Efficient hypothetical reasoning system for predicate-logic knowledge base

A Kondo, T Makino and M Ishizuka

A hypothetical reasoning system is an important framework in the development of advanced knowledge-based systems. It can be effectively applied to many practical problems including model-based diagnosis, and designs. However, the inference speed of its PROLOG*-based implementation is slow, and this is particularly because of inefficient backtracking. In order to overcome this problem, a fast hypothetical reasoning mechanism for propositional-logic knowledge has been developed by combining the advantages of forward and backward reasoning styles. This fast mechanism, however, cannot be applied to hypothetical reasoning with predicate-logic knowledge where variables are included as arguments. The paper presents a fast hypothetical reasoning mechanism for predicate-logic knowledge as an extension of the above idea. A reasoning method developed in the deductive database area is effectively utilized to realize this fast mechanism, which can even manipulate recursive rules.*

Keywords: hypothetical reasoning, predicate logic

Knowledge is often incomplete; that is, it often involves exception or contradiction. The handling of incomplete knowledge in the knowledge-base is an important function in expanding the capability of current knowledge bases [1]. Hypothetical reasoning can handle such incomplete knowledge as hypotheses [2,3]. It can be directly applied to model-based diagnosis systems [2,3], design systems [4] etc. Thus the hypothetical reasoning system is an important framework for a next-generation knowledge-based system both from the theoretical and the practical viewpoints. The most crucial problem in hypothetical reasoning is its slow inference speed due to its non-monotonic reasoning nature.

One practical way to overcome this problem is to incorporate heuristic knowledge which serves to navigate inference paths. However, this causes a knowledge acquisition bottleneck, because it is difficult to collect all the necessary heuristic knowledge to cover all the areas of a given problem domain. Therefore, it is necessary to find a fast hypothetical reasoning method working under declarative knowledge.

We have developed two fast hypothetical reasoning systems for a propositional-logic knowledge base. The first one is based on the formation of an inference-path network for a given goal [6]; the second one adopts a parallel inference on a hypothetical-lattice structure [8]. In order to improve efficiency, both of them avoid inefficient backtracking caused by inconsistency among hypotheses. Parallel inference methods similar to ATMSs [5] are employed for this purpose.

In general, variables play an important role in expanding the scope of the knowledge representation capability. If we represent knowledge in propositional logic with no variables, the scale of the knowledge base becomes too large for many practical cases. Using the variables in predicate logic representation, we can express necessary knowledge in a compact form. Thus it is required to develop a fast hypothetical reasoning system capable of working for predicate-logic knowledge with variables.

The methods of the above fast hypothetical reasoning systems developed for propositional logic, however, cannot be applied in a straightforward manner to the predicate-logic case. In this paper we present a fast hypothetical reasoning mechanism that is effective for predicate-logic knowledge (actually for function-free predicate Horn-clause knowledge). A reasoning method developed in the deductive database area is effectively applied to this mechanism.

LOGIC-BASED HYPOTHETICAL REASONING SYSTEM

Our hypothetical reasoning is based on a logic framework first presented in Theorist [2], where knowledge is divided into fact (complete knowledge) and hypothesis (incomplete knowledge). It can deal with incomplete

Institute of Industrial Science, University of Tokyo, 7–22–1 Roppongi, Minato-ku, Tokyo 106, Japan
Paper received 2 September 1991. Accepted 21 February 1992

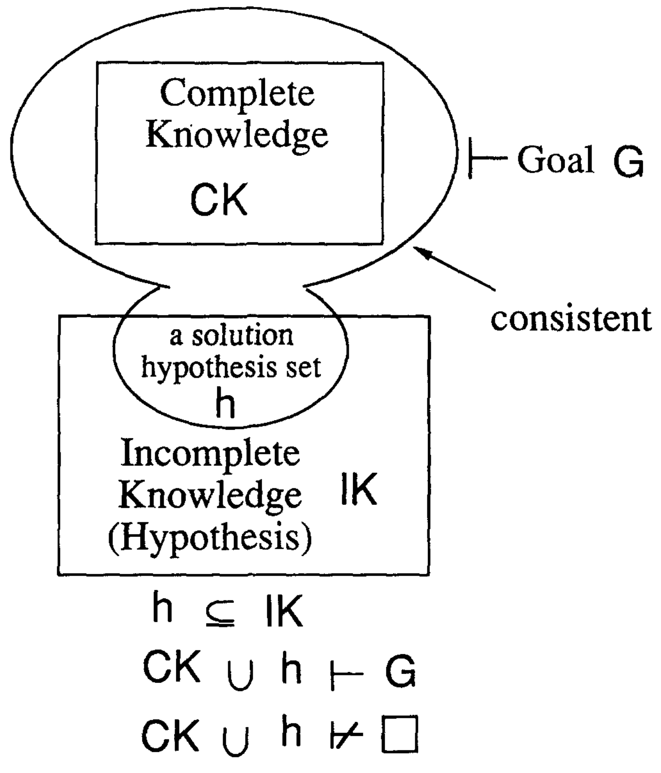


Figure 1. Logic-based hypothetical reasoning system

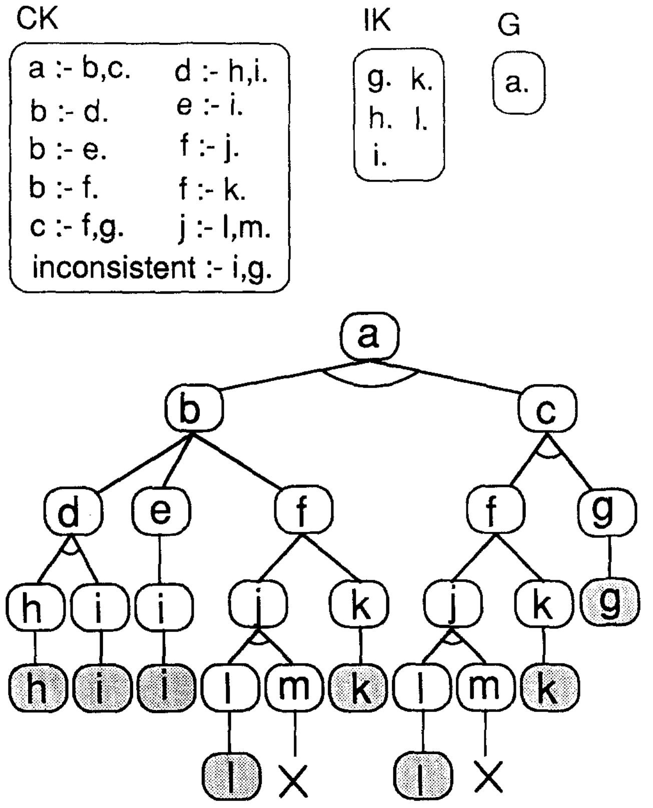


Figure 2. Example of inefficiency of PROLOG-based hypothetical reasoning

knowledge as hypothesis, which is defeasible knowledge having the possibility of contradiction with other knowledge. The basic behaviour of this hypothetical reasoning is as follows: if a given goal cannot be proved with only complete knowledge, the system adopts a consistent set of the hypotheses for proving the goal.

This selected hypothesis set, which we call 'a solution hypothesis set', becomes an answer, that is, it may be a fault in a diagnosis problem or a combination of possible design components in a design problem. While the deductive inference mechanism is used to prove the given goal, it can be said that a reverse deductive inference mechanism is utilized to search the solution hypothesis set. Because of this generating function of the consistent solution hypotheses set, the system has the practical importance of being applicable to many problems, such as diagnoses [2,3] and design [4]. Furthermore, it becomes a framework of abduction.

Figure 1 shows the basic structure of the hypothetical reasoning. The knowledge base consists of two parts. One is a set of complete knowledge *CK* (which is always true in the world); the other is a set of hypotheses or incomplete knowledge *IK* (which is not always true in the world and sometimes contradicts other knowledge). Let *G* be a given goal, and *h* be a subset of *IK*. Then the basic function of the system can be written as that of finding a solution hypothesis set *h* satisfying the following three logical equations:

$$h \subseteq IK$$

(h is a subset of IK),

$$CK \cup h \vdash G$$

(G can be proved from CK and h),

$$CK \cup h \nvdash \square$$

(CK and h are consistent). It is desirable in general that the solution hypothesis set h is a minimal one; that is, there is no solution hypothesis set h' such that $h' \subset h$ and h' satisfies the above three logical equations.

INEFFICIENCY OF SIMPLE IMPLEMENTATION UTILIZING PROLOG INFERENCE MECHANISM

A hypothetical reasoning system can be easily implemented utilizing the inference mechanism of PROLOG. In this case, a necessary set of hypotheses is generated along the depth-first inference path. This generated hypothesis set is then subjected to a consistency check. If a contradiction is found, a part of the generated hypothesis set is discarded and another hypothesis is generated in accordance with the backtracking mechanism of PROLOG. When the inference succeeds, the generated consistent hypothesis set becomes a solution hypothesis set.

However, this simple implementation using the PROLOG inference mechanism is not efficient, as described below. Figure 2 exemplifies this inefficient inference behavior.

- *Search for non-promising branches:* This is the case where a branch has a false node on the inference path. Since the node of this branch has no possibility of being true, it is useless to search this branch. For example, in Figure 2, the node *j* is always false because its child node *m* is false. Thus a search of the node *j* branch will be in vain.
- *Plural searches for the same branch:* This is a general problem with PROLOG's backtracking. After changing a hypothesis upon backtracking, no information

remains about the backtracked branch. If the same branch appears again, then it may be searched again. For example, in Figure 2, at node *b*, the child node *d* is first selected. At the search stage before node *g*, the candidate for the solution hypotheses set is [*h, i, k*]. Adding the hypothesis *g*, backtracking is invoked owing to the inconsistency between *i* and *g*. At this time the information about the node *f* branch is lost. Therefore, after searching the next child node *e* of the node *b*, the node *f* branch is searched again. In hypothetical reasoning, backtracking occurs more often than in the usual inference cases because of the possibility of inconsistent hypotheses. Hence this inefficiency is very serious in hypothetical reasoning.

- *Plural searches for the same sub-tree:* There may be more than two identical sub-trees on the inference tree. Since these sub-trees cannot be identified, they are searched respectively. For example, in Figure 2, there are two nodes *f* on the inference tree. In the inference process, these nodes *f* are searched. This is also a general problem of PROLOG-based backward inference.
- *Search for redundant solutions:* An obtained solution may not be a minimal one, i.e. it may be a redundant solution. Since the PROLOG inference mechanism retains just one solution at a time, it is impossible to know whether or not the solution is minimal. For example, in Figure 2, the solution hypothesis set at the node *d* is [*h, i*] and that of the node *e* is [*i*]. As [*h, i*] is redundant to [*i*], the search for the node *d* should be avoided.

We describe in the following section the fast hypothetical reasoning systems that we have developed for propositional-logic knowledge in order to solve these problems.

FAST HYPOTHETICAL REASONING FOR PROPOSITIONAL-LOGIC KNOWLEDGE

There are two main inference methods, i.e. backward (top-down) inference and forward (bottom-up) inference. The backward inference, as in PROLOG, has the advantage of searching only goal-related nodes, and the disadvantage of searching the same node more than twice. Especially in hypothetical reasoning, this disadvantage is very serious, because backtracking is invoked frequently owing to the inconsistency between hypotheses, as described in the previous section.

On the other hand, the forward inference, which is suitable for searching all the solutions, has the advantage of not searching the same node twice, and the disadvantage of searching the nodes that are not related to the given goal. In the ATMS [5], which is usually used in combination with a forward production system, an efficient parallel forward inference is realized by maintaining multiple consistent hypothesis sets (environments). The control leading to a goal-directed inference path depends on heuristic rules written by a user in the ATMS. In our logic-based hypothetical reasoning, we cannot rely on this type of heuristic knowledge.

The problems described in the previous section can be solved by combining the advantages of backward and forward reasoning.

The first problem is due to the depth-first search mechanism of PROLOG. Even if a false node exists in the right space of an AND branch, it cannot be recognized before the search. Since synthetizing hypotheses is very expensive in hypothetical reasoning, it is important to prune non-promising branches (branches involving false nodes) before the hypothesis synthesis. This problem can be solved by forming a compiled inference path (backward inference process) before synthetizing the hypotheses (forward inference process).

The second problem is due to the fact that the PROLOG-based version holds a single environment. This problem can be solved by using a parallel forward inference with a multiple environment to avoid the backtracking caused by inconsistency between hypotheses.

The third problem can be solved by merging the identical nodes into one to form a compiled inference-path network.

The fourth problem can be solved by holding multiple environments at each node, and deleting redundant (non-minimal) hypothesis sets.

Considering these points, we have so far developed the following two fast hypothetical reasoning systems for propositional-logic knowledge.

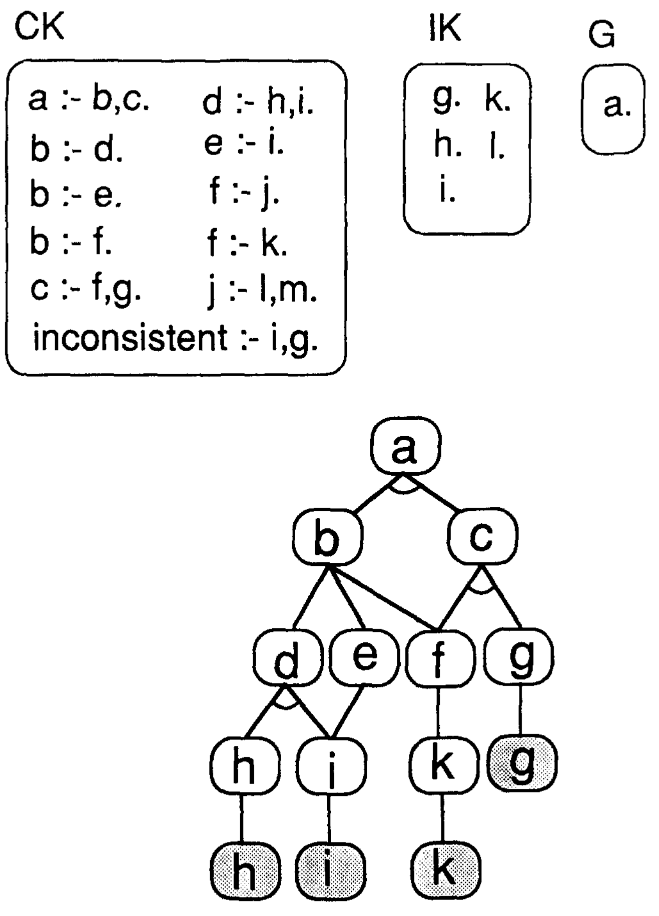
Fast hypothetical reasoning using inference-path network

In this method [6], a goal-directed initial inference-path network is first formed by connecting related knowledge. Identical nodes are merged into one. By propagating truth or false values from leaf complete knowledge nodes, inference paths known to be always-true or always-false regardless of the hypotheses are deleted. As a result, an inference-path network can be formed for the given goal. We call this process the 'inference-path formation phase'. The first and third problems can be solved in this phase. This inference-path formation phase is very efficient since it is based on a linear-time algorithm for testing the satisfiability of the propositional Horn formulae by Dowling and Gallier [7], Next, hypotheses are synthetized in a forward inference manner along this inference-path network. We call this process the 'hypothesis synthesis phase'. Since the hypotheses are synthetized with holding multiple combinations (environments), as in the ATMS [5], the second and fourth problems are solved. Figure 3 shows a formed inference-path network for the same knowledge base as is described in Figure 2.

Parallel inference utilizing hypotheses-lattice structure

In this method [8], a given goal is first unfolded using only complete knowledge into a number of sub-goals. Since always-true and always-false nodes, for which truth values are determined regardless of the hypotheses, disappear during this process, the first problem is solved. Next, supporting hypotheses for each sub-goal are mapped onto a hypotheses lattice. Since minimal hypothesis sets can be easily found on the lattice, the fourth problem is solved. Finally, these hypotheses on the lattices are synthetized into solution hypothesis sets. Since there is no backtracking, the second problem is solved. This method does not compile knowledge into a network structure; the third problem is not solved.

These two hypothetical reasoning systems greatly improved the inference speed for propositional-logic



```
CK
a :- b,c.      d :- h,i.
b :- d.        e :- i.
b :- e.        f :- j.
b :- f.        f :- k.
c :- f,g.      j :- l,m.
inconsistent :- i,g.

IK
g. k.
h. l.
i.

G
a.
```

Figure 3. Example of efficient hypothetical reasoning utilizing inference-path network

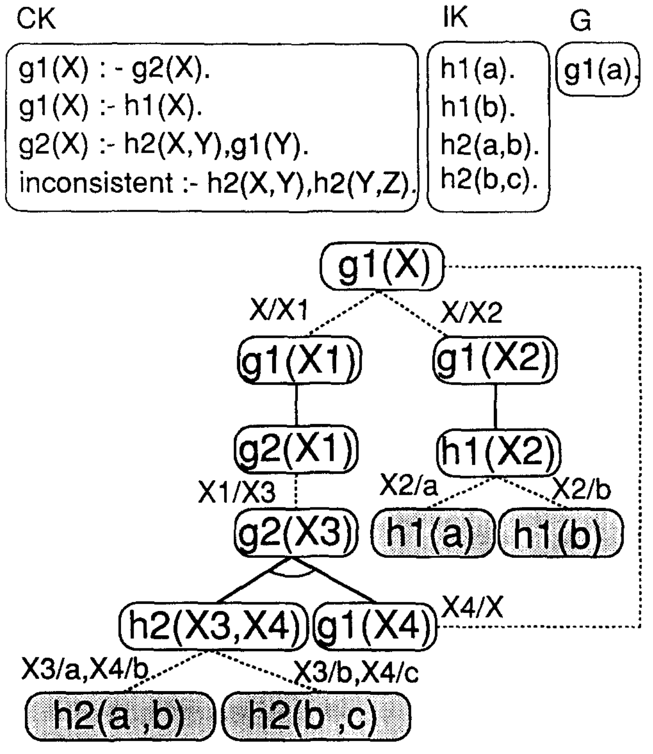
knowledge. In particular, the first system achieved an inference speed that was thousands of times faster than that of the system implemented utilizing the inference mechanism of PROLOG.

EXTENSION TO PREDICATE-LOGIC KNOWLEDGE

In this section, we consider the application of these fast hypothetical reasoning mechanisms to the predicate-logic case. They cannot, however, be applied easily owing to the unification among the nodes.

First, we consider an extension of the first method, using the inference-path network [6]. Figure 4 shows an example of the initial inference-path network for the predicate-logic knowledge base. In Figure 4, different variable names are assigned to the same node since the variable of each rule is independent. Rule recursion is included in this example.

If a goal is $g1(a)$, the instantiation X to a is propagated along this initial network; then a contradiction occurs because of different instantiations X to a and b (X/a and X/b through $X/X1/X3/a$, $X4/X1/b$). This is because $g1(a)$ and $g1(b)$ are expressed as the same node $g1(X)$, although these nodes should be different nodes. In general, the truth value of the node cannot be determined before the node is instantiated. Therefore, in order to form the inference-path network for predicate-logic knowledge, knowledge has to be expanded in the Herbrand universe. However, the size of network in this case will become too large for practical use.



```
CK
g1(X) : - g2(X).
g1(X) :- h1(X).
g2(X) :- h2(X,Y),g1(Y).
inconsistent :- h2(X,Y),h2(Y,Z).

IK
h1(a).
h1(b).
h2(a,b).
h2(b,c).

G
g1(a)
```

Figure 4. Example of initial network for predicate-logic knowledge base

Thus we consider an extension of the second method, i.e. parallel inference utilizing a hypothetical-lattice structure [8]. However, the unfolding method in the propositional-logic case cannot be applied to the predicate-logic case for the following reasons

Cases where unfolding is inefficient without hypotheses

Since the variables of the nodes may remain as they are unless hypotheses are adopted, unnecessary branches not related to the goal may be searched. Figure 5, for example, shows this type of inefficiency with respect to the application of the unfolding method. Since the node $g3(X1, X)$ would be instantiated to $g3(a, b)$ with the adoption of the hypotheses $h1(a)$ and $h2(b)$, the system needs to prove only the node $g3(a, b)$ for this node $g3(X1, X)$ in this case. However, as the variables X and $X1$ of the node $g3(X1, X)$ cannot be determined without adopting hypotheses, the system goes on to prove the node $g3(X1, X)$. This search is inefficient. In this example, the unfolding method without the adoption of hypotheses searches hypotheses $h3(a)$, $h3(b)$, $h4(a)$ and $h4(b)$, although only $h4(a)$ needs to be searched.

Cases where unfolding is impossible without hypotheses

Without determining the variables of the nodes, the unfolding may become impossible, especially when some rules are recursive. A recursive rule should be excluded in propositional logic because the inference does not terminate. However, in the predicate-logic case, the inference terminates even with recursive rules after the instantiation of the variables. Thus it is natural to permit recursive rules in a predicate-logic knowledge base. Figure 6

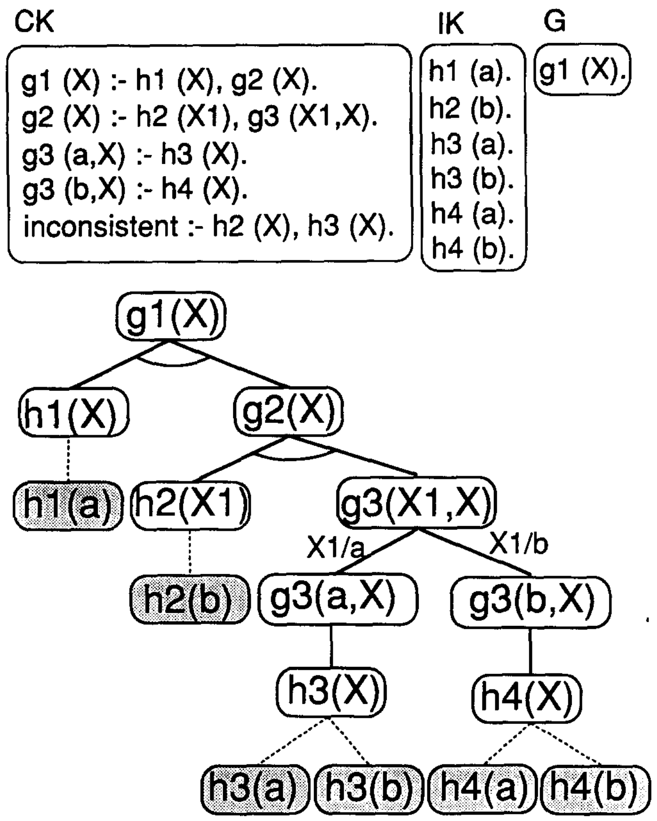


Figure 5. Example in which unfolding is inefficient without adoption of hypotheses

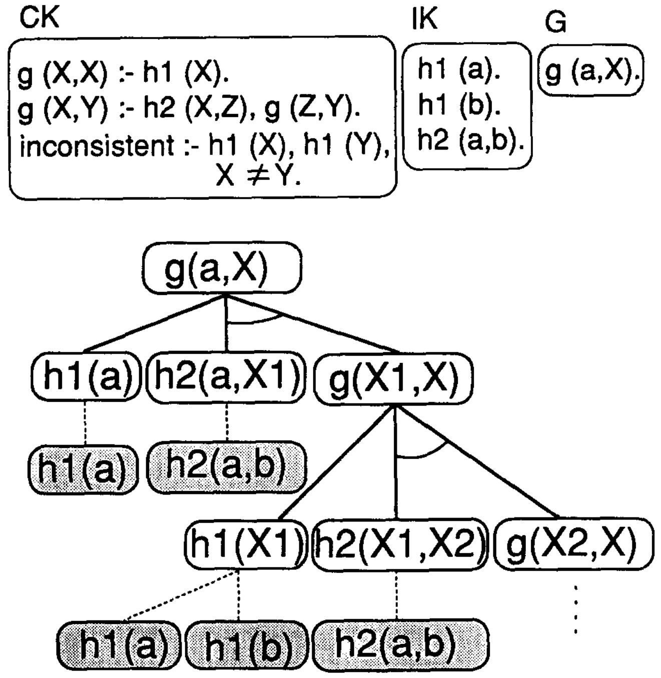


Figure 6. Example in which unfolding is impossible without adoption of hypotheses

shows an example where the unfolding is impossible unless hypotheses are adopted. The goal could be proved even in this example if all the variables of the nodes were determined by adopting hypotheses.

Consequently, it becomes necessary in the case of predicate logic to determine the variables of the node by adopting hypotheses, and to propagate this unification information among the nodes under search. Considering the above issues, we have developed a fast hypothetical reasoning system called KICK-HOPE (Knowledge-Base Handling Incomplete Knowledge — by Holding Parallel Solution on Environment Lattice) which is applicable to function-free predicate-logic Horn-clause knowledge. The inference mechanism of KICK-HOPE corresponds to that of the QSQR method [9, 10] in deductive database technology, but KICK-HOPE can also manipulate hypotheses (defeasible knowledge).

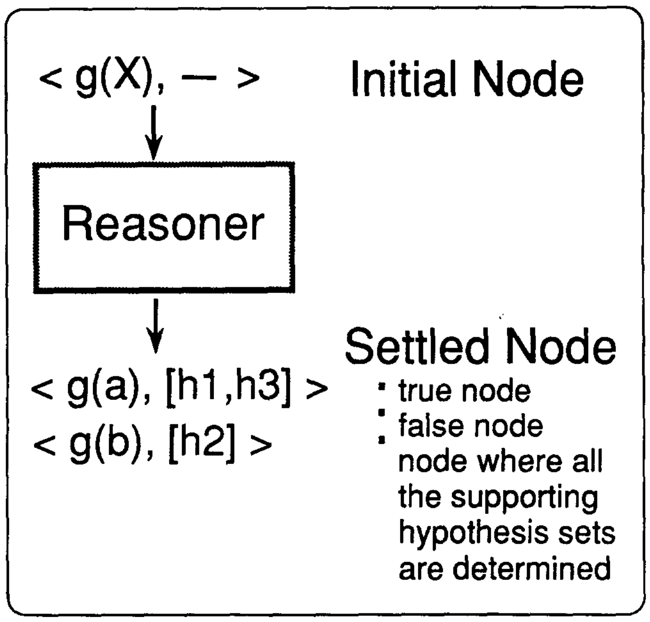


Figure 7. Function of KICK-HOPE reasoner

KICK-HOPE: A FAST HYPOTHETICAL REASONING SYSTEM APPLICABLE TO PREDICATE-LOGIC KNOWLEDGE

While rule-type incomplete knowledge is allowed in our knowledge base, we transform it by pre-processing into newly introduced unit-clause incomplete knowledge and a modified complete knowledge version of this rule-type knowledge. (The rule-type knowledge corresponds to IDB in deductive databases.) For example, incomplete knowlege '*a*:-*b*.' is transformed before reasoning into complete knowledge '*a*:-*b*,*c*.' and incomplete knowledge '*c*.'. Then all incomplete knowledge becomes unit clauses (fact-type), which are placed at the leaf position of the inference tree.

The data structure of a node in KICK-HOPE is

⟨Node-Name, Supporting Hypothesis Sets⟩

At the initial stage, Node-Name may have variables and Supporting Hypothesis Sets is undecided. These inital nodes are transformed into settled nodes through the reasoner of KICK-HOPE. (See as an example Figure 7). We call this transformation process 'solving the node' to obtain all the settled nodes for a certain node as in this example. The settled nodes are classified into the following three categories (each Node-Name is instantiated):

- True node: always true (no need for hypotheses):

  ⟨Node-Name, true⟩

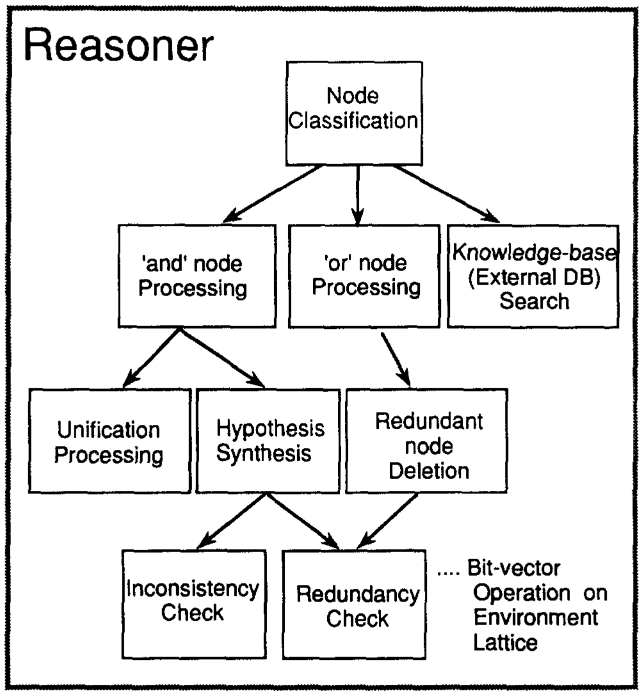


Figure 8. Behavior of KICK-HOPE reasoner

- *False node:* always false:

  ⟨Node-Name, false⟩

- *Node:* where all the supporting hypothesis sets are determined:

  ⟨Node-Name, supporting hypotheses sets⟩

Figure 8 shows the behavior of the KICK-HOPE reasoner. First, a node is judged as any of 'and-node', 'or-node' or others. If the node is 'and-node', 'and-node processing' is executed. If the node is 'or-node', 'or-node processing' is executed. Otherwise, 'knowledge-base (external DB) search processing' is executed. Algorithms for 'and-node processing' and 'or-node processing' are as follows:

```
《Algorithm for node (A and B)》
(1) Solve the node A. (Settled nodes for the node A
    are obtained.)
(2) Unify all the settled nodes for the node A with
    the node B.
(3) Solve all the unified nodes B.
(4) Synthetize supporting hypotheses sets among
    the mutually unified nodes A and B. Delete
    inconsistent or redundant hypothesis sets.
```

```
《Algorithm for node (A or B)》
(1) Solve the node A and the node B. (Settled nodes
    for both node A and node B are obtained.)
(2) Delete redundant hypothesis sets.
```

'Knowledge-base search processing' is executed when the node is not either 'and-node' or 'or-node', that is, the node is a unit clause. This processing is as follows.

```
《Algorithm for unit clause A》
(1) Obtain a list of return nodes for all knowledge
    unified with the node A in the knowledge base.
```

The return node is classified into the following four cases, according to the knowledge to be unified:

- *Case 1:* In this case, the node is unified with rule-type complete knowledge. Node-Name is the body of the unified complete knowledge, and Supporting Hypotheses Sets remains undecided. Since this return node is not yet a settled node, this node is solved afterwards.
- *Case 2:* In this case, the node is unified with fact-type complete knowledge. The return node becomes a true node (a settled node).
- *Case 3:* In this case, the node is unified with incomplete knowledge. Node-Name is this unified incomplete knowledge (hypothesis), and Supporting Hypothesis Set is a list of this hypothesis. This return node is a settled node.
- *Case 4:* In this case, there is no knowledge to be unified. The return node is a false node (a settled node).

Reasoning systems holding parallel solutions, like KICK-HOPE, take a lot of time for merge operations such as the synthetizing of hypotheses and the deleting of redundant hypotheses. In KICK-HOPE, where the hypotheses are expressed as bit vectors as in the ATMS, this merge operation is executed efficiently by bit operations on the hypothesis lattice as in the propositional-logic case, because this operation is done after the nodes have been settled.

ESTIMATION OF INFERENCE SPEED

Figures 9 and 10 show Example 1 and Example 2, respectively, of predicate knowledge bases and corresponding inference-tree structures. Using these examples, we estimate the inference speed of KICK-HOPE compared with that of the implementation utilizing the inference mechanism of PROLOG.

While the CK of both the examples is the same, the IK of Example 2 is an incomplete knowledge set excluding such knowledge as shape $h(\_,1)$ from the IK of Example 1. When we represent the scale of knowledge of the incomplete knowledge $h(X,Y)$ $[X = 1,2,\ldots, N-1, Y = 1,2,3]$ as N, then the number of nodes on the inference tree is $6N + 4$ for both examples. Example 1 is an example in which the same branches are searched multiply because of the backtracking invoked by the inconsistent condition (inconsistent :-$h(X,1)$ & $h(X,3)$.) in the inference of PROLOG. On the other hand, Example 2 is an example in which no plural search happens in both KICK-HOPE and the PROLOG-based inference.

Figure 11 depicts an inference-time result for Example 1. The inefficiency of the PROLOG-based inference is apparent in Figure 11. For example, the inference time at $N = 3$ is 0.12 s in KICK-HOPE and 0.13 s in the PROLOG-based inference mechanism, whereas at $N = 15$ it is 2.22 s in KICK-HOPE and 2693.86 s in the PROLOG-

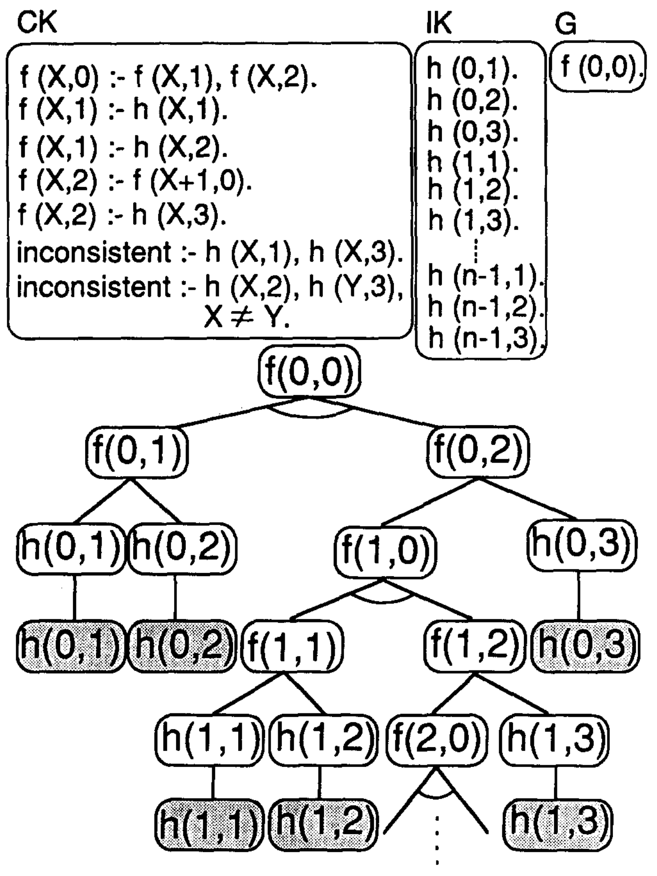


Figure 9. Knowledge base and corresponding inference-tree structure (Example 1)

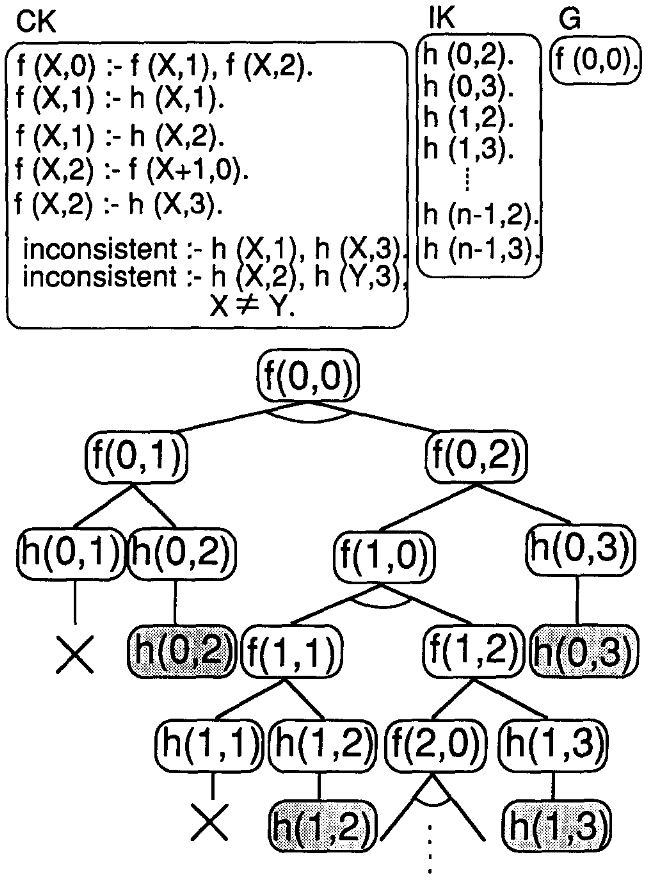


Figure 10. Knowledge base and corresponding inference-tree structure (Example 2)

based inference mechanism. (These data are measured on a Sun-4.)

Figure 12 depicts an inference-time result for Example 2. This shows that the PROLOG-based inference mechanism infers slightly faster than KICK-HOPE. This phenomenon is due to the high processing cost of the merge operation in KICK-HOPE whereas no inefficiency of plural searches in the PROLOG-based inference appears in this case. However, the slope of the inference-time increase of KICK-HOPE is not steep, and its inference speed does not exceed a constant times (only 2–3 times) the range of the PROLOG-based inference mechanism.

These two graphs reveal that the inference speed of the PROLOG-based inference is extremely affected by backtracking, but the KICK-HOPE speed is not; that is, it depends only on the number of nodes on the inference tree. In practical knowledge bases, the number of backtrackings is expected to lie between those in Example 1 and Example 2. Thus KICK-HOPE is superior to a large extent in inference speed over the hypothetical reasoning system using the PROLOG-based inference mechanism.

CONCLUSIONS

We have described the fast hypothetical reasoning system called KICK-HOPE for function-free predicate-logic Horn-clause knowledge. KICK-HOPE has solved the second and fourth problems described in the third section.

The second problem regarding wasteful multiple searches for the same branch is a crucial one, especially in hypothetical reasoning, because of the backtracking

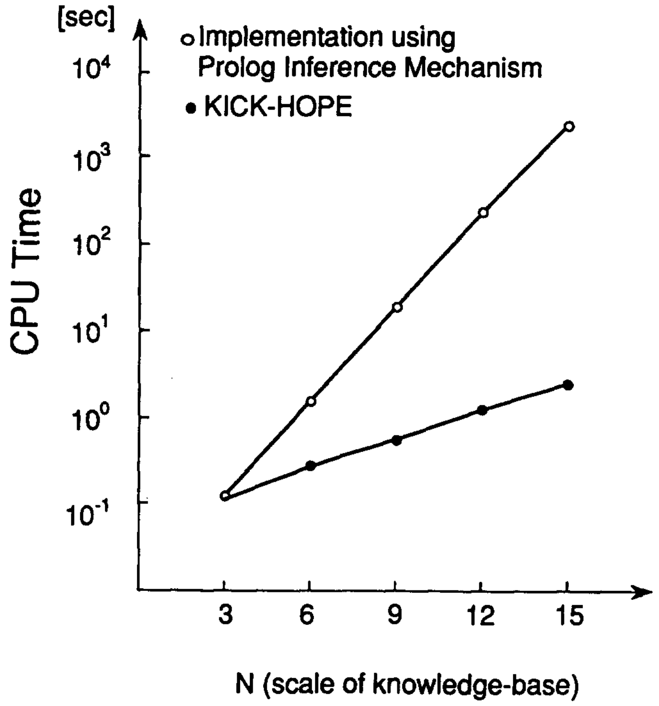


Figure 11. Inference time for knowledge base and goal of Example 1

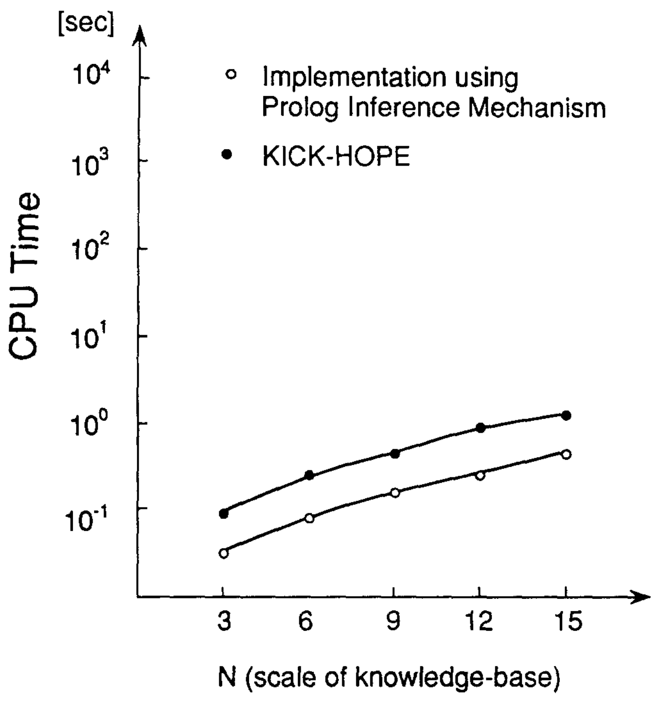


Figure 12. Inference time for knowledge base and goal of Example 2

invoked by inconsistency between hypotheses. By solving this problem, the inference speed has been largely improved (see the previous section).

For the fourth problem, the deletion of redundant hypothesis sets at each node is very important for predicate-logic knowledge. In KICK-HOPE, at each node, after settling the left-positioned child node, the right-positioned child nodes are unified. Deleting redundant hypothesis sets at the left-positioned child node decreases the number of the right-positioned child nodes to be unified; this contributes to narrowing down the right search space.

Thus the feature of KICK-HOPE is that it searches just once for only the necessary inference path related to proving the goal. In general, the number of nodes on the inference trees, however, increases exponentially with respect to the scale of knowledge. Accordingly, the inference speed of KICK-HOPE increases exponentially, as seen in Figures 11 and 12. To make KICK-HOPE faster, we should investigate techniques for lowering the cost of the merge operation, such as synthetizing hypotheses and deleting redundant hypothesis sets. There are some efficient techniques for join operations with bit vectors in relational database technology [11, 12]. Still, since the computational complexity of non-monotonic reasoning including hypothetical reasoning has been proved to be NP-complete or NP-hard [13], even in the propositional-logic case, the inference speed in the worst case cannot exceed the limit of the exponential order if we stay in ordinal search mechanisms. To overcome this limit, the transformation of knowledge (learning) [14] and the utilization of past reasoning results (analogy) [15] are promising approaches. We are now exploring these approaches for further efficient hypothetical reasoning systems.

ACKNOWLEDGEMENTS

The authors are grateful to Professor Randy Goebel (University of Alberta, Canada) for his comments. This work was supported by Ministry of Education Grant-in-Aids 02452154 (B) and 02215105 (Special Area on Intelligent Information & Communications).

REFERENCES

[1] M. Ishizuka: An Approach Toward Next-Generation Knowledge-Base System by Handling Incomplete Knowledge, Journal of JSAI, Vol. 3, No. 5, pp. 552–562 (1988) (in Japanese)

[2] D. Poole, R. Aleliunas and R. Goebel: Theorist; A logical Reasoning System for Defaults and Diagnosis *in The Knowledge Frontier: Essays in the Knowledge Representation* (N. J. Cercone and G. McCalla (Eds.)), Springer-Verlag, USA (1987)

[3] M. Ishizuka and T. Matsuda: Knowledge Acquisition Mechanisms for a Logical Knowledge Base including Hypothesis, Knowledge-Based Systems, Vol, 3, No. 2, pp.77–86 (1990)

[4] T. Makino and M. Ishizuka: A Hypothetical Reasoning System with Constraint Handling Mechanism and its Application to Circuit-Block Synthesis, Proc. PRICAI'90, Nagoya, pp.122–127 (1990)

[5] J. de Kleer: An Assumption-Based TMS, Artificial Intelligence, Vol. 28, pp.127–162 (1986)

[6] F. Ito and M. Ishizuka: Fast Hypothetical Reasoning System using Inference-Path Network, Journal of JSAI, Vol. 6, No. 4, pp.501–509 (1991)

[7] W. F. Dowling, J. H. Gallier: Linear-time Algorithm for Testing the Satisfiability of Propositional Horn Formulae, Journal of Logic Programming, Vol. 3, pp.267–284 (1984)

[8] A. Kondo, T. Makino and M. Ishizuka: An Efficient Inference for Hypothetical Reasoning System by using Parallel Solving Technique with Environment Lattice, 40th Nat. Conv. IPSJ, 6c–1 (1990.3) (in Japanese)

[9] L. Vieille: Recursive Axioms in Deductive Database: The Query/Subquery Approach, Proc. First International Conference on Expert Database Systems, pp.179–193 (1986)

[10] S. Nishio and Y. Kusumi: Evaluation Methods for Recursive Queries in Deductive Databases, Journal of IPSJ, Vol. 29, No. 3, pp.240–255 (1988) (in Japanese)

[11] Y. Stanley and W. Su: *Database Computers* McGraw–Hill (1988)

[12] E. Ozkarahan: *Database Machines and Database Management* Prentice–Hall (1986)

[13] H. A. Kautz and B. Selman: Hard Problems for Simple Default Logics, Proc. of KR'89, Toronto, Canada, pp.189–197 (1989)

[14] T. Makino and M. Ishizuka: A Hypothetical Reasoning System with Experiences-Based Learning Mechanism, 1990 Autumn Nat. Conv. IEICE, No. D-155 (1990.10) (in Japanese)

[15] A. Abe and M. Ishizuka: Fast Hypothetical Reasoning System using Analogy on Inference-Path Network, IPSJ Technical Report 90-AI-72-2 (1990.9) (in Japanese)