Knowledge acquisition mechanisms for a logical knowledge base including hypotheses

Mitsuru Ishizuka and Tetsusi Matsuda*

A hypothesis-based reasoning system handles a knowledge base including complete (fact) and incomplete (hypothesis) knowledge. The handling of the incomplete knowledge plays an important role in realizing advanced AI functions such as commonsense, flexible matching, learning, etc. From both theoretical and practical viewpoints, the hypothesis-based logical reasoning system can be considered to be an important start point towards a next-generation knowledge base architecture. This paper describes an enhanced knowledge representation of the hypothetical reasoning system designed particularly for interactive diagnosis problems. The mechanisms of two knowledge acquisition modules developed for this knowledge base including hypotheses are then described. The first module, which is based on an inductive inference mechanism for the knowledge including hypotheses, enables multiple-concept formation from given examples. The key technology of this mechanism is a minimum generalization extended to the knowledge including hypotheses. The second module provides knowledge assimilation and management functions for the frame knowledge base constructed on the hypothetical reasoning system. That is, this module enables the following functions: (1) the system can assimilate new knowledge while maintaining the consistency and nonredundancy of the knowledge base; (2) the system rearranges the knowledge base when existing knowlege is deleted; and (3) the system can control the inheritance link of the frame knowledge base in response to an input indicating that the property inheritance from an upper frame is denied as an exceptional case. The nonmonotonicity of the hypothetical reasoning system is considered in these mechanisms. Shapiro's logical debugging algorithm is employed effectively to identify the knowledge which causes inconsistency. The whole system has been implemented using the meta-programming of Prolog.

Keywords: logical knowledge base, hypothetical reasoning, knowledge assimilation, knowledge management, inductive inference

Today's knowledge systems are mostly based on the deductive inference mechanism which has thus far been explored in the artificial intelligence field. An important theme in the development of future knowledge systems is to explore advanced AI mechanisms such as recognition, analogy, learning, creative functions, abduction, etc. One key area is the handling of incomplete knowledge in addition to complete knowledge in a knowledge base. This incomplete knowledge includes hypothetical knowledge, knowledge with exceptions, commonsense knowledge which works in the background in case of need, the knowledge which interpretation can generalize in case of need, etc. If we write these sorts of incomplete knowledge in a knowledge base, the incomplete knowledge becomes false in some situations, and conflicting answers may be deduced from the knowledge base. However, the incomplete knowledge greatly enhances the flexibility of the knowledge base, and becomes a key factor in the realization of advanced AI mechanisms.

As one step towards an advanced knowledge base capable of handling incomplete knowledge, this paper will deal with a hypothesis-based logical reasoning system, since this is important from the viewpoints of applicability to practical diagnosis and design problems as well as theoretical foundation. A hypothetical reasoning system in the logic framework has been proposed by Pool *et al.*[1]. In this paper it will be presented as an enhanced knowledge representation in a hypothesis-based logical reasoning system suitable for diagnosis or classification problems. An inductive inference module will then be presented which generates knowledge, including hypotheses (incomplete knowledge), from given examples. Also described is a

Institute of Industrial Science, University of Tokyo 7-22-1, Roppongi, Minato-ku, Tokyo 106, Japan
(*Presently with Sumitomo Electric Corp. Ltd)

Paper received 11 November 1988. Accepted 13 September 1989

 0950-7051/90/020077-10

knowledge assimilation and management module for a knowledge base including hypothesis knowledge. The mechanisms described in this paper have been implemented as a meta-interpreter on Prolog. Prolog-like notations, such as ':–', are used throughout the paper.

ENHANCED KNOWLEDGE REPRESENTATION

The hypothetical reasoning system on the logic framework is described in Reference 1, where the fragments of knowledge are classified into two sets: facts (complete knowledge) and hypotheses (incomplete knowledge). The set of facts F represents the knowledge which is always true in a problem domain, while the set of hypotheses H represents the knowledge which is not always true and sometimes contradicts other knowledge. Figure 1 shows the conceptual architecture of the hypothesis-based logical reasoning system.

When a set of observations O is given, the basic mechanism of the hypothetical reasoning system tries to construct h, which is the subset of H (i.e., $h \subseteq H$) satisfying the following logical relations:

$F \cup h \vdash O$ (O is deducible from F∪h)
$F \cup h \not\vdash \square$ (F∪h is not inconsistent)

That is, h is a set of consistent hypotheses to explain the observation O. This formalism is also called consistent theory formation, which is achieved through a logical theorem proving procedure. It is shown that default logic in the scope of normal default[2] can be dealt with in the framework of this hypothesis-based logical reasoning[1]. It is important practically that the above formalism is well fitted to logical diagnosis or design problems if we regard the possible causes of the fault or the possible components of the design as hypotheses.

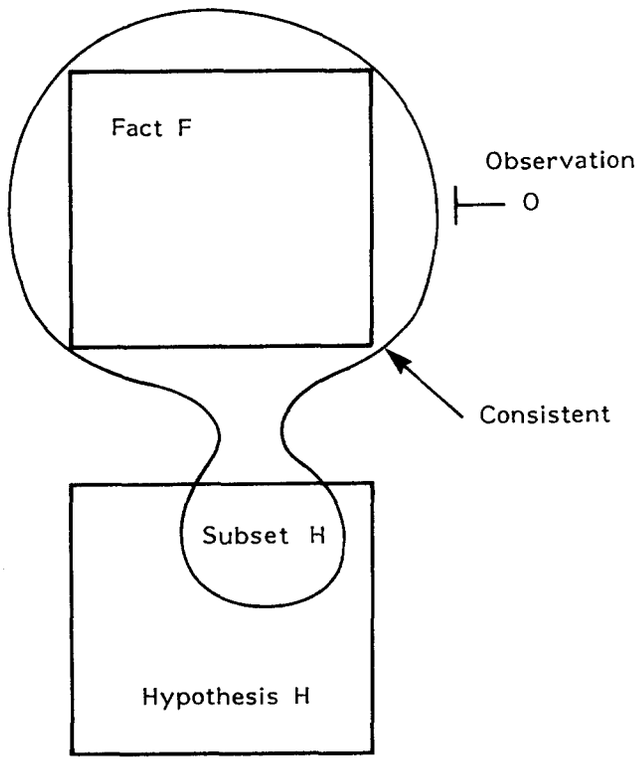


Figure 1. Hypothesis-based logical reasoning system

More than two subsets of hypotheses often satisfy the above condition, so that a process of selecting one subset of hypotheses must be invoked. This process generates a series of critical questions to distinguish one subset from the others. The use of hierarchical relations among the hypotheses in the hypothesis formation and selection processes is studied in Reference 3.

Although consistency checking is an essential function in hypothetical reasoning, it cannot be simply dealt with using Prolog, in which the scope of knowledge representation is restricted to Horn clauses and logical negation cannot be expressed. The MESON proof procedure[4] provides a way to implement full first-order predicate logic using Prolog, and since this paper emphasizes theoretical aspects, a hypothesis-based logical reasoning system will be constructed around such an implementation. (One problem of using the MESON proof procedure is its slow inference speed). The inconsistency in this case is, therefore, the situation that P and $\neg P$ co-exist in the same world under consideration. The logical negation in the current system is expressed as n( ), in contrast to not( ) which denotes the negation as failure in usual Prolog.

Considering the application of the hypothetical reasoning system particularly to diagnosis problems, the knowledge representation format is defined as:

fact (ID, NID, PREC, logical-form knowledge)
  for fact knowledge.
hyp(Name, ID, NID, PREC, logical-form knowledge)
  for hypothesis knowledge.

where

ID = identifier of the knowledge (usually numerical).
NID = list of ID which cannot be used together with its knowledge. This is used to express exclusive knowledge.
PREC = preconditions that have to be set before the use of the knowledge.
Name = name of hypothesis. This takes a predicate form in which arguments denote the variables appearing in the hypothesis knowledge.

The following shows some examples of this knowledge representation format:

```
fact(1,[2],[ ],(q:-p))
fact(2,[1],[ ],(r:-p))
hyp(nm1,3,[ ],[temp_point1(T1)],(T1>80:-p))
```

The ID, NID and PREC are introduced to express and utilize the knowledge of (⟨symptoms⟩:-⟨cause⟩) type, particularly in diagnosis problems.

PREC is introduced to treat the case that, for example, if part_A is a fault (fault(part_A)), then the temperature of point1 is higher that 80 °C. The knowledge may be interpreted as:

T1 > 80:–fault(part_A) where temp_point1(T1)

Here, the 'where' part denotes the precondition that

the knowledge becomes usable. If we express it in logic, it becomes:

T1>80:–fault(part_A),temp_point1(T1)

In the application of the hypothetical reasoning system, it is desirable to express the knowledge uniformly as:

⟨observations⟩:–⟨hypothesis⟩

To maintain this uniformity, we express the preconditions of the 'where' part in PREC in our knowledge format. This separation of the knowledge expression can be effectively utilized in the hypothesis selection process. When the knowledge seems to be useful in the inference process and the truth value of PREC is not known, the system first asks for the argument value of the predicate expressed in PREC.

ID and NID are introduced mainly to clarify the meaning of the knowledge which has disjunction ($\vee$) in its head part. For example, the knowledge $q1 \vee q2$:–p can be represented, if we express all its contrapositives which will be used in the MESON proof procedure, as:

$(q1$:–p,$\neg q2) \wedge (q2$:-p,$\neg q1) \wedge (\neg p$:–$\neg q1, q2)$.

In this case, we cannot prove either q1 or q2 if we assume only p. This is inconvenient, since the intended meaning of $q1 \vee q2$:–p in ordinary cases is that q1 or q2 is true if p is true. This problem will be overcome as follows. The disjunctive formula can be dissolved into the disjunction of mutually contradicting conjunctive formulae. For example, $q1 \vee q2$ can be expressed as:

$(q1 \wedge q2) \vee (q1 \wedge \neg q2) \vee (\neg q1 \wedge q2)$

This exclusive knowledge is expressed using ID and NID. If $q1 \vee q2$:–p is fact knowledge, then it will be expressed in our system as:

```
fact(1,[2,3],[ ],(q1&q2:–p))
fact(2,[1,3],[ ],(q1&n(q2):–p))
fact(3,[1,2],[ ],(n(q1)&q2:–p))
```

where n( ) denotes the logical negation. The conjunction in the head part and the disjunction in the body part are eventually removed by logical transformations.

HYPOTHESIS SELECTION AND ASSOCIATED FUNCTIONS

In order to understand the practical usefulness of the hypothesis-based logical reasoning system, its application to a fault diagnosis problem will now be shown.

The hypothesis selection mechanism implemented in the current system is the following general one. That is, observable data in a problem domain is predefined as

observable([predicate–list])

When more than two sets of hypotheses are formed, the system generates a question to obtain a critical additional observation among the predefined observable data for eliminating a part of the hypothesis sets. Suppose two sets of hypotheses, h1 and h2, remain as a possible answer. The system tries to find an instantiated observation O′ satisfying

$F \cup h1 \vdash O'$ and $F \cup h2 \nvdash O'$ (or vice versa)

To eliminate either h1 or h2 it then asks the user whether or not O′ is true.

The amount of observable data is limited and sometimes not enough. On the other hand, in some cases there are controllable input points to which various signals can be applied to obtain necessary additional observations. Therefore, our hypothetical reasoning system allows the declaration of such controllable input points as

askable([predicate–list])

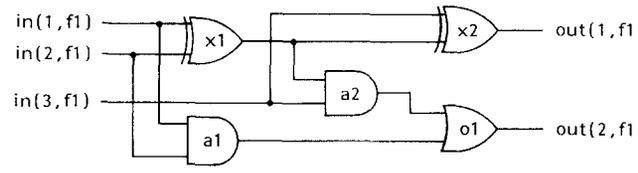
According to this declaration and the related predicate–argument type declaration being written elsewhere, the system generates, if necessary, instantiated input patterns in the hypothesis selection process to obtain useful observations.

An optimal strategy for the hypothesis selection is to choose the question which can discriminate two groups of the hypothesis set with balanced numbers, since it makes it possible to reach a conclusion with the minimum number of questions. The current system, however, has not been equipped with this strategy. In Reference 5, an efficient method of generating questions with which to select either set of hypotheses h1 or h2 is presented for the case when there is an observation which can be derived from $(h1 \cup h2)$ but not from either h1 or h2. This is, however, a very special case.

Figure 2 shows an application of the current system to the fault diagnosis of a digital circuit. Two types of fault are assumed at each gate, namely stuck-on and stuck-off, in which the output is stuck at 1 and 0, respectively. Since the function of each gate is either normal (ok) or fault (stuckon, stuckoff), these functions are expressed as exclusive hypotheses. On the other hand, the connections between gates and terminals are expressed as fact knowledge because they are assumed never to be in fault.

Once a set of malfunctions is given as an observation, the system first forms 14 sets of hypotheses including multiple faults as shown in Figure 2. By obtaining additional observations through the interactive hypothesis-selection process, the system eventually determines the fault to be that the exlusive-OR gate is stuck-off.

An important point here is that the diagnosis system can be built by describing the knowledge systematically, rather than by describing causal relations and/or heuristic knowledge as in existing expert systems. Thus the system components with no fault possibility and with fault possibility are described as fact (complete) knowledge and hypothesis (incomplete) knowledge, respectively. This approach enables expert systems to be built using deep knowledge. Heuristic knowledge may be incorporated as meta-knowledge to achieve efficient inference. This hypothesis-based logical reasoning system can also be applied to design problems if we regard given specifications and possible design components as given observations and hypotheses, respectively. The effective use of



Digital circuit (full adder, name: f1)

```
| ?-hypoth.
Hypothesis formation starts.
Please input knowledge base name
>'ex1.11'.
ex1.11 reconsulted 4552 bytes 0.763003 sec.
Please input symptoms
>((val(out(1,f1),0)&val(out(2,f1),0):-val(in(1,f1),0)&val(in(2,f1),0)&val(in(3,f1),0))
&(val(out(1,f1),0)&val(out(2,f1),0):-val(in(1,f1),0)&val(in(2,f1),1)&val(in(3,f1),0))
&(val(out(1,f1),0)&val(out(2,f1),0):-val(in(1,f1),1)&val(in(2,f1),0)&val(in(3,f1),0))).
Forming hypotheses...
Formed hypotheses are:
⟨stat(a1,ok) & stat(a2,ok) & stat(o1,ok) & stat(x1,stuckoff) & stat(x2,ok)⟩
⟨stat(a1,stuckoff) & stat(a2,ok) & stat(o1,ok) & stat(x1,stuckoff) & stat(x2,ok)⟩
⟨stat(a1,ok) & stat(a2,stuckoff) & stat(o1,ok) & stat(x1,stuckoff) & stat(x2,ok)⟩
⟨stat(a1,stuckoff) & stat(a2,stuckoff) & stat(o1,ok) & stat(x1,stuckoff) & stat(x2,ok)⟩
⟨stat(o1,stuckoff) & stat(x1,stuckoff) & stat(x2,ok)⟩
⟨stat(a1,ok) & stat(x1,ok) & stat(a2,ok) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(a1,stuckoff) & stat(x1,ok) & stat(a2,ok) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(a1,ok) & stat(x1,stuckon) & stat(a2,ok) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(a1,stuckoff) & stat(x1,stuckon) & stat(a2,ok) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(a1,ok) & stat(x1,stuckoff) & stat(a2,ok) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(a1,stuckoff) & stat(x1,stuckoff) & stat(a2,ok) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(a1,ok) & stat(a2,stuckoff) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(a1,stuckoff) & stat(a2,stuckoff) & stat(o1,ok) & stat(x2,stuckoff)⟩
⟨stat(o1,stuckoff) & stat(x2,stuckoff)⟩

Observation:
val(out(1,f1),1):-val(in(3,f1),1)
is right?(y/n)
>n.
Observation:
val(out(2,f1),1):-val(in(2,f1),1)&val(in(1,f1),1)&val(in(3,f1),0)
is right?(y/n)
>y.
Observation:
val(out(2,f1),0):-val(in(2,f1),0)&val(in(1,f1),0)&val(in(3,f1),1)
is right?(y/n)
>y.
Observation:
val(out(2,f1),1):-val(in(2,f1),1)&val(in(1,f1),0)&val(in(3,f1),1)
is right?(y/n)
>y.
Verified hypothesis is:
⟨stat(a1,ok) & stat(x1,ok) & stat(a2,ok) & stat(o1,ok) & stat(x2,stuckoff)⟩

Do you continue?(y/n)
>n.
yes
```

Observation

Hypotheses formation

14 hypotheses formed at first step

Question-answering for hypothesis selection

Figure 2. An application of the hypothesis-based reasoning system to the fault diagnosis of a digital circuit. val(out(1,f1),0) means that the value of the node out (1,f1) is 0, and stat(a1,ok) means that the status of the a1 gate is OK

constraint knowledge becomes crucial, particularly in design problems, in narrowing down the search space.

INDUCTIVE CONCEPT LEARNING MECHANISM

The knowledge acquisition support or learning capability is important for next-generation knowledge systems. Two knowledge acquisition modules have been added to the current hypothesis-based logical reasoning system as shown in Figure 3. The first module, which is based on an inductive inference extended to deal with the knowledge base with hypotheses, enables concept learning from given examples. The second module provides knowledge assimilation and management functions for the frame knowledge base with hypotheses. The redundant knowledge base is attached in Figure 3 to cope with the nonmonotonicity of the hypothetical reasoning system. These knowledge acquisition mechanisms will be seen in the following.

An overview of inductive concept learning methods for complete knowledge is described in Reference 6.

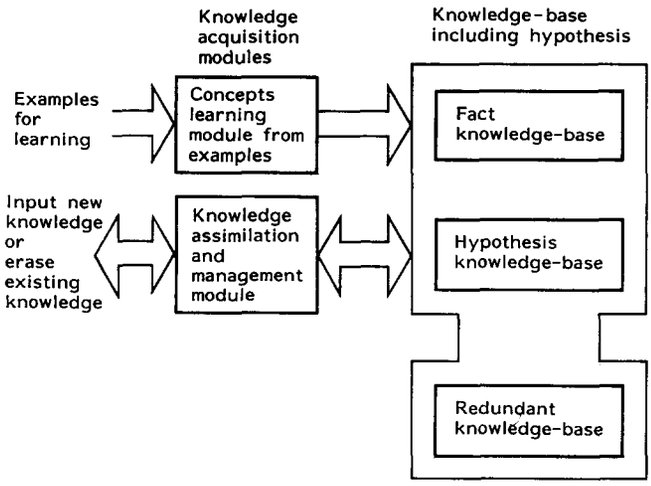


Figure 3. Knowledge acquisition modules for the knowledge base including hypotheses

Considered here will be the knowledge-base case including hypotheses, which allows the acceptance of even contradicting examples.

Let learning examples be given in the form of

obs_a & obs_b & ..... :–concept_1

where the body and head parts are the concept and its associated observations. ('&' in the head part may be replaced by ',' in our system.) In the fault diagnosis system, the cause and symptoms correspond to the concept and observations, respectively, in the above formalism.

We assume here that all the given examples are correct; i.e., no erroneous example is involved. We also assume that all the necessary observations are given for concluding one concept. (For example, the learning examples are collected from the diagnoses of medical doctors, whose decisions always rely on sufficient observations.)

The current system allows a hierarchical frame-structured organization of the concepts as shown in the next section. However, the automatic learning of the hierarchical organization is not supported at present, since it requires prior knowledge about concept hierarchy. Thus inductive learning here is to acquire decision rules which show causal relations between the target concept and observations. Since all the possible observations are to be explainable from fact and hypothesis knowledge in the hypothesis-based reasoning system, the learning module forms as many as generalized knowledge, mostly as hypotheses.

The basic strategy of the inductive learning module is to proceed the generalization of observations regarding one concept while excluding the observations of other concepts in the given examples. The key technology here is an extended version of minimum generalization in a logic framework[7].

The argument of the predicate is chosen as the target item of the generalization. To perform the generalization, the system has to know the type of the argument. The type is indicated in some place, for example, as:

defgentype(p(ord, dis))
defgentype(r(any))

where the first and second arguments of predicate p are defined as ordered number and disjoint entity, respectively, and the argument of predicate r is defined as any entity. The following exemplifies the minimum generalization according to this type declaration:

[Ex] Knowledge formation by the minimum generalization regarding r(s) and r(t) when r(s):–c and r(t):–c are given as learning examples.

- if defgentype(r(ord)) and s<t, then
  fact(ID,[ ],[r(X)],(s=<X<=t:–c.))
- if defgentype(r(dis)) and s≠t, then
  fact(ID,[ ],[r(X)],(X=s;t:–c.))
- if defgentype(r(any)) and s≠t, then
  fact(ID,[ ],[ ],(r(X):–c.))

The extended minimum generalization for the knowledge including hypotheses is as follows, where 1st–3rd arguments of fact ( ) and 1st–4th arguments of hyp( ) in the knowledge representation described earlier are omitted for simplicity.

Extended minimum generalization

When fact($A_1, \ldots, A_l$:–c) and hyp($A_{l+1}, \ldots, A_m$:–c) already exist with respect to the concept c, the procedure of the extended minimum generalization with the new input example $a_1, \ldots, a_m$:–c is as follows:

(where *i, j, k, l, m, n* are 1, 2, 3, . . ., and n( ) denotes logical negation)

- for $j = 1$ to $j = m$
  if a_j is comparable* with $A_k (1 \leq k \leq n)$
  then replace A_k with the minimum generalization of (A_k, a_j)
  else if a_j is comparable with $n(A_k)$
  then remove A_k from the head of fact($A_1, \ldots, A_l$:–c) and
  add A_k and a_j to the head of hyp($A_{l+1}, \ldots, A_n$:–c)
  else add a_j to the head of hyp($A_{l+1}, \ldots, A_n$:–c)
- for $j = 1$ to $j = l$
  if A_j is not comparable with either $a_1, \ldots, a_m$
  then remove A_j from the head of fact($A_1, \ldots, A_l$:–c) and
  add A_j to the head of hyp($A_{l+1}, \ldots, A_n$:–c)

Using this extended minimum generalization, we can construct our inductive learning mechanism. The generalization sometimes results in over-generalization, in which the knowledge induced with respect to one concept becomes able to explain an example belonging to another concept (counter example). In this case, we have to find an adequate splitting of the input examples into more than two groups such that extended minimum generalization does not conflict with counter examples. The knowledge induced from each split group is combined in an exclusive-OR relation. (The smaller the number of split groups the better.) To realize this procedure efficiently, we first

*If A_1 and A_2 are literals with the same predicate symbol and the same logical symbol (negation symbol), we call them comparable.

construct the lattice (all the subsets) of the input examples with respect to one concept. Then we check the subset of the input examples in order, starting with the largest subset, to find out whether or not its extended minimum generalization is over-generalization. If not over-generalization, the induced knowledge from the subset is established. The check of a subset included in the larger subset from which the induced knowledge is already established is then skipped.

Usually in multiple-concept learning for classification or diagnosis problems, the goal is to generate the minimum knowledge necessary for discriminating an observation associated with another concept (counter example). Thus the procedure which keeps only the minimum fact knowledge necessary for discriminating counter examples as fact knowledge, changing other fact knowledge to hypothesis knowledge, is attached at the final phase of the inductive concept learning process. The discriminating fact knowledge pertaining to a particular concept indicates that the observation will always appear along with the concept. Conversely, all the fact knowledge of one concept can never be true in observations from other concepts.

Examples of the above inductive concept learning are shown in Figures 4a and b, where diagnosis knowledge for identifying the cause of food poisoning (*Bacillus botulinus*, *Staphylococcus*, *Vibrio enteritis* or *Salmonella*) is induced from input examples representing their observations.

When compared with the inductive concept learning on version space[9], the mechanism presented here allows the formation of concept description in logical OR relation in addition to AND relation.

```
/* file name exg5 */
ex(pain(stomach),nausea(nauseating),nervous_paralysis(eye),diarrhea,n(fever).
        latent_hour(6),food(boiled_fishpast):-                    botulinus_bacillus).
ex(nauseating(vomiting),nervous_paralysis(throat),diarrhea,n(fever),pain(upper_abdomen).
        latent_hour(2),food(salad):-                              botulinus_bacillus).
ex(nausea(nauseating),nervous_paralysis(eye),diarrhea,n(fever),pain(stomach).
        latent_hour(8),food(box_lunch):-                          botulinus_bacillus).
ex(pain(upper_abdomen),nausea(nauseating),food(cream_puff),diarrhea.
        n(fever),latent_hour(3),nervous_paralysis(none):-           staphylococcus).
ex(pain(stomach),food(rice_ball),diarrhea,n(fever),latent_hour(1).
        nervous_paralysis(none):-                                 staphylococcus).
ex(pain(upper_abdomen),nausea(vomiting),food(pudding),nervous_paralysis(none).
        diarrhea,n(fever),latent_hour(2):-                        staphylococcus).
ex(pain(upper_abdomen),nausea(nauseating),food(salad),diarrhea,n(fever).
        latent_hour(6),nervous_paralysis(none):-                  staphylococcus).
ex(pain(stomach),nausea(nauseating),food(boiled_fishpast),diarrhea,n(fever).
        latent_hour(4),nervous_paralysis(none):-                  staphylococcus).
ex(pain(stomach),nausea(nauseating),food(shrimp),diarrhea,fever.
        latent_hour(12),nervous_paralysis(none):-                 enteritis_vibrio).
ex(pain(upper_abdomen),nausea(vomiting),food(cuttlefish),diarrhea,fever.
        latent_hour(18),nervous_paralysis(none):-                 enteritis_vibrio).
ex(pain(stomach),food(sushi),diarrhea,fever,latent_hour(16).
        nervous_paralysis(none),nausea(none):-                    enteritis_vibrio).
ex(pain(upper_abdomen),food(box_lunch),diarrhea,fever,latent_hour(12).
        nervous_paralysis(none),nausea(none):-                    enteritis_vibrio).
ex(pain(stomach),nausea(nauseating),food(raw_tuna),diarrhea,fever.
        latent_hour(20),nervous_paralysis(none):-                 enteritis_vibrio).
ex(pain(stomach),food(salad),diarrhea,fever,latent_hour(18).
        nervous_paralysis(none),nausea(none):-                      salmonella).
ex(food(box_lunch),diarrhea,fever,latent_hour(12),pain(stomach).
        nervous_paralysis(none),nausea(none):-                      salmonella).
ex(pain(upper_abdomen),food(steamed_fishpast),diarrhea,fever,latent_hour(20).
        nervous_paralysis(none),nausea(none):-                      salmonella).
ex(nausea(nauseating),food(raw_oyster),diarrhea,fever,latent_hour(16).
        nervous_paralysis(none),pain(stomach):-                     salmonella).
ex(food(cooled_tofu),diarrhea,fever,latent_hour(22),nervous_paralysis(none).
        nausea(none),pain(upper_abdomen):-                          salmonella).

defgentype(pain(dis)).
defgentype(latent_hour(ord)).
defgentype(food(dis)).
defgentype(nausea(dis)).
defgentype(diarrhea).
defgentype(fever).
defgentype(nervous_paralysis(dis)).
```

a

Figure 4a. An example of inductive concept learning; input examples for learning

```
| ?-learn.
Learning sessions starts. . .
Please input example database name
>exg5.
Learned rules are as below
fact(1,[],[nervous_paralysis(Y_545)],((Y_545==eye;Y_545==throat):-botulinus_bacillus))
hyp(s#000(Y_489,Y_665,Y_719,Y_774),1,[],[nausea(Y_489)].
((Y_489==nausea;Y_489==vomiting):-botulinus_bacillus))
hyp(s#000(Y_489,Y_665,Y_719,Y_774),1,[],[],(diarrhea:-botulinus_bacillus))
hyp(s#000(Y_489,Y_665,Y_719,Y_774),1,[],[],(n(fever):-botulinus_bacillus))
hyp(s#000(Y_489,Y_665,Y_719,Y_774),1,[],[pain(Y_665)].
((Y_665==stomach;Y_665==upper_abdomen):-botulinus_bacillus))
hyp(s#000(Y_489,Y_665,Y_719,Y_774),1,[],[latent_hour(Y_719)].
(2=<Y_719,Y_719=<8:-botulinus_bacillus))
hyp(s#000(Y_489,Y_665,Y_719,Y_774),1,[],[food(Y_774)].
((Y_774==box_lunch;Y_774==boiled_fishpast;Y_774==salad):-botulinus_bacillus))
fact(2,[],[],(n(fever):-staphylococcus))
fact(2,[],[],(nervous_paralysis(none):-staphylococcus))
hyp(s#001(Y_2098,Y_2154,Y_2276,Y_2740),2,[],[pain(Y_2098)].
((Y_2098==upper_abdomen;Y_2098==stomach):-staphylococcus))
hyp(s#001(Y_2098,Y_2154,Y_2276,Y_2740),2,[],[food(Y_2154)].
((Y_2154==boiled_fishpast;Y_2154==salad;Y_2154==pudding;Y_2154==cream_puff;Y_
2154==rice_ball):-staphylococcus))
hyp(s#001(Y_2098,Y_2154,Y_2276,Y_2740),2,[],[],(diarrhea:-staphylococcus))
hyp(s#001(Y_2098,Y_2154,Y_2276,Y_2740),2,[],[],[latent_hour(Y_2276)].
((1=<Y_2276,Y_2276=<6:-staphylococcus))
hyp(s#001(Y_2098,Y_2154,Y_2276,Y_2740),2,[],[nausea(Y_2740)].
((Y_2740==nauseating;Y_2740==vomiting):-staphylococcus))
fact(3,[4],[],(pain(upper_abdomen):-enteritis_vibrio))
fact(3,[4],[food(Y_5834)],((Y_5834==cuttlefish;Y_5834==box_lunch):-enteritis_vibrio))
hyp(s#002(Y_5956,Y_6061),3[4],[],(diarrhea:-enteritis_vibrio))
hyp(s#002(Y_5956,Y_6061),3,[4],[],(fever:-enteritis_vibrio))
hyp(s#002(Y_5956,Y_6061),3,[4],[latent_hour(Y_5956)].
(12=<Y_5956,Y_5956=<18:-enteritis_vibrio))
hyp(s#002(Y_5956,Y_6061),3,[4],[],(nervous_paralysis(none):-enteritis_vibrio))
hyp(s#002(Y_5956,Y_6061),3,[4],[nausea(Y_6061)].
((Y_6061==vomiting;Y_6061==none):=enteritis_vibrio))
fact(4,[3],[food(Y_4604)].
((Y_4604==raw_tuna;Y_4604==sushi;Y_4604==shrimp;Y_4604==cuttlefish):-enteritis_
vibrio))
hyp(s#003(Y_4496,Y_4550,Y_4720),4,[3],[pain(Y_4496)].
((Y_4496==stomach;Y_4496==upper_abdomen):-enteritis_vibrio))
hyp(s#003(Y_4496,Y_4550,Y_4720),4,[3],[nausea(Y_4550)].
((Y_4550==none;Y_4550==nauseating;Y_4550==vomiting):-enteritis_vibrio))
hyp(s#003(Y_4496,Y_4550,Y_4720),4,[3],[],(diarrhea:-enteritis_vibrio))
hyp(s#003(Y_4496,Y_4550,Y_4720),4,[3],[],(fever:-enteritis_vibrio))
hyp(s#003(Y_4496,Y_4550,Y_4720),4,[3],[],(latent_hour(Y_4720)].
(12=<Y_4720,Y_4720=<20:-enteritis_vibrio))
hyp(s#003(Y_4496,Y_4550,Y_4720),4,[3],[],(nervous_paralysis(none):-enteritis_vibrio))
fact(5,[6],[food(Y_8373)].
((Y_8373==raw_oyster;Y_8373==salad; Y_8373==box_lunch):-salmonella))
fact(5,[6],[],(fever:-salmonella))
fact(5,[6],[],(pain(stomach):-salmonella))
hyp(s#004(Y_8754,Y_8495),5,[6],[nausea(Y_8754)].
((Y_8754==none;Y_8754==nauseating):-salmonella))
hyp(s#004(Y_8754,Y_8495),5,[6],[],)diarrhea:-salmonella))
hyp(s#004(Y_8754,Y_8495),5,[6],[latent_hour(Y_8495)],(12=<Y_8495,Y_
8495=<18:=salmonella))
hyp(s#004(Y_8754,Y_8495),5,[6],(nervous_paralysis(none):-salmonella))
fact(6,[5],[food(Y_7116)],.
((Y_7116==cooled_tofu;Y_7116==raw_oyster;Y_7116==salad;Y_7116==steamed_fishpast):-
salmonella))
fact(6,[5],[],(fever:-salmonella))
hyp(s#005(Y_7232,Y_7446,Y_7062),6,[5],[],(diarrhea:-salmonella))
hyp(s#005(Y_7232,Y_7446,Y_7062),6,[5],[latent_hour(Y_7232)].
(16=<Y_7232,Y_7232=<22:-salmonella))
hyp(s#005(Y_7232,Y_7446,Y_7062),6,[5],[],(nervous_paralysis(none):-salmonella))
hyp(s#005(Y_7232,Y_7446,Y_7062),6,[5],[nausea(Y_7446)].
((Y_7446==none;Y_7446==nauseating):-salmonella))
hyp(s#005(Y_7232,Y_7446,Y_7062),6,[5],[pain(Y_7062)].
((Y_7062==stomach;Y_7062==upper_abdomen):-salmonella))
Do you continue?
>n.
```

b

Figure 4b. An example of inductive concept learning; induced knowledge from examples

KNOWLEDGE ASSIMILATION AND MANAGEMENT FOR FRAME KNOWLEDGE BASE INCLUDING HYPOTHESES

The mechanisms of knowledge assimilation and management have been presented for a logical knowledge base[10]. The logical deductive inference power of Prolog facilitates the detection of redundancy and inconsistency of the knowledge base. This section constitutes a brief description of the mechanism of a knowledge assimilation and management module for the frame knowledge base including hypotheses.

One of the distinctive features of this knowledge base is that it can accept knowledge with exceptions or even contradicting knowledge as hypotheses. However, problems resulting from nonmonotonicity of the inference also have to be dealt with. The redundant knowledge base shown in Figure 3 is a place to keep redundant knowledge temporarily as it may become nonredundant when new knowledge is entered. The knowledge base including only complete knowledge does not need such a store if the manner of knowledge acquisition is incremental. The knowledge in the redundant knowledge base is written in the current system as:

red_db([list-of-[frame-name,knowledge]]).

The hierarchical knowledge representation of the frame and its property inheritance function through the 'isa' link can be easily realized using logic programming. The special operator 'isa' in the system is introduced to denote 'is a relation'. The logical meaning of this 'isa' is logical implication (→). For example, the frame hierarchy of 'tom'→'cock'→'bird' is defined with contrapositives to be used in the MESON proof procedure as:

tom isa cock
cock isa bird
n(cock) isa n(tom)
n(bird) isa n(cock)

Knowledge assimilation

The features of the knowledge assimilation mechanism are as follows:

(a) Even if the new knowledge is inconsistent with existing knowledge, it can be assimilated by changing the existing fact knowledge to hypotheses. Shapiro's logical debugging algorithm[11] is effectively employed to identify the inconsistent existing knowledge.
(b) The system keeps redundant knowledge in a separate place, and tries to re-assimilate it when its redundancy changes due to the nonmonotonicity of the inference.

Suppose that the user is trying to assimilate new knowledge K to a frame Fr. Here the frames and their hierarchy are assumed to be predefined. The knowledge assimilation module executes the following procedure:

1 The set of fact (F) and hypotheses (H) knowledge which can be accessed through the isa link from frame Fr is collected.
2 If F∪h⊢K, where h is a consistent subset of H, K is redundant and is put into the redundant knowledge base red_db( ), and the procedure stopped. Even if K is rule knowledge including body part, it can be checked[10].
3 If F⊢ K with an instantiation of the variables in K, K is inconsistent. Then, by an inconsistent knowledge tracing algorithm (described below), the fact knowledge Fh (⊂F) which causes inconsistency with K is found. Fh is changed to hypothesis. If Fh is knowledge stored directly in Fr, then K is stored as hypothesis knowledge in Fr.
4 The redundancy of K is again checked as in 1 above.
5 If K does not cause inconsistency in the lower frames of Fr, it is assimilated as fact knowledge; otherwise, K is assimilated as hypothesis knowledge.
6 [Recheck of the redundant knowledge base.] The redundancy of the knowledge in the redundant knowledge base may change in this system only when (a) new fact knowledge is assimilated, (b) knowledge is deleted, or (c) the content of NID is modified. The redundancy of the knowledge in the redundant knowledge base is then rechecked. If nonredundant, attempts should be made to assimilate it by the above procedure (1–5).

Knowledge management

The inconsistent knowledge tracing algorithm is designed on the basis of Shapiro's logical debugging algorithm[11], which identifies bug knowledge through a question–answer procedure when an incorrect answer is deduced from the knowledge base. This algorithm is modified to work in the current hypothetical reasoning system based on the MESON proof procedure.

The knowledge management for knowledge deletion is relatively simple. That is, delete the designated knowledge from the frame, and execute the recheck of the redundant knowledge base as described in 6 above.

The management of the frame system including hypothesis knowledge with exceptions is complicated where a semantic network having an isa link with exceptions is studied[12,13]. Here the management mechanism of the isa link with exceptions in the current system is described. This mechanism is invoked in response to the user's request for erasing an inherited knowledge at a frame Fr. The exception of the isa inheritance link is represented in the current system by adding not(Fr) to the body of the related knowledge, where not( ) denotes the negation as failure as used in ordinary Prolog. This is a way for expressing an exceptional case explicitly.

In order to identify the knowledge to which the exception is attached, the modification of Shapiro's debugging algorithm is employed, where in this case hypothesis knowledge as well as fact knowledge becomes the possible target items to identify. As for the use of the hypothesis knowledge, it is necessary to check its consistency with other knowledge. Once the appropriate knowledge is identified, not(Fr) is added to its body. Then the recheck of the redundancy of knowledge stored in the redundant knowledge base is

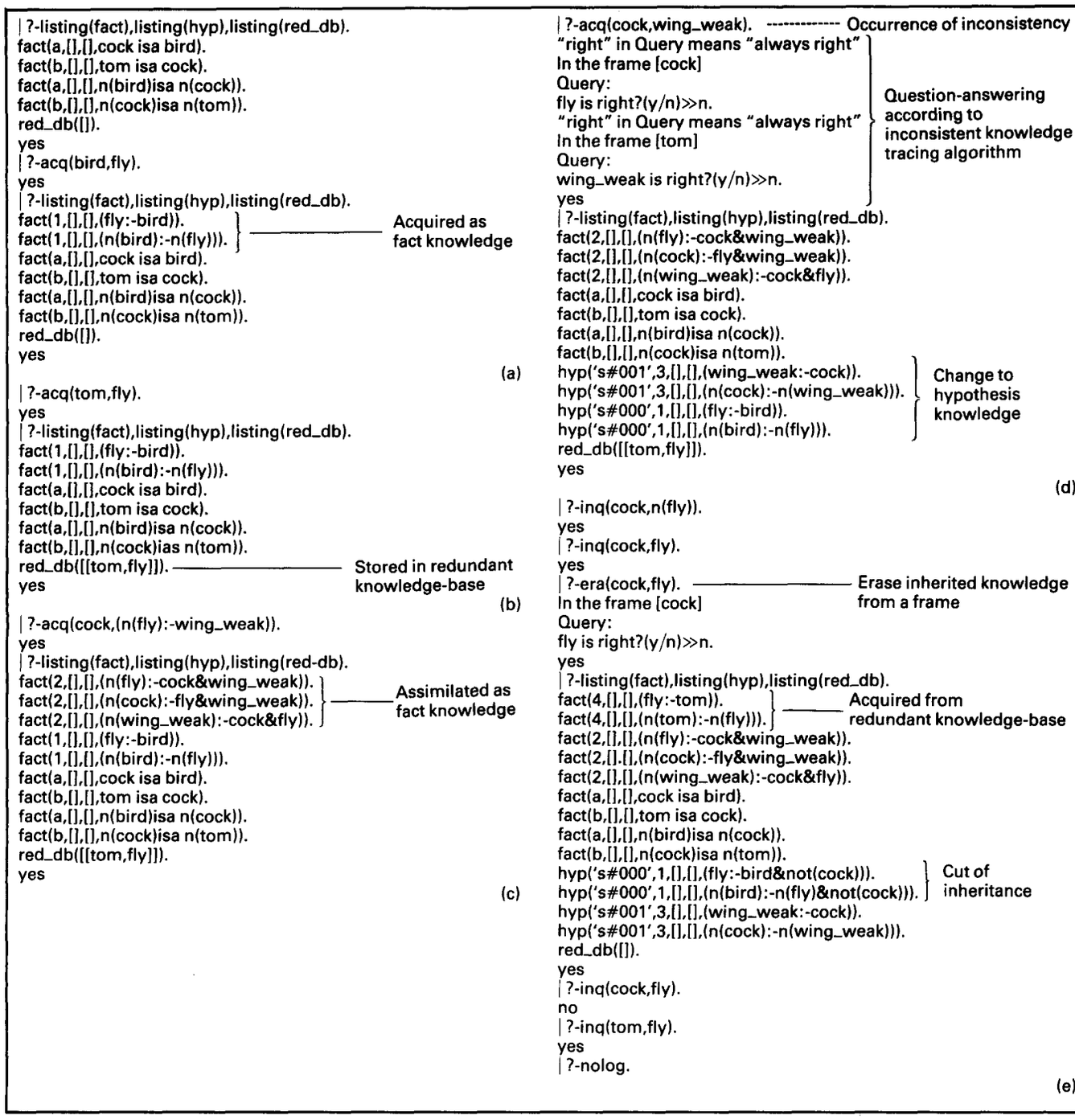


Figure 5. An example of knowledge assimilation and management processes

invoked. In this way, the isa link with exceptions can be adjusted semi-automatically.

The following predicates are implemented for executing the above-mentioned knowledge assimilation and management procedures:

acq(frame-name, new-knowledge)
for assimilating new knowledge into the frame,
del(frame-name, delete-knowledge)
for deleting the knowledge from the frame,
era(frame-name, knowledge-to-deny-its-inheritance)
for adjusting the isa link by adding an exception, and
inq(frame-name, knowledge)
for inquiring whether the knowledge exists in the frame or is inherited from the upper frame.

An example of the behaviour of these mechanisms is shown in Figure 5, where the frame hierarchy of tom→cock→bird is predefined. In Figure 5a, the knowledge 'fly' is given to the 'bird' frame as new knowledge. Since it is neither inconsistent nor redundant, it is assimilated with corresponding contrapositives as fact knowledge. In Figure 5b, the knowledge 'fly' is given to the 'tom' frame; however, it is placed in the redundant knowledge base since it is redundant at this moment. In Figure 5c, the knowledge 'n(fly):-wing_weak' is given to the 'cock' frame; it is assimilated

as fact knowledge. In Figure 5d, the knowledge 'wing_weak' is given to the 'cock' frame. This gives rise to an inconsistency, because 'n(fly)' and the inherited 'fly' co-exist in the 'cock' frame. Therefore the inconsistent knowledge tracing algorithm is invoked. As the result of question answering, the knowledge 'fly' in the 'bird' frame is changed from fact to hypothesis knowledge, and the knowledge 'wing_weak' is assimilated as hypothesis knowledge in the 'cock' frame.

In Figure 5e, the knowledge 'fly' is denied in the 'cock' frame. Then, through question answering generated by the modified debugging algorithm, the expression of exception 'not(cock)' is added to the body of the knowledge 'fly' in the 'bird' frame. In addition, through the recheck of the redundant knowledge base, the knowledge 'fly' is assimilated into the 'tom' frame.

CONCLUSIONS

The mechanisms of two knowledge acquisition modules have been described for the hypothesis-based logical reasoning system, which is very important from the viewpoints both of the theoretical foundation of handling incomplete knowledge and of practical usefulness. Based on this system, the authors are now constructing a next-generation knowledge-based system incrementally to support advanced AI functions such as analogy, commonsense, creative design, learning, etc. From a practical viewpoint, the improvement of inference speed is crucial. To this end we shall have to exploit a mechanism similar to ATMS[14]: while it is necessary for us to treat the knowledge represented in predicate logic with variables, the knowledge in ATMS is restricted to propositional logic with no variable.

ACKNOWLEDGEMENT

The authors wish to thank the members of ICOT's KSS (in 1987) and KSA (in 1988) working group for their discussions on hypothetical reasoning and related issues.

REFERENCES

1 **Pool, D L, Aleliunas, A and Gobel, R** 'Theorist: a logical reasoning system for default and diagnosis' in **Cercone, N J and McCalla, G (Eds)** *The Knowledge Frontier – Essays in the Representation of Knowledge* Springer-Verlag (1987)
2 **Reiter, R** 'A logic for default reasoning' *Artif. Intell.* Vol 13 (1988) pp 81–132
3 **Kunifuji, K, Turumaki, K and Furukawa, K** 'A consideration about a hypothesis-based reasoning system' (in Japanese) *J. Jap. Soc. Artif. Intell.* Vol 2 No 2 (1986) pp 228–237
4 **Loveland, D W** *Automated Theorem Proving: A Logical Basis* North-Holland (1978)
5 **Seki, H and Takeuchi, A** 'An algorithm for finding a query which discriminates competing hypotheses' *ICOT Tech. Rep.* (1985)
6 **Michalski, R S** 'A theory and methodology of inductive learning' in **Michalski, R S *et al.* (Eds)** *Machine Learning* Springer-Verlag (1984)
7 **Plotokin, G D** *A Note on Inductive Generalization: Machine Intelligence 5* John Wiley & Sons (1970)
8 **Matsuda, T and Ishizuka, M** 'An enhanced knowledge representation and concept learning mechanism in a hypothesis-based reasoning system' (in Japanese) *J. Jap. Soc. Artif. Intell.* Vol 3 No 1 (1988) pp 94–102
9 **Mitchell, T M** 'Version space: a candidate elimination approach to rule learning' *5th IJCAI* (1977)
10 **Miyachi, T, Kunifuji, S, Kitami, H and Furukawa, K** 'A knowledge assimilation method for logic databases' *New Generation Comput.* Vol 2 No 4 (1984) pp 385–404
11 **Shapiro, E Y** *Algorithmic Program Debugging* MIT Press (1983)
12 **Etherington, D W and Reiter, R** 'On inheritance hierarchies with exceptions' *AAAI-83* (1983)
13 **Sandwall, E** 'Nonmonotonic inference rules for multiple inheritance with exceptions' *Proc. IEEE* Vol 74 No 10 (1986) pp 1345–1353
14 **DeKleer, J** 'An assumption-based TMS' *Artif. Intell.* Vol 28 (1986) pp 127–162