(y_2)) \wedge$
$\vdots$
$\forall y_n.\ (R_n(x,y_n) \wedge RC_n(y_n)))$
where C_i and RC_j are Concepts.

(5) **defRole**$(R,[R_1,R_2])$.
$\forall x.\ \forall y.\ (R(x,y) \leftrightarrow \exists z.\ (R_1(x,z) \wedge R_2(z,y)))$

FIGURE 2. Operations to define complex terms

The second operation is to specialize a Concept by restricting its Role value. Concepts appearing for value restriction are called *Value-Restrictive Concept* (VRC)s. For example, when we want to define a *'X.25_network'* as a *network* whose *protocol* is *'X.25'*, it is expressed as follows:

defConcept('X.25_network',network,protocol,'X.25').

The third one means that a Concept C is partitioned into $C_1, \cdots ,C_n$. This kind of definition is very useful in checking the coherency of knowledge being acquired. According to these definitions, there cannot be a common subsumee of disjoint Concepts. In assertional aspect, no individual can be an instance of disjoint Concepts simultaneously. For example, a *gender* is partitioned into *male*, *female*, and *neutral*:

defConcept([male,female,neutral],gender).

The fourth operation is a combination of previous operations to define a more complex Concept at once. The second argument is a list of superConcepts and the third argument is a list of Role restrictions.

Finally, the last operation defines a chained Role which is a relational composition of Roles. For example, the *grandchild* Role can be defined as the chain of two *child* Roles. How a chained Role affects the assertional reasoning will be shown in the next section.

The most basic relation between terms is a subsumption relation (Brachman and Levesque 1984) since most of reasoning in the TBox is based on *classification*. To classify a term means to place it on its proper location in the TBox network, so that it is below all more general terms than it and it is above all terms which specialize it (Brachman and Schmolze 1985). The set of direct subsumers of a term is called the *most specific generalization* (MSG) and the set of direct subsumees is called the *most general specialization* (MGS).

To classify a term, we must determine its MSG and MGS by examining subsumption relations between other terms when it is added newly to the TBox. Subsumption relation is defined as follows:

Definition 1: Subsumption

Concept C_1 subsumes C_2 if and only if every extension of C_2 is an extension of C_1, i.e.:

$$\models \forall x.\ (C_2(x) \rightarrow C_1(x))$$

As Brachman and Levesque pointed out (Brachman and Levesque 1984), there cannot be an efficient algorithm for subsumption even for a very restricted frame-based description language. As a result, we must select term-forming operations carefully and precisely to hold the efficiency as KRYPTON did or sacrifice the completeness of the subsumption algorithm as NIKL did. Sphinx follows the second way for more expressiveness power which turns out to be more useful in practice.

Besides term-forming operations, there are a set of operations in the TBox for queries inquiring the structural and analytic nature of the TBox network. Nebel suggested a set of useful and fundamental queries for a terminological reasoner in (Nebel 1987). They are queries for *subsumption*, *classification*, *disjointness*, *incoherence*, and *property possesion*. Most of them are defined in Sphinx as various TBox queries. Fig. 3 shows fundamental TBox queries designed for this purpose.

An important chracteristic of this kind of TBox queries is that they may be used as constructive queries as well as simple yes/no queries, since the TBox language itself is a logic programming language. That is, if a query contains variables, correct answers for these variables are substituted, provided it is proved successfully.

(1) **subsume**(T_1, T_2)
(2) **findConcept**(X,[$C_1, \cdots, C_m$], [R_1:$RC_1, \cdots, R_n$:RC_n])
(3) **find_MSG**(T,MSG)
(4) **find_MGS**(T,MGS)
(5) **disjoint**(T_1, T_2)

FIGURE 3. Queries to inspect KB in the TBox

The first operation is to see if a subsumption relation between two terms, T_1 and T_2, holds. For example, consider *mailing_list_service* and *mail_transfer* defined as follows:

```
defConcept(mailing_list_service,network_service,
        transfer_object,mailing_list).
defConcept(mail_transfer,network_service,
        transfer_object,mail).
```

From these definitions, we can conclude that *subsume(mail_transfer,mailing_list_service)* holds if *mail* subsumes *mailing_list*.

The second operation, *findConcept*, is the most complex and general query. It inquires if X is the conjunction of a set of the given Concepts as the second argument and has a set of pairs of Roles and VRCs. It is very useful operation in classification-based reasoning since it finds out MSG satisfying those constraints. In addition, it reflects another fundamental terminological reasoning called the *property inheritance* as some VRCs may be inherited from superConcepts.

The third and fourth operations are primitive operations for classification-based reasoning. If a new term is introduced to the TBox, a facility called the *classifier* invokes these two operations to find out the proper location of the given term in the TBox network.

The *disjoint* operation finds out if two given terms are mutually exclusive. The disjointness between terms are established when a set of terms partitions a term or is defined by refering to disjoint terms. Checking out the disjointness between terms helps to maintain the consistency of the TBox network.

3 ASSERTIONAL REASONING IN SPHINX

The assertional component of hybrid systems usually performs recording and reasoning about extensions of the terminological knowledge represented in the TBox. Although first-order logic is the most appropriate tool for assertional reasoning and has powerful expressiveness, the full first-order theorem prover is prone to time-consuming search and its expressiveness is too powerful compared to that of the terminological reasoner.

What one really wants from the assertional component of hybrid systems may be summarized as follows. They are also the design aims of the assertional component of Sphinx.

1. It can reason over formulas consisting of the extensions of the TBox terms.
2. It can prove quantified formulas with variables.
3. It can prove and answer a given query with correct answer substitution.
4. It should be possible to add a fact in any order and retract it at any time with consistency.

These aims are realized partially by restricting the ABox such that only Horn clauses are represented explicitly and no explicit negation is allowed. Since there is no function in the ABox and the NAF rule is adopted, we can prove a query and revise KB in a simple and efficient way.

3.1 Knowledge Represented In The ABox

Three kinds of knowledge are stored and manipulated in the ABox, while these are represented in two different forms; facts and rules. The first one is a set of propositions asserted by the user. We call them *ABox facts* or *premises* and they denote instances of terms in the TBox. Following are some examples of ABox facts.

```
    :
unix_workstation(casun).
system_V_machine(cygnus).
mailing_list(neuron).
os(casun,'Sun UNIX 4.2').
moderator(ailist,skhan).
    :
```

Like other hybrid systems, predicate symbols appearing in the ABox are related to terms in the TBox. That is, 1-place predicate symbols and 2-place predicate symbols correspond to Concepts and Roles, respectively.

The hybrid reasoning of Sphinx can be accomplished in principle by unifying predicate symbols with terms. In this way, terminological knowledge

in the TBox is available in assertional reasoning and thus the meanings of sentences in the assertional component can be affected by it. Moreover, some logical consequences of KB which are not inferred by a resolution-based reasoner alone with ABox facts can be deduced by considering TBox information.

Although an assertion for a Concept is accomplished completely only if it is consistent with a KB, assertions corresponding to extensions of Roles may require additional deductions due to their definitions or usages. First, as a Role instance must satisfy the domain and range contraints, it must be confirmed that individuals appearing as arguments are instances of Concepts corresponding to the domain and range when it is introduced.

Second, when a Role is used in specializing a Concept or it is referred in defining another chained Role, its instances may induce inferences of new instances of VRC or derivations of chained instances.

For example, assume that the current KB contains the following facts:

```
unix_machine(aroma).
os(aroma,'Sun UNIX 4.2').
```

then it must be provable that *'Sun UNIX 4.2'* is an instance of *unix_os* if *unix_machine* is defined as a *computer* whose *os* is a *unix_os*. However, there is no way to prove it in the ABox alone as this inference is possible from the definition of *unix_machine*.

For this, KRYPTON extends the unification procedure of its theorem prover to reflect terminological knowledge (Brachman et al. 1985). Other systems like KL-TWO (Vilain 1985) and BACK (Nebel and von Luck 1987), make additional inferences whenever a new assertion for a Role is acquired and record the justifications for the inference if necessary.

In Sphinx, an additional axiom is added to the ABox to infer those facts when a term is defined. The additional axiom is the second type of knowledge stored in the ABox. When a Role is used to define a specialized Concept and its value is restricted to some Concept or when a Role is defined as a chain of Roles, a deduction rule called the *ABox rule* is inserted to the ABox.

For example, we can infer *unix_os(aroma)* from the above facts, provided the following rule is added to the ABox:

$$unix\_os(X) :- os(Y,X) \ \&\ unix\_machine(Y).$$

Although the ABox rule is written in Prolog-style, it is not proved by Prolog but by the ABox reasoner, i.e., a query evaluation procedure. It is because subgoals of a rule may be proved by both the ABox and TBox as other top-level queries.

The last type of knowledge in the ABox is also the deduction rule which is asserted by the user, not the system. It is useful to represent a general relation among several objects or a relation which is more natural to be represented by a rule rather than a Concept or Role. For example, a *file_transferable* relation between two hosts may be defined simply as follows:

$$file\_transferable(X,Y) :- uucp\_based(X,Y) \ or \ ftp\_based(X,Y).$$

This rule denotes that a host can transfer its files to another host when they use the uucp protocol or ftp protocol. Of course, the predicate symbol *file_transferable* should not appear as a term in the TBox.

The equality between individuals in the ABox is handled by a similar method in the RUP system. Sphinx constructs a congruence class for each equality relation and any predicate satisfied by an element in one congruence class must be satisfied by other members in that congruence class. When an equality relation is asserted, two congruence classes for each individual referred by it are merged into one.

3.2 Knowledge-Level Operations in the ABox

Knowledge-level (KL) operations defined in the ABox are *ask* and *tell* operations. The *ask*(Q) operation asks if a query Q can be proved from current KB involving the TBox network and the ABox KB. It also returns the correct answer substitution if the query is provable. On the other hand, the *tell*(K) operation is used to assert a new knowledge to KB when a positive fact is asserted or to retract existing knowledge when a negative one is asserted.

Below, the formal definitions of *ask* and *tell* operations are given as Definition 2. In this definition, the following notations are used.

Notation 1:
KB : A current KB in Sphinx
KB^+ : KB after applying *ask* or *tell* operation
$KB \vdash \alpha$: α is deducible from KB
$KB \nvdash \alpha$: α is not deducible from KB[1]

Definition 2: *ask* and *tell*

1. ask(ω):
   If there exists a substitution θ, such that $KB \vdash \omega\theta$, where ω is a well-formed formula, then returns {yes,θ},
   Otherwise, returns {no}
2. tell(α):
   (i) When α is a positive literal, returns $KB^+ = KB \cup \{\alpha\}$ if $KB \nvdash \alpha$
   (ii) When α is a negative literal, say *not*(κ), and
   if $KB \vdash \kappa$, returns KB^+ such that $KB^+ \subseteq KB$ and $KB^+ \nvdash \kappa$
   if $KB \nvdash \kappa$, returns KB^+ such that $KB^+ = KB$
   (iii) When α is a rule, returns $KB^+ = KB \cup \{\alpha\}$ if it is not contained in the current ABox.

In this definition, *ask* operation returns a pair of a truth value and an answer substitution. This means that the ABox query evaluation procedure does not only prove a query but also returns an answer if it is satisfied. Instantiation for quantified queries is performed by either the ABox through the answer substitution or the TBox reasoner through classification and inheritance.

The formula used in the *ask* operation is written in the ABox query language. It is a pure first-order language involving equality. It also permits explicit quantifiers and logical connectives for representational expressiveness. In this respect, it is a kind of extended logic programming language similar to that presented in (Lloyd and Topor 1984). Followings is the informal syntax used to express the ABox query. More formal definitions can be seen in (Han et al. 1987).

P & Q - Conjunction of P and Q
P or Q - Disjunction of P and Q
not(P) - Negation of P
forall(X,P) - Universal quantification
some(X,P) - Existential quantification
c <=> d - constant c is equal to constant d

Following are some examples of the *ask* operations.

```
?- ask(csnet_gateway(sorak) & protocol(sorak,P)).

   P = mmdf
   /* Is sorak a CSNET gateway and what is its
   protocol? */

?- ask(some(X,(unix_machine(X) &
   not(workstation(X))))).

   X = sel
   /* Is there a UNIX machine which is not a
   workstation? */
```

The second KL operation, *tell*(K), is used to assert a new knowledge expressed as K to a KB. Because of the NAF rule, the addition of a new fact can prevent a theorem that previously held from proving. In particular, when a positive literal L is asserted, $\sim L$ can be no longer proved. Hence occurrences of inconsistencies raised by the coexistence of a positive fact and its negation simultaneously in a KB will not happen in Sphinx.

On the contrary, if K is a negative literal, say $\sim L$, it plays a role of retracting a set of knowledge stored in the ABox since $\sim L$ holds if L must not be inferred. It invokes the truth maintenance module in Sphinx for consistency maintenance. More details on effects of asserting or retracting knowledge will be considered and explained in Section 4. Following are some examples of the tell operations.

```
?- tell(unix_workstation(cosmos)).
?- tell(not(ms_dos_machine(cygnus))).
?- tell(csd <=> cskaist).
```

The *ask* and *tell* operations can be connected through a couple of connectives: "," for conjunction and ";" for disjunction.

3.3 Query Evaluation Procedure (QEP)

In Sphinx, a query is not proved by the SLD-resolution alone but by a kind of the theory resolution (Stickel 1985) with the TBox theory, i.e., the terminological definitions. In addition, in order to prove a negated or universally quantified formula, elements in the current domain are examined as need. For this, the following assumption is made:

Assumption:
There is no function in Sphinx. This means that the Herbrand Universe of the ABox is a finite set of the constants appearing currently in the ABox. It is constructed incrementally as a new constant is introduced to the ABox.

With this assumption, it is possible to prove a universally quantified formula with respect to the domain, i.e., the formula is proved if it is satisfied for all members in the current domain.

For the negative information, the so-called Negation As Failure rule is used in Sphinx as other logic programming languages. With the above assumption, we can remove the safe restriction of the NAF rule (Lloyd 1984). As the domain is finite, we can discover which individual satisfies a negative goal when the goal contains variables.

The basic unit in the query evaluation is a literal except the universal quantified formula. Let ω be a given query. This query is transformed into conjunctions or disjunctions of the basic units by a similar method to the meta-interpreter technique (Sterling and Shapiro 1986). With this transformation, subgoals to be proved are classified into three categories: positive literal, negative literal, and universally quantified formula.

After transforming the query, the query evaluation procedure (QEP) is executed. QEP selects positive subgoals preferentially so that the arguments of the negative subgoal can be instantiated as many as possible. By this strategy, we can reduce the overhead of proving the negative literal on the basis of the model. In QEP, we use the following notations:

Notation 2:
$\mathbf{KB}_{\mathbf{ABox}}$: KB in the ABox

$\mathbf{KB}_{\mathbf{TBox}}$: KB in the TBox

KB : Whole KB in Sphinx
$\Gamma(C)$: A Concept corresponding to a predicate symbol C
$C_1 < C_2$: C_1 is subsumed by C_2
$\mathbf{D}$: A set of individuals appeared in the ABox (Domain)

Procedure QEP

Let G_i be the current subgoal selected.

1. If there is an ABox fact in $\mathbf{KB}_{\mathbf{ABox}}$ with which G_i is unified using a substitution θ, then return $\{yes,\theta\}$.
2. If there is an ABox rule such as $Head :- Body$, where $Head$ is unified with G_i using a mgu θ, then prove $Body\theta$ by QEP.
3. If G_i is a universally quantified formula, say $forall(X,F_i)$, prove $F_i\{i/X\}$ for each member $i \in \mathbf{D}$. If it is successful for all members, return $\{yes,\theta\}$, where θ is the answer substitution for other free variables in F_i.
4. If G_i is a ground positive literal, then do the followings:
   4.1 If G_i has a form of $P(a)$, i.e., a ground 1-place predicate:
      4.1.1 Determine if there is a fact $P(b)$ in the $\mathbf{KB}_{\mathbf{ABox}}$ where a and b are members of the same congruence class. Then, return $\{yes,\{\}\}$.
      4.1.2 If there exists a Concept C such that $C(i)$ holds for some $i \in \mathbf{D}$ and $i <=> a$ and $\Gamma(C) < \Gamma(P)$. Return $\{yes,\{\}\}$ if successful.
      4.1.3 Otherwise, obtain the definition of $\Gamma(P)$ and transform it into a query involving a. Prove this query by QEP and return $\{yes,\{\}\}$ if succeeded.
   4.2 If G_i has the form of $a <=> b$, return $\{yes,\{\}\}$ if a and b are in the same congruence class. Otherwise, return $\{no\}$.
   4.3 If G_i has the form of $R(a,b)$, i.e., an instantiated 2-place predicate, then prove if there exists a $R'(a,b)$ such that $\Gamma(R')$ is subsumed by $\Gamma(R)$. Or, test if $R(a',b')$ can be proved for some a', $b' \in \mathbf{D}$ such that $a' <=> a$ and $b' <=> b$. Return $\{yes,\{\}\}$ if successful, otherwise $\{no\}$.
5. If G_i is a positive literal with variables, prove it by the following steps:
   5.1 If G_i has a form of $P(X)$, prove if there exists $i \in \mathbf{I}$ and C such that $C(i)$ is provable and $\Gamma(C) < \Gamma(P)$. If successful, $\{yes,\{i/X\}\}$ is returned.
   5.2 If G_i has a form of $R(X,Y)$, prove if $R(a,b)$ is deducible from KB for some a, $b \in \mathbf{D}$. If successful, $\{yes,\{a/X, b/Y\}\}$ is returned.

6. If G_i is a ground negative literal, say $not(P(a))$, prove if $KB \vdash P(a)$. If so, {no} is returned, otherwise return {yes,{}}.
7. If G_i is a negative literal and contains variables, say $not(P(X))$ or $not(P(X,Y))$, find $i,j \in \mathbf{D}$ such that $P\{i/X,j/Y\}$ fails. If there are such elements, return {yes,$\{i/X\}$} or {yes,$\{i/X, j/Y\}$} for each case.

QEP is very simple because that functions and recursive definitions are not allowed in the KB. We can avoid the recursive definition in the KB as we prohibit the cyclic definition of terms in the TBox and rules in the ABox.

4 TRUTH MAINTENANCE

In order for a knowledge representation system not to be just a mechanised reasoner, it should be able to maintain the represented knowledge, i.e., it ought to be possible to add and retract knowledge at any time and in any order. However, maintenance of the represented knowledge essentially requires the inclusion of a nonmonotonic reasoning capability into a knowledge representation system.

One approach to handle this problem is to construct the problem solver with a truth maintenance system like TMS (Doyle 1978), RUP (McAllester 1982), and ATMS (de Kleer 1986). However, TMS-like truth maintenance systems can be hardly used in a system requring a large KB since a great amount of space would be needed to maintain almost all deductive consequences.

We may resolve this problem by building a theorem prover being able to re-form its theory to reflect effects of addition or retraction of knowledge. Furthermore, it is quite unreasonable to record deducible facts even though they are inferrable from the current set of user-asserted facts as logic is used as the foundation of the assertional component in the hybrid systems.

Truth maintenance in Sphinx is performed by the re-formation of the ABox. To be effective, however, only literals - positive and negative facts - are allowed to be added to or retracted from the ABox. Among these facts, Sphinx distinguishes user-asserted facts called *premises* from deduced facts, i.e., logical consequences of premises.

In general, when a fact is asserted, it may cause an inconsistency with existing KB. In Sphinx, however, an addition of a positive fact is regarded as a denial of its complementary negative fact. Since a negative fact holds if its complementary fact is not proved, it is no longer proved if its complementary fact is added to the ABox. Therefore, there is no possibility that a fact and its negation can exist simultaneously. In addition, additions of negative literals are equivalent to retractions of their positive complements because of the NAF rule. Hence we can consider the truth maintenance procedure of Sphinx only in two phases. The first phase is the addition of a new positive fact and the second one is the retraction of an existing fact.

We can categorized the situation when a fact F is added to the ABox into 4 cases according to QEP:

(1) It exists explicitly in the ABox (a *premise*).
(2) It is proved by the definition of the predicate of F
(3) It is proved by a subsumption relation.
(4) It is proved by a deduction rule in the ABox.

Of course, when F is asserted, it is neglected in the first case because it already exists. However, facts in the second class are not recorded as premises, even though they are asserted by the user. This kind of facts is called the *defined facts*. For example, *unix_workstation*(*casun*) can be inferred from two premises as follows since *unix_workstation* is defined as a conjunction of *workstation* and *unix_machine*.

workstation(casun).
unix_machine(casun).

In this case, as *unix_workstation*(*casun*) and a conjunction of the two premises are equivalent, it is unnecessary to record it explicitly even if it is asserted by the user. Furthermore, an inconsistency may occur when one of them is retracted, if all the premises are recorded.

The third and fourth type of facts are called the *deducible facts*. The third type of facts means implicit facts inferred by subsumption relations while the fourth type is inferred from premises and deduction rules. The deducible fact is not recorded explicitly in the ABox unless it is asserted by the user, so that it can be denied automatically if one of its supporting premises is denied.

Even though a fact is deducible from a KB, it is recorded explicitly if the user asserts it. In this case, a deducible fact becomes a premise. Once a fact is asserted by the user, it is negated by only

the user. Therefore, unless a fact is a defined fact, it is not retracted even if premises which justified its proof previously are negated.

With these notions, incremental additions of the ABox facts can be performed very easily. Procedure TELL_POSITIVE_LITERAL shows the whole procedure.

Procedure TELL_POSITIVE_LITERAL

If the argument α of a *tell*(α) operation is a positive literal, the following procedure is executed.

1. If α already exists in the ABox as a premise, return.
2. If α can be proved by the definition of a term corresponding to the predicate symbol of α, return.
3. Otherwise, add α to the ABox and retract premises which become defined facts by introducing α.

Although TELL_POSITIVE_LITERAL pursues the minimization of the space required to store the assertional knowledge, some redundancies according to an order of assertions may arise. For example, premises may become defined facts as a certain fact is asserted and, as a result, those premises must be removed from the ABox as shown in Step 3. Such premises are found out by tracing dependency links between a term and other terms whose definitions are based on the former in the TBox.

Owing to the NAF rule, when an argument of a *tell* operation is a negative literal, it is interpreted as a retraction of its complementary positive literal. In order for a knowledge representation system to preserve consistency, the logical consequences of a fact must be also retracted when a fact is retracted. In Sphinx, the retraction of knowledge can be examined in two respects.

If a fact to be retracted is a premise, it is necessary to retract the fact itself without considerations on logical consequences as they are not recorded unless the user asserted them. However, this is not enough. There may be premises which hold for subterms of the term for which the given premise satisfies. If such occasion arises, those premises must be also retracted when a premise is retracted. This procedure is applied to all subterms in turn.

Similarly, a retraction of a nonexisting deducible fact or a defined fact causes retractions of premises which justify the fact to be retracted. Therefore, it is important to find out which premises are related to a given premise. These premises are called the *justification* of a fact and there are a set of justifications for a fact. The justification of a fact is determined on the basis of the terminological definition of the term for which a fact is instantiated.

Consequently, the following procedure is applied when a fact is retracted from the ABox. More detail description can be found in (Han et al. 1987). The time required to determine the justification of a given fact is proportional to the time to prove it within a constant factor.

Procedure TELL_NEGATIVE_LITERAL

If an argument of the *tell* operation is a negative literal, say $\sim\alpha$, the following procedure is performed.

1. If α is not proved, return.
2. If α is a premise, do the followings and return:
   2.1. Retract α.
   2.2. Retract premises holding for terms which are subterms of a term for which α satisfies.
3. If α does not exist but holds, find the justification of α.
4. Deny each element of the justification by asking the user which premise in that element be denied.

When an intelligent system makes a decision, it must be able to explain how the decision is made. In Sphinx, as the justification for a deduction is determined not only by the ABox facts and rules but also by the TBox structure, the explanation is generated by considering all of them. The explanation facility of Sphinx shows how a given query can or cannot be proved by presenting inference steps one by one.

Hybrid systems can provide appropriate representational formalisms and reasoning schemes for multiple types of knowledge. In compensation for sacrificing the expressiveness of the first-order logic to some extent, more computational efficiency can be obtained by augmenting special purpose reasoners with the conventional theorem prover. Hybrid KR systems can provide well-defined and more precise semantics to the intelligent systems

since semantics of KL operations and the interaction among multiple reasoners are defined precisely.

Sphinx is also a kind of the hybrid system consisting of two major components. We proposed an efficient combination of a logic programming language with a frame-based description language so as to perform the hybrid reasoning. We found out that logic programming is very useful and efficient in the representation and reasoning of assertional knowledge in the hybrid systems particularly.

Sphinx is an evolving KR system. Various enhancements are investigated in several aspects including extensions of the TBox operations for more expressive power, augmentations of more specialized reasoners, and extensions of the ABox language to be able to represent and express more complex sentences.

REFERENCES

Brachman, R. J., Fikes, R. E., and Levesque, H. J. 1983. Krypton: A Functional Approach to Knowledge Representation. *IEEE Comput* October:67-73.

Brachman, R. J., and Levesque, H. J. 1984. The Tractability of Subsumption in Frame-Based Description Languages. *Proc. of AAAI-84*, pp. 34-37, August 6-10, Austin, Texas.

Brachman, R. J., Pigman-Gilbert V., and Levesque, H. J. 1985. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton. *Proc. of IJCAI-85*, pp. 532-539, August 18-23, Los Angeles.

Brachman, R. J., and Schmolze, J. G. 1985. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* June:171-216.

Chon , K., Kim C. S., Cha, E. Y., and Chung, S. K. 1984. System Development Network. *Proc. of TENCON*, April, Singapore.

Clark, K. L. 1978. Negation as Failure. In *Logic and Data Bases*, eds. H. Gallaire and J. Minker, pp. 293-322. New York:Plenum Press.

de Kleer, J. 1986. An Assumption-based TMS. *Artificial Intelligence* March:127-162.

Doyle, J. 1978. Truth Maintenance Systems for Problem Solving. AI-TR-419, MIT AILAB, Cambridge, Mass.

Han, S., Shin, D. W., Kim, Y., Jun, Y. P., Maeng, S. R., and Cho, J. W., 1987. A Logic Programming Approach To Hybrid Knowledge Representation, CAL-TR-008, CALAB, KAIST, Seoul, Korea (will appear in *Applied Artificial Intelligence - An International Journal*).

Lloyd, J. W. 1984. *Foundations of Logic Programming*, Berlin:Springer-Verlag.

Lloyd, J. W. and Topor, R. W. 1984. Making Prolog More Expressive. *Journal of Logic Programming* October:225-240.

von Luck, K., Nebel, B., Peltason, C., and Schmiedel, A. 1987. The Anatomy of the BACK system. KIT-REPORT 41, Technische Universitat Berlin, Berlin, Germany.

Mac Gregor, R., and Bates, R. 1987. The Loom Knowledge Representation Language. ISI/RS-87-188, USC/ISI, University of Southern California, Marina del Rey, CA.

McAllester, D. A. 1982. Reasoning Utility Package User's Manual. AI Memo 667; MIT AILAB, Cambridge, Mass.

Nebel, B. 1987. Computational Complexity of Terminological Reasoning In BACK. KIT-REPORT 43, Technische Universitat Berlin, Berlin, Germany.

Nebel, B., and von Luck, K. 1987. Issues of Integration and Balancing in Hybrid Knowledge Representation Systems. KIT-REPORT 46, Technische Universitat Berlin, Berlin, Germany.

Schmolze, J. C., and Israel, D. J. 1983. KL-ONE: Semantics and classification. In *Research in knowledge representation and natural language understanding -- Annual Report, 1 September 1982 - 31 August 1983*, ed. C. L. Sidner, pp. 27-39. Bolt Beranek and Newman Report No. 5421.

Sterling, L., and Shapiro, E. 1986. *The Art of Prolog: Advanced Programming Techniques*. Cambridge:The MIT Press.

Stickel, M. E. 1985. Automated Deduction by Theory Resolution. *Proc. of AAAI-85*, pp. 1181-1186, August 18-23, Los Angeles.

Vilain, M. 1984. KL-TWO, A Hybrid Knowledge Representation System. In *Research in Knowledge Representation for Natural Language Annual Report BBN Report 5694*, ed. C. L. Sidner, pp. 9-29, Cambridge..

Vilain, M. 1985. The Restricted Language Architecture of a Hybrid Representation System. *Proc. of IJCAI-85*, pp. 532-539, August 18-23, Los Angeles.