An Efficient Hypothetical Reasoning System for Predicate-logic Knowledge-base

Akiko KONDO, Toshiro MAKINO and Mitsuru ISHIZUKA

Institute of Industrial Science, University of Tokyo
Minato-ku, Tokyo 106, Japan

Abstract
A hypothetical reasoning system is an important framework toward advanced knowledge-base systems. It can be effectively applied to many practical problems including model-based diagnosis, designs, etc. However, the inference speed of its Prolog-based implementation is slow particularly due to 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 applicable to hypothetical reasoning with predicate-logic knowledge where variables are included as arguments. In this paper, we present a fast hypothetical reasoning mechanism for the predicate-logic knowledge as an extension of above idea. A reasoning method developed in deductive database area is effectively utilized to realize this fast mechanism, which can manipulate even recursive rules.

1. Introduction
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 to expand the capability of current knowledge-base[1]. A hypothetical reasoning can handle such incomplete knowledge as hypothesis[2,3]. It can be directly applicable to model-based diagnosis systems[2,3], design systems[4], etc. Thus the hypothetical reasoning system is an important framework as a next-generation knowledge-based system both from theoretical and practical viewpoints. The most crucial problem of the 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-path. But this causes knowledge acquisition bottleneck, because it is difficult to collect all the necessary heuristic knowledge to cover all the area 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 propositional-logic knowledge-base. The first one is based on the formation of an inference-path network for a given goal [6]; and the second one adopts a parallel inference on a hypothetical-lattice structure [8]. In order to improve the efficiency, both of them avoid inefficient backtracking caused by inconsistency among hypotheses. Parallel inference methods similar to ATMS [5] are employed for this purpose.

In general, variable plays an important role to expand the scope of knowledge representation capability. If we represent knowledge in propositional logic with no variable, the scale of 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 above fast hypothetical reasoning systems developed for propositional logic, however, cannot be applicable in a straightforward manner to the predicate-logic case. In this paper we present a fast hypothetical reasoning mechanism effective for predicate-logic knowledge (actually for function-free predicate hom-clause knowledge). A reasoning method developed in deductive database area is effectively applied to this mechanism.

2. 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 knowledge as hypothesis which is defeasible knowledge having the possibility of contradiction with other knowledge. The basic behavior 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 a practical importance of being applicable to many problems such as diagnoses [2,3] and design [4], etc. 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); and the other is a set of hypotheses or incomplete knowledge IK (which is not always true in the world and sometime contradicts with other knowledge). Let G be a given goal, h be a subset of IK. Then the basic function of the system can be written as finding a solution hypothesis set h satisfying the following three logical equations:

\[
\begin{align*}
    h & \subseteq \text{IK} \\
    \text{CK} \cup h & \vdash G \\
    \text{CK} \cup h & \not\vdash \Box
\end{align*}
\]

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' \( \subseteq \) h and h' satisfies the above three logical equations.

3. Inefficiency of Simple Implementation Utilizing Prolog's 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 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 Prolog inference mechanism is not efficient as described below. Figure 2 exemplifies this inefficient inference behavior.

1) 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 on Figure 2, the node j is always false because its child-node m is false. Thus searching the node j branch is in vain.

2) Plural searches for the same branch

This is a general problem of 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 on 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, a backtracking is invoked due 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 the hypothetical reasoning, backtrackings occur more often than in usual inference cases because of the possibility of inconsistent hypotheses. Hence this inefficiency is very serious in the hypothetical reasoning.
3) 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 on Figure 2, there are two nodes f on the inference tree. In the inference process, these nodes f are searched respectively. This is also a general problem of Prolog-based backward inference.

4) Search for redundant solutions
An obtained solution may not be minimal one, i.e., may be redundant solution. Since the Prolog inference mechanism keeps just one solution at a time, it is impossible to know whether or not the solution is minimal. For example on 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 section 4 about fast hypothetical reasoning systems that we had developed for propositional-logic knowledge in order to solve these problems.

4. Fast Hypothetical Reasoning for Propositional-Logic Knowledge
Mainly there are two inference methods, i.e., backward (top-down) inference and forward (bottom-up) one. The backward inference, like in Prolog, has an advantage of searching only goal-related nodes, and a disadvantage of searching the same node more than twice. Especially in the hypothetical reasoning, this disadvantage is very serious because backtracking is invoked frequently due to the inconsistency among hypotheses, as described in section 3.

On the other hand, the forward inference, suitable for searching all the solutions, has an advantage of not searching the same node twice, and a disadvantage of searching the nodes not related to the given goal. In 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 section 3 can be solved by combining the advantages of backward and forward reasonings.

The problem 1) 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 searched. Since synthesizing hypotheses is very expensive in the hypothetical reasoning, it is important to prune non-promising branches (branches involving false nodes) before hypothesis synthesis. This problem can be solved by forming a compiled inference path (backward inference process) before synthesizing hypotheses (forward inference process).

The problem 2) is due to the fact that Prolog-based version holds a single environment. This problem can be solved by using a parallel forward inference with multiple environment to avoid the backtracking caused by inconsistency among hypotheses.

The problem 3) can be solved by merging the identical nodes into one to form a compiled inference-path network.

The problem 4) 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.

(1) A Fast Hypothetical Reasoning Using Inference-Path Network [6]
In this method, 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 hypotheses, are deleted. As a result, an inference-path network can be formed for the given goal. We call this process "Inference-path Formation phase". Problems 1) & 3) 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

![Fig.3 An example of efficient hypothetical reasoning utilizing inference-path network](image)
satisfiability of propositional horn formulae by Dowling & Gallier [7]. Next, hypotheses are synthesized in a forward inference manner along this inference-path network. We call this process "Hypothesis Synthesis phase". Since the hypotheses are synthesized with holding multiple combinations (environments) as in ATMS [5], the problems 2) & 4) are solved. Figure 3 shows a formed inference-path network for the same knowledge-base as described in Figure 2.

(2) A Parallel Inference Utilizing Hypotheses-Lattice Structure [8]

In this method, a given goal is first unfolded using only complete knowledge into some sub-goals. Since always-true and always-false nodes, which truth values are determined regardless of the hypotheses, disappear during this process, the problem 1) is solved. Next, supporting hypotheses for each sub-goal are mapped onto a hypotheses-lattice. Since minimal hypothesis sets can be easily find on the lattice, the problem 4) is solved. Finally, these hypotheses on the lattices are synthesized into solution hypothesis sets. Since there is no backtracking, the problem 2) is solved. This method doesn't compile knowledge' into a network structure; the problem 3) is not solved.

These two hypothetical reasoning systems greatly improved the inference-speed for propositional-logic knowledge. Especially the system (1) achieved an inference-speed thousands times faster than that of the system implemented utilizing the inference mechanism of Prolog.

5. An Extension to Predicate-Logic Knowledge

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

First, we consider an extension of method (1), using the inference-path network [6]. Figure 4 shows an example of the initial inference-path network for a predicate-logic knowledge-base described above the figure. In this figure, different variable-names are assigned to the same node since the variable of each rule is independent. A rule recursion is included in this example.

If a goal is $g_1(a)$, 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 $X/X_1/X_3/a$, $X_4/X_1/b$). This is because $g_1(a)$ and $g_1(b)$ are expressed as the same node $g_1(X)$, though these nodes should be different ones. 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.

Thus we consider an extension of method (2), i.e., parallel inference utilizing hypothetical-lattice structure [8]. However, the unfolding method in the propositional-logic case cannot be applied to predicate-logic case for the following reasons.

(a) There are cases where the unfolding is inefficient without adopting hypotheses.

Since the variables of the nodes may remain as they are unless adopting hypotheses, unnecessary branches not related to the goal may be searched. Figure 5, for example, shows this type of inefficiency with respect to applying the unfolding method. Since the node $g_3(X_1,X)$ would be instantiated to $g_3(a,b)$ with adopting hypotheses $h_1(a)$ and $h_2(b)$, the system needs to prove only the node $g_3(a,b)$ for this node $g_3(X_1,X)$ in this case. However, as the variables $X$ and $X_1$ of the node $g_3(X_1,X)$ cannot be determined without adopting hypotheses, the system goes to prove the node $g_3(X_1,X)$. This search is inefficient.

In this example, the unfolding method without adopting hypotheses searches hypotheses $h_3(a)$, $h_3(b)$, $h_4(a)$ and $h_4(b)$, though only $h_4(a)$ needs to be searched.

(b) There are cases where the unfolding is impossible without adopting hypotheses.

Without determining the variables of the nodes, the unfolding may become impossible especially when some rules are recursive. Recursive rule should be excluded in propositional-logic because the inference doesn't terminate. But in 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 predicate-logic knowledge-base. Figure 6 shows an example where the unfolding is impossible unless adopting hypotheses. The goal could be proved even in this example if all the variables of the nodes are 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 nodes under search. Considering 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 QSQR method [9, 10] in deductive database technology; but KICK-HOPE can also manipulate hypotheses (defeasible knowledge).

6. 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 as pre-processing into a 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 database.). For example, incomplete knowledge "a:-b,c." has been transformed before reasoning into complete knowledge "a:-b,c," and incomplete knowledge "c." Then all incomplete knowledge becomes unit-clauses (fact-type), which is placed at the leaf position of the inference-tree.

The data structure of a node in KICK-HOPE is

\(<\text{Node-Name}, \text{Supporting Hypothesis Sets}>\)

At initial stage, the Node-Name may have variables and the Supporting Hypothesis Sets are undecided. These initial nodes are transformed into settled nodes through the reasoner of KICK-HOPE. (See an example of 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):

1. True node --- always true (no need for hypotheses)
   \(<\text{Node-Name}, \text{true}>\>

2. False node --- always false
   \(<\text{Node-Name}, \text{false}>\>

3. Node where all the supporting hypothesis sets are determined
   \(<\text{Node-Name}, \text{Supporting Hypothesis Sets}>\>

Figure 8 shows the reasoner's behavior of KICK-HOPE. 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. Synthesize 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, respectively. (Settled nodes for both nodes A and 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>>
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 knowledge to be unified:

Case 1 in which the node is unified with rule-type complete knowledge.
Case 2 in which the node is unified with fact-type complete knowledge.
Case 3 in which the node is unified with incomplete knowledge.
Case 4 in which there is no knowledge to be unified.

The return-node is true-node (a settled node).

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

7. 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 the implementation utilizing the inference mechanism of Prolog.

While the CK of the both examples are the same, the IK of Example-2 is an incomplete knowledge set excluding such knowledge as shape h(X,1) from the IK of Example-1. When we represent the scale of knowledge as N of the incomplete knowledge h(X,Y) [X=1,2,..., N-1, Y=1,2,3], then the number of nodes on the inference-tree is 6N+4 for both examples. Example-1 is an example where the same branches are searched plurality 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 where no plural search happens in both KICK-HOPE and the Prolog-based inference.

Figure 11 depicts a result of inference time for Example-1. The inefficiency of the Prolog-based inference is apparent in this figure. For example, the inference-time at N=3 is 0.12 sec in KICK-HOPE and 0.13 sec in the Prolog-based inference mechanism. Whereas at N=15 it is 2.22 sec in KICK-HOPE, it is 2693.86 sec in the Prolog-based inference mechanism. (These data are measured on SUN-4.)
Figure 12 depicts a result of inference time 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) range of the Prolog-based inference mechanism.

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

8. Conclusion

We have described the fast hypothetical reasoning system called KICK-HOPE for function-free predicate-logic horn-clause knowledge. KICK-HOPE has solved the two problems 2) & 4) described in section 3.

The problem 2) regarding wasteful plural search for the same branch is a crucial one especially in hypothetical
reasoning, because of the backtracking invoked by inconsistency among hypotheses. By solving this problem, the inference-speed has been largely improved (See section 7).

For the problem 4), 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 narrow down the right search space.

Thus, the feature of KICK-HOPE is to search just once for only 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 exponentially goes up as seen in Figures 11 and 12. To make KICK-HOPE faster, we should investigate techniques for lowering the cost of merge operation such as synthesizing hypotheses and deleting redundant hypothesis sets. There are some efficient techniques for join operation 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 propositional-logic case, the inference-speed in the worst case cannot exceed the limit of 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 Prof. Randy Goebel (Univ. of Alberta) for his comments. This work was supported by the Grant-in-Aids of Ministry of Education. No.02452154 (B) and No. 02215105 (Special Area on Intelligent Information & Communications).

References