TM-0024

A Methodology for Implementation of A Knowledge Acquisition System

by
H. Kitakami, S. Kunifuji,
T. Miyachi, K. Furukawa

August, 1983

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Technical Memorandum of ICOT Research Center : TM-0024 24-August, 1983

A Methodology for Implementation of a Knowledge Acquisition System

H.Kitakami, S.Kunifuji,
T.Miyachi, K.Furukawa

24-August, 1983

ICOT Research Center

ICOT Research Center
Institute for New Generation Computer Technology

Mita Kokusai Bldg. 21F, 1-4-28 Mita,
Minato-ku, Tokyo, 108
Tel:03-456-3195, Telex:ICOT J32964

A Methodology for Implementation of a Knowledge Acquisition System

H.Kitakami, S.Kunifuji, T.Miyachi and K.Furukawa,

Institute for New Generation Computer Technology (ICOT)

Mitakokusai Building (21F)

1-4-28, Mita, Minato-ku, Tokyo, 108, Japan

[Abstract]

This paper describes an investigation conducted on Knowledge Acquisition System for a Knowledge Base System, and discusses a conceptual configuration and implementation method for some mechanisms in this system. These mechanisms include a meta inference mechanism, inductive inference mechanism, knowledge assimilation mechanism and knowledge accommodation mechanism. These mechanisms enable this system to turn the knowledge into the user's purpose. The discussion of the implementation method for the inductive inference mechanism attempts to explain speeding up strategy.

These mechanisms include the manipulation of facts, rules and integrity constraints as knowledge.

Finally, the authors present certain some execution traces of this system.

1. Introduction

To approximate human mental processes, a computer system must be equipped with various mechanisms. One is a mechanism for systematically storing and managing human knowledge. This mechanism is known as the knowledge base system [1]. As part of our research, we are studing methodologies for implementing a knowledge acquisition system [5]. Knowledge acquisition [2,3,4] means the function of collecting knowledge by assimilation and accommodation in a knowledge base.

In this paper, knowledge stored in a knowledge base is regarded as (i) facts (individual facts), (ii) rules (general rules) and (iii) integrity constraints (integrity constraints on facts and rules) ; and the knowledge base format, as a deductive question-answering system for relational databases.

Relational databases which support SEQUEL, QUEL or other languages as a deductive function handle mainly facts as knowledge [6,7]. A knowledge base, however, also treats rules as knowledge on a full scale basis.

This paper discusses the conceptual configuration and methodology for the implementation of a knowledge-based knowledge acquisition system.

The results of its implementation in Prolog are also reported. The programs were all interpreted under the Edinburgh Prolog-10 [8].

2. Configuration of the knowledge acquisition system

This section discusses the conceptual configuration of a knowledge acquisition system. Knowledge in the knowledge base will basically take the form of Horn clauses. In order to systematically acquire expert knowledge, knowledge acquisition system repeats assimilation and accommodation in the knowledge base. The user then monitors the knowledge base using a deductive question answering mechanism. These operations enable this system to turn the knowledge into the user's purpose.

Figure 1 illustrates the concept of a knowledge acquisition system constructed for knowledge base systems.

This system includes a meta inference mechanism which is defined as an amalgamation of object language(Prolog) and meta language(including demo-predicate). The theoretical study of meta inference is described by Kowalski et al [9,10]. Implementation of this mechanism is discussed in the next chapter.

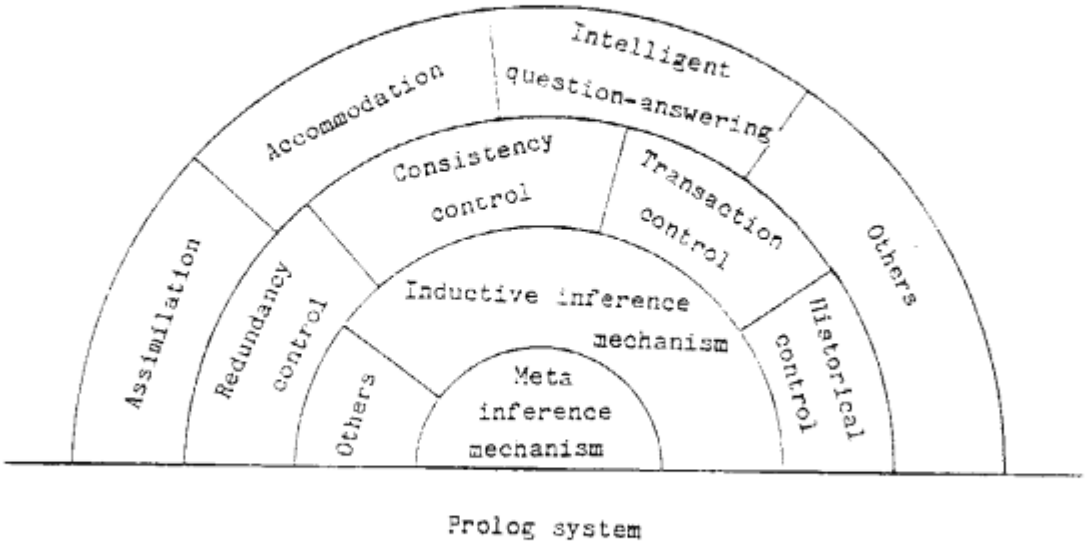

Figure 1. Concept of Knowledge Acquisition System.

The inductive inference mechanism is treated as a model inference drawn from Shapiro's research [17,18] in Prolog. This mechanism on knowledge acquisition can play a role in the following operations :

(1) Creation of knowledge suited to user's intentions.

(2) Revision of existing knowledge(mainly rules and facts).

(3) Self-organizing of existing knowledge.

We are capable of obtaining useful results for (1) and (2) now, but (3) required further research.

Redundancy control is the function for eliminating redundancy, and is carried out at the user's discretion each time new knowledge is acquired by the knowledge base.

History control is the function for solving various past, present and future propositions by recording the assimilation and accommodation processes in the knowledge base.

Consistency control is the function for defense against inconsistencies in the knowledge base.

Assimilation is used to store knowledge suited to the user's intentions in the knowledge base. User's intentions are defined by integrity constraints. In this case, existing knowledge is not revised by assimilating external knowledge.

Accommodation is used to revise existing knowledge using external knowledge supplied by the user, which is always regarded as being correct in this operation. Some accommodation methods are listed below:

(1) Revising existing knowledge based on facts provided by the user;

(2) Revising carried out directly by the user;

(3) Revisions carried out by combining methods (1) and (2);

(4) Deletion of existing knowledge which subsequently becomes unnecessary.

For details, refer to [11].

When two or more assimilations and accommodations are used as a single unit, the operation in a single unit may not satisfy intermediate integrity constraints. Transaction control is required to provide facilities for this purpose.

Intelligent question-answering provides a knowledge base question-answering capability based on meta predicates.

Other functions include those designed to prove inclusion and equality related to knowledge base rules. These are important functions for effective utilization of knowledge bases built in other fields.

3. Meta inference mechanism

A demo-predicate used as a meta inference is used to solve the following problems:

(1) Prove existential propositions.

(2) Prove universal propositions.

(3) Prove goals (Prolog interpreter in Prolog).

(4) Others.

This chapter discusses the implementation of a meta predicate "demonstrate" which solves problems (1) and (2). Existential propositions don't contain variables but universal propositions contain a variable in the argument. The "demonstrate" predicate is used as knowledge assimilation in chapter 5. Problem (3) is described in the next chapter.

Figure 2 illustrates an implementation of the "demonstrate" predicate to prove propositions (1) and (2). These propositions(clauses) are assumed to take the following form:

1) Head: -Goals.

2) Head.

However, "Head" represents one predicate and "Goals" consists of the logical sum and logical product of predicates. "Goals" is allowed to contain the cut operator and the system predicate.

The "demonstrate" predicate in Figure 2 has three arguments. The first argument gives the name list, KBNL, of the knowledge bases to be used; the second argument specifies the existential/universal proposition to be proved; the third argument specifies identification(ID) of knowledge in the existing knowledge base which is not useful to solve this problem.

The "demo" predicate in Figure 2 is known as a Prolog interpreter [15] in Prolog and is improved by the authors. This "demo" predicate has four arguments. The first and second arguments have the same specifications as the "demonstrate" predicate; the third argument gives the following information to control the resolution process:

(1) Temporary knowledge(Goals) used in this process.

(2) Identification(ID) of knowledge as not useful.

(3) Working variable to control the cut operator.

(4) Number of maximum resolution steps.

The fourth argument returns the following resolution results:

(1) Truth value(true/overflow).

(2) Proof tree, if truth value is overflow.

The "select_variable_list" predicate is the predicate that selects the variables from the clause given by the user.

The "skolem" predicate is predicate to provide constants(unique in system) to the variables.

The "clause_kb" predicate is the predicate that searches for the "goals" which corresponds the "Head" from the knowledge base list, KBNL. The "next_clause" predicate is the predicate that searches for the "goals" which corresponds the "Head" from the knowledge base, KBN.

The physical structure of knowledge is as follows:

KBN(P1,P2,P3,Clause).

Each piece of knowledge in the knowledge base is linked by a before pointer P3 and forward pointer P2. P1 is the address pointer of the clause. Identifier ID of knowledge is defined by the three pointers [P1,P2,P3].

```
/* Demonstrate existential/universal Clause */

demonstrate(KBNL,Clause,Cond):-
  verify( (select_variable_list(Clause,Variable_list),
   skolem(Variable_list),
   (Clause=(P:-Q)->demo(KBNL,P,[Q,Cond,Cut,50],[true,[]]);
    demo(KBNL,Clause,[[],Cond,Cut,50],[true,[]])) ).

verify(P):- \+(\+(P)).

demo(KBNL,true,[Res,Cond,Cut,Count],[true,[]]):-!.
demo(KBNL,Goals,[Res,Cond,Cut,0],[overflow,[]]):-!.
demo(KBNL,!,[Res,Cond,Cut,Count],[Result,Stack]).
demo(KBNL,!,[Res,Cond,cut,Count],[Result,Stack]).
demo(KBNL,(P;Q),[Res,Cond,Cut,Count],[Result,Stack]):-!,
    (demo(KBNL,P,[Res,Cond,Cut,Count],[Result,Stack]) ;
     demo(KBNL,Q,[Res,Cond,Cut,Count],[Result,Stack])).
demo(KBNL,(P,Q),[Res,Cond,Cut,Count],[Result,Stack]):-!,
    demo(KBNL,P,[Res,Cond,Value,Count],[Result1,Stack1]),
    (Value==cut,Cut=cut,Result=Result1,Stack=Stack1,! ;
     Result1=true->demo(KBNL,Q,[Res,Cond,Cut,Count],[Result,Stack]) ;
     Result=Result1,Stack=Stack1).
demo(KBNL,P,[Res,Cond,Cut,Count],[Result,Stack]):-
    system(P)->P,Result=true,Stack=[] ;
    Count1 is Count-1,
    clause_kb(KBNL,P,ID,Q,[Res,Cond]),
    demo(KBNL,Q,[Res,Cond,Cut,Count1],[Result1,Stack1]),
    (Cut==cut,!,fail ;
     Result1=true->Result=true,Stack=[] ;
     Result=overflow,Stack=[(P:-Q)|Stack1]).

clause_kb((KBN,KBNL),Head,ID,Goals,[Res,Cond]):-!,
     ( clause_kb(KBN,Head,ID,Goals,[Res,Cond]) ;
       clause_kb(KBNL,Head,ID,Goals,[Res,Cond]) ).
clause_kb(KBN,Head,ID,Goals,[Res,Cond]):-
     next_clause(KBN,ID,Clause,Cond),
     ( Clause=(Head:-Goals) ;
       Clause=Head,Goals=true  ) .
clause_kb(KBN,Head,[],true,[Res,Cond]):-
     Res\==[],unify(Res,Head).
```

Figure 2. A Prolog Program for the "demonstrate" Predicate for Proof of Existential/Universal Propositions.

4. Inductive inference mechanism

The inductive inference mechanism in Figure 1 will be treated as a problem concerning inductive model inference from facts. This concept of model inference draws on Shapiro's research [17,18].

This section discusses our research into model inference, and presents the relations between "demo" predicate and the model inference. We also discuss a speeding up strategy for the model inference.

4.1 Model inference

Figure 3 shows an incremental model inference algorithm presented by Shapiro. This algorism is given observation data Fn=<OBS,V>.

```
Sfalse={[]},Strue={ }.
L0={[]},mark [] "false".
repeat
   Read the next observation data Fn=<OBS,V> and add OBS to Sv.
  repeat
     while { Derive OBS from conjecture Lp, OBS belongs to Sfalse } do
        By contradiction backtracing,
       discover the refuting hypotheses H and delete H from Lp.
     while { ~(Derive OBSi from conjecture Lp), OBSi belongs to Strue } do
        By refinement operation,
       discover hypotheses Hi which is satisfiable to derive Hi to OBSi and
       add Hi to Lp ( Lq = Lp ^ Hi, q=p+1 ).
  until { Neither of the while loops is entered }
  output Lp
forever
```

Figure 3. An Incremental Model Inference Algorithm.

The observation sentence OBS is a fact and V is the truth value of "true" or "false". The set of inferable hypotheses is the procedures (set of rules and facts) programmed in Prolog.

The derivation problem in Figure 3 can be implemented by the "demo1" predicate shown in Figure 4. In this way, the physical separation of the knowledge base can be obtained for the model inference system. Figure 5 shows a Prolog program which applies a "demo1" predicate to Figure 3.

```
demo1(KBNL,Goals):-
     demo(KBNL,Goals,[[],[],Cut,50],[Result,Stack]),
     (Result\==true,! ; true),
     (Result=overflow ->nl,stack_overflow(KBNL,Goals,Stack),
      demo1(KBNL,Goals) ; true).
```

Figure 4. A Prolog Program for The "demo1" Predicate.

```
model_inference(KBN,FN) :-
   nl,ask_for('Next fact(sentence,true/false) or end ',Fact),
   ( Fact=end ;
   ( Fact=check -> model_inference1(KBN,_) ;
     ( Fact=(P,V),P=..[FN|Y];
       nl,write(' Error Functor Name.'),fail ),!,
       Fact=(P,V), (V=true ; V=false)  ->
       assert_fact(P,V), model_inference1(KBN,P) ;
    write('!Illegal input'), nl ),!,nl,next_time,
   model_inference(KBN,FN)).
model_inference(KBN,FN):-
   nl,ask_for('Next fact(sentence,true/false) or end ',Fact),
   ( Fact=end ;
   ( Fact=check->model_inference1(KBN,_);
     Fact=(P,V),(V=true;V=false) ->
     assert_fact(P,V),model_inference1(KBN,P);
    write('!Illegal input'),nl ),!,nl,next_time,
   model_inference(KBN,FN)).

model_inference1(KBN,P) :-
   write('Checking fact(s)...'), ttyflush,
   ( fact(P,true),not(demo1(KBN,P))->
     nl,missing_solution(KBN,P),model_inference1(KBN,_) ;    % (Too Weak)
     fact(P,false),demo1(KBN,P)->
     nl, false_solution(KBN,P), model_inference1(KBN,_) ;    % (Too Strong)
     write('no error found.'),nl ).
```

Figure 5. A Prolog Program for Model Inference System.

The refinement operation in Figure 3 is capable of refining predicate "p" to predicate "q" in the following four ways:

(1) q(X1,X2,X3,...,Xn), if p=[].

(2) q=Unify (p(X1,X2,X3,...,Xn)).
(Xi=Xj)

(3) q=Instatiate(p(X1,X2,X3,...,Xn)).
(Xi=t(Y1,Y2,...,Ym))

(4) q=Add_goal(p(X1,X2,X3,...,Xn)).
(Subset of p)

4.2 Speedup strategies

The speeding up strategy for model inference is as follows:

(1) Type(structural) specifications of predicate arguments.

(2) Specifications of the incorporating predicate.

(3) I/O specifications of predicate arguments.

(4) Selecting a hypothesis with a simple structure.

(5) Preventing recomputation by keeping refuted hypotheses.

(6) Others.

As strategies (1)-(4) have already been discussed in model inference system [17], this section will discuss strategy (5) in Figure 6. We assume that the search strategy for the refinement graph in Figure 5 is a breadth-first search and our goal is to search for hypothesis "Hj". If the hypothesis is refuted by a false sentence, the system can search for a new hypothesis, "H8", without recomputation from "H1" by keeping hypothesis "H6". The results of this experiment are shown later.

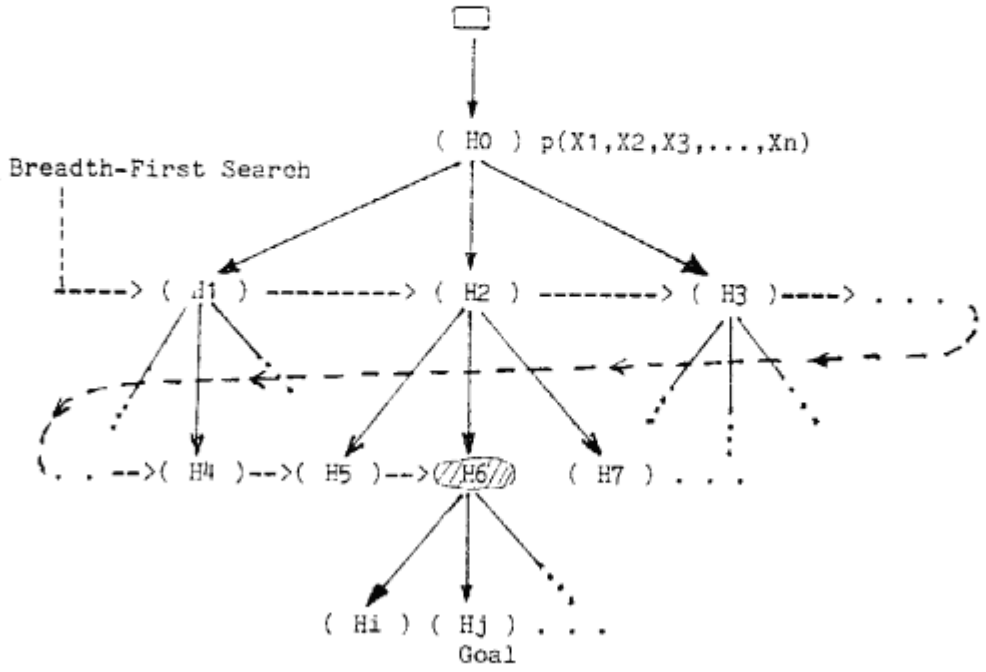

Figure 6. The Speeding up Strategy in Refinement Graph.

5. Knowledge assimilation

This section describes an algorithm for assimilation as shown in Figure 1 on the basis of Kowalski's idea [12,13]. The algorithm is predicated on the following major assumptions:

1) The closed-world assumption is employed.

2) The knowledge base under consideration is consistent.
(Integrity constraints are also consistent.)

3) Knowledge (rules, facts and integrity constraints) in the knowledge base are described in Prolog.

The assimilation algorithm is organized as follows:

(A1) If input knowledge can be proved from the knowledge base, do not assimilate the knowledge.

(A2) Remove redundancy from the knowledge base on the assumption that input knowledge has been assimilated into the knowledge base.
However, redundancy removal processing depends on user's intention.

(A3) If the integrity constraint concerned can't be proved from the knowledge base on the assumption that input knowledge has been assimilated into the knowledge base, restore the knowledge base to its original state because it is inconsistent.
(In (A3), redundancy may be removed.)

(A4) If input knowledge qualifies for assimilation under all of procedures (A1) through (A5) , insert the input knowledge into the knowledge base.

Figure 7 shows a Prolog program for assimilation by this algorithm.

6. Knowledge accommodation

Knowledge accommodation for a knowledge base is similar to the debugging of a logic programming and this accommodation includes the functions of revising existing knowledge from facts and replacing directly existing knowledge with new knowledge. For this operation, this accommodation is necessary in the following procedures:

(1) Revision of Knowledge in the knowledge base.

(2) Redundancy elimination (depends on user's intention).

(3) Knowledge base recovery by mean of discovering contradictions.

```
/* (A1) : Is "Input" derivable from "KBN" ?          */
/*                                                   */
assimilate(KBN,IC_name,Input):-
   demonstrate(KBN,Input,[]),
   message(derive,Input).

/* (A2) : Does "Input" imply information already     */
/*                implicitly contained in "KBN"  ?   */
assimilate(KBN,IC_name,Input):-
   insert_knowledge(KBN,Input,ID),
   assert(inserted(KBN,ID,Input)),
   query_redundancy_elimination,
   eliminate_redundancy(KBN,ID),fail.

/* (A3) : Is ["KBN" + "Input"] inconsistent          */
/*                                  with "IC"  ?     */
assimilate(KBN,IC_name,Input):-
   IC_name\==[],
   next_clause(IC_name,ID,Clause,[]),
   head_part(Clause,Head),
   not(demonstrate((IC_name,KBN),Head,[])),
   restore_kb(KBN),
   message(inconsist,Input).

/* (A4) : "Input" is logically independent from      */
/*                                          "KBN" .  */
assimilate(KBN,IC_name,Input):-
   post_processing(KBN,Input),
   message(independ,Input).

/* Eliminate redundancy */
eliminate_redundancy(KBN,ID):-
   next_clause(KBN,ID1,Clause,[ID]),
   demonstrate(KBN,Clause,[ID1]),
   update_knowledge(KBN,[],ID1),
   assert(updated(KBN,ID1,Clause)),
   message(redundant,Clause),
   fail.

/* Head part */
head_part(Clause,Head):-
   (Clause=(Head:-Goals),!;Clause=Head).
```

Figure 7. A Prolog Program for Knowledge Assimilation.

7. Example of knowledge acquisition program executions

This chapter presents two examples using the well-known building block problem. One is a knowledge assimilation problem and the other is model inference problem used in the previously described speedup strategies.

7.1 Problem setting

Figure 8 shows an arrangement of building blocks. Suppose that towers are built on floor a, using building blocks b, c,..., h and i. Building block f is rectangular, the others square. Assume further that building blocks j and k are in hand, characterized by the absence of data related to j and k in the predicate called the relation "on". j is rectangular, k square.

The Prolog program in Figure 9 shows location relations of the building blocks presented in Figure 8. Each building block is represented by the predicate "block"; their stack relations, by the predicate "on".

The facts in this knowledge base are the predicates "rectangular_block", "square_block", "floor", and "on"; and the rules are the predicates "block" and "tower". The knowledge base is named "build".

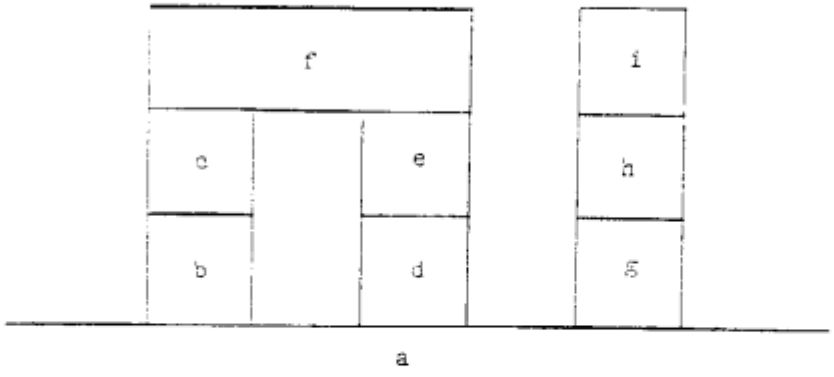

Figure 8. Arrangement of Building Blocks.

```
/* rectangular blocks */

floor(a).
rectangular_block(f).
rectangular_block(j).

/* square blocks */

square_block(b).
square_block(c).
square_block(d).
square_block(e).
square_block(g).
square_block(h).
square_block(i).

/* block */

block(x):-
   square_block(x);
   rectangular_block(x);
   floor(x).

/* on(X,Y) : X is on Y */

on(b,a).
on(d,a).
on(c,b).
on(f,c).
on(e,b).
on(f,e).
on(g,a).
on(h,g).
on(i,h).

/*   tower(X,Y) : The tower consists of block X */
/*              on top of tower Y.               */

tower(X,Y):-tower1(X,Y), Y\==[].
tower(X,[]):-floor(X).
tower(X,[Y|Z]):-block(X),on(X,Y),tower1(Y,Z).
```

Figure 9. Knowledge Base Represented Location Relations of Building Blocks.

The following restrictions are assumed as integrity constraints (ICs).

(1) "floor(a)" is never deleted.

(2) "rectangular_block" is supported by two or more "towers".

(3) In the "tower" in (2) there are one or more "tower" consisting solely of "square_block".

Figure 10 shows these relations.

```
/* Integrity Constraints */

consistent(1):-floor(a).

consistent(2):-
   \+(demo1(build,(rectangular_block(X),
   setof_kb(build,[Y],(tower(X,Y),ne(Y,[])),S),
   length(S,N),N<2))).

consistent(3):-
   \+(demo1(build,(rectangular_block(X),
   setof_kb(build,[Y],(tower(X,Y),ne(Y,[])),S),
   length(S,N1),N1<2,
   setof_kb(build,[V],(tower(X,V),on(X,W),
         square_block(W),ne(V,[])),S2),
   length(S2,N2),N2=0))).
```

The "setof_kb" predicate is the predicate defined below:

```
setof_kb(KBNL,X,Goals,Set):-setof(X,demo1(KBNL,Goals),Set).
```

Figure 10. Integrity Constraints on Building Block Knowledge Base.

From the foregoing, the program is executed to solve the following problems:

(Problem 1) Problems of integrity constraints related to assembly of building blocks.

(Problem 2) Accommodation of the predicate "above"

(Problem 3) Model inference problem of "arch".

Results of the execution are given in the Appendix. Accommodation omits integrity constraint checks for simplicity's sake. The Shapiro's model inference system cant't solve the Problem 3 under the Edinburgh prolog-10 but the our system can solve it in 64.852 CPU seconds from 2 facts.

8. Conclusion

We have discussed a conceptual configuration for a knowledge-based knowledge acquisition system and presented results of its implementation in Prolog. The implementation parts in this system are demo-predicate, assimilation and model inference system modified for speed.

All programs are interpreted under the Edinburgh Prolog-10 on a DEC2060.

Our future research plans will concern the following subjects:

(1) A method for implementing knowledge assimilation which allows input of two or more clauses of the knowledge, instead of only one as discussed here, in each processing run.

(2) Generalization and speedup of redundancy removal.

(3) Speedup of inductive model inference.

(4) Inductive inference for integrity constraints from facts.

(5) The problem of proving equality between concepts in knowledge bases.

(6) Speedup of processing through introducing a parallel computation mechanism.

[Acknowledgements]

The authors wish to thank K.Fuchi, Director, ICOT Research Center for providing the opportunity for this research and M.Asou, First Research Laboratory of ICOT, for their useful recommendations.

[References]

[1] Fuchi, K.: Problem Solving and Inference Mechanism, Information Processing Society Journal Vol. 19 No.10 (in Japanese) (1978).

[2] McDermott, J.: ANA: An Assimilating and Accommodating Production System, Tech. Rept. CMU-CS-78-156, Carnegie-Melon University, Dept. of Computer Science (1978).

[3] Grumbach, A.: Knowlegde Acquisition in Prolog, Proceeding of the First International Logic Programming Conference in France (1982).

[4] Shimura, M.: Knowledge Acquisition and Learning, Telecomunications Society Technical Report (in Japanese) AL80-46 (1980).

[5] Miyachi, T., Kunifuji, S., Kitakami, H., Furukawa, K., Takeuchi, S., Yokota, H.: A Knowledge Assimilation Method for Logic Knowledge Bases, Research Institute for Mathematical Sciences(RIMS) of Kyoto University, Symposium on "A Study in Theory and Practice of Model Representation and its Construction" Seminar report (in Japanese) (1983).

[6] Chamberlin, D.D. et al.: A Unified Approach to Definition, Manipulation, and Control, IBM J. RES. DEVELOP. (1976).

[7] Stonebraker, M. et al.: The Design and Implementation of INGRES, Memo. No. ERL-M 577, Univ. of California, Berkeley (1976).

[8] Bowen, D.L.: DEC-10 PROLOG USER'S MANUAL, University of Edinburgh (1981).

[9] Bowen, K.A., Kowalski, R.A.: Amalgamating Language and Meta Language in Logic Programming, School of Computer and Information Sciences, University of Syracuse (1981).

[10] Kunifuji, S., Asou M., Sakai K., Miyachi, T., Kitakami, H.,Yokota, H., Yasukawa, H., Furukawa, K.: Amalgamation of Object Knowledge and Meta Knowledge in Prolog and its Application, Information Processing Society Research Committee Material, "Knowledge Engineering and Artificial Intelligence" (in Japanese) (1983).

[11] Kitakami, H., Miyachi, T., Kunifuji, S., Furukawa, K.: The concept of Knowledge Base System KAISER, 26th National Conference of Information Processing Society (in Japanese) (1983).

[12] Eswaran, K.P. and Chambelin, D.D.: Functional Specification of a System for Knowledge Base Integrity, Proc. 1st VLDB Conf. (1975).

[13] Kowalski, R.A.: Logic as a Knowledge Base Language,Department of Computing, Imperial College (1981).

[14] Kowalski, R.A.: Logic for Problem Solving, Elsevier North Holland, Inc., pp. 239-246 (1979).

[15] Coelho, H., Cotta, J.C. and Pereira, L.M.: How to Solve It with PROLOG (2nd. edition), Laborato'rio Nacional de Engenharia Civil, Lisboa (1980).

[16] Mitchell, T.M.: Version Spaces: An Approach to Concept Learning, Stanford CS Report, STAN-CS-78-711 (1978).

[17] Shapiro, E.Y.: Inductive Inference of Theories From Facts,Technical Report 192, Department of Computer Science, Yale University (1981).

[18] Shapiro, E.Y.: Algorithmic Program Debugging, Research Report 237, Yale University (1982).

[19] Doyle, J.: Truth Maintenance System for Problem Solving, AI-TR-419 (1978).

[Appendix]

(Problem 1)

```
$ ?- assimilate(build,ic,on(j,i)).

* Eliminate Redundancy ? (y/n) n

* The Input_Knowledge "on(j,i)"
 is inconsist with the Integrity_Constraints.

 2.130 sec.

yes
$ ?- assimilate(build,ic,on(a,i)).

* Eliminate Redundancy ? (y/n) n.

* The Input_Knowledge "build.(i)"
 was assimilated.

 3.868 sec.

yes
```

(Problem 2)

```
$ ?- assimilate(build,[],[above(X,Y):-on(X,Y)]).

* Eliminate Redundancy ? (y/n) n.

* The Input_Knowledge "[above(_0,_1): on(_0,_1)]"
 was assimilated.

 0.228 sec.

yes
$ ?- assimilate(build,[],[above(X,Z):-on(X,Z),on(Z,Y)]).

* Eliminate Redundancy ? (y/n) y.

* The Input_Knowledge "[above(_0,_2):-on(_0,_2),on(_2,_1)]"
 was assimilated.

 10.646 sec.

yes
$ ?- accommodate(build,[],above(X,Y)).

* Initialize parameter(y/n)? n

* The Knowledge_base "solutions" was created on Core_Space.

* The Knowledge_base "rejected" was created on Core_Space.

Next fact(sentence,true/false) or end ? above(i,a),true.
Checking fact(s)...
Error: missing solution above(i,a). diagnosing...

Query: on(i,a)? n.

Query: on(h,a)? n.
Error diagnosed: above(i,a) is uncovered.

Searching for a cover to above(i,a)...
Refining: (above(X,Y):-true)
Refining: (above(X,Y):-on(X,U))
Refining: (above(X,Y):-on(X,U),on(U,W))
Refining: (above(X,Y):-on(X,U),on(U,W))
Refining: (above(X,Y):-on(X,U),above(U,W))
Checking: (above(X,Y):-on(X,U),above(U,Y))
Found clause: (above(X,Y):-on(X,U),above(U,Y))
    after searching 20 clauses.
Listing of above(Y,Y):
    (above(X,Y):-on(X,Y)).
    (above(X,Y):-on(X,U),on(U,Y)).
    (above(X,Y):-on(X,U),above(U,Y)).

Checking fact(s)...no error found.

5.37 sec.

Next fact(sentence,true/false) or end ? end.

* Purge Facts(y/n) ? n.

* Refuted Theory(y/n) ? n.

* Eliminate Redundancy ? (y/n) y.

* The Knowledge "[above(_0,_1):-on(_0,_2),on(_2,_1)]"
 was eliminated from Knowledge_Base "build" .

* Eliminate rejected theory? (y/n) n.

* Construction of the Knowledge "above"
 has been finished.  17.680 sec.
```

(Problem 3)

```
$ ?- create(build,[?.arch(____)]).

* Initialize parameter(y/n)? n

* The Knowledge_base "solutions" was created on Core_Space.

* The Knowledge_base "rejected" was created on Core_Space.

Next fact(sentence,true/false) or end ? arch(f,[a,b,a],[a,d,a]),true.
Checking fact(s)...
Error: missing solution arch(f,[c,b,a],[e,d,a]). diagnosing...
Error diagnosed: arch(f,[c,b,a],[e,d,a]) is uncovered.

Searching for a cover to arch(f,[c,b,a],[e,d,a])...
Refining: (arch(X,Y,Z):-true)
Refining: (arch(X,[Y|Z],U):-true)
Refining: (arch(X,Y,[Z|U]):-true)
Refining: (arch(X,Y,Z):-tower(X,Y))
Refining: (arch(X,[Y|Z],[U|V]):-true)
Refining: (arch(X,[Y,Z|U],V):-true)
Refining: (arch(X,[Y|Z],U):-tower(X,W))
Refining: (arch(X,[Y|Z],[U|V]):-true)
Refining: (arch(X,Y,[Z,U|V]):-true)
Refining: (arch(X,Y,[Z|U]):-tower(X,W))
Refining: (arch(X,Y,Z):-tower(X,Y),tower(Y,Z))
Checking: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X))
Found clause: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X))
    after searching 8 clauses.
Listing of arch(X,Y,Z):
    (arch(X,Y,Z): tower(X,Y),tower(X,Z),rectangular_block(X)).
Checking fact(s)...
Error: wrong solution arch(f,[e,b,a],[c,b,a]). diagnosing...
Error diagnosed: (arch(f,[c,b,a],[e,b,a]):-
 tower(f,[e,b,a]),tower(f,[c,b,a]),rectangular_block(f)) is false.
Listing of arch(X,Y,Z):

Checking fact(s)...
Error: missing solution arch(f,[c,b,a],[e,d,a]). diagnosing...
Error diagnosed: arch(f,[c,b,a],[e,d,a]) is uncovered.

Searching for a cover to arch(f,[c,b,a],[e,d,a])...
Refining: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X))
Checking: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X))
Found clause: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X))
    after searching 8 clauses.
Listing of arch(X,Y,Z):
    (arch(X,Y,Z): tower(X,Y),tower(X,Z),rectangular_block(X)).
Checking fact(s)...
Error: wrong solution arch(f,[e,b,a],[c,b,a]). diagnosing...
Error diagnosed: (arch(f,[c,b,a],[e,b,a]):-
 tower(f,[e,b,a]),tower(f,[c,b,a]),rectangular_block(f)) is false.
Listing of arch(X,Y,Z):

Checking fact(s)...
Error: missing solution arch(f,[c,b,a],[e,d,a]). diagnosing...
Error diagnosed: arch(f,[c,b,a],[e,d,a]) is uncovered.

Searching for a cover to arch(f,[c,b,a],[e,d,a])...
Refining: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X))
Checking: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X),Y\==Z)
Found clause: (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X),Y\==Z)
    after searching 9 clauses.
Listing of arch(X,Y,Z):
    (arch(X,Y,Z):-tower(X,Y),tower(X,Z),rectangular_block(X),Y\==Z).

Checking fact(s)...no error found.

52.771 sec.

Next fact(sentence,true/false) or end ? end.

* Purge Facts(y/n) ? n.

* Refuted Theory(y/n) ? n.

* Eliminate Redundancy ? (y/n) n.

* Eliminate rejected theory? (y/n) n.

* Construction of the Knowledge "arch"
 has been finished.  64.852 sec.

$ ?- demo(build,arch(X,Y,Z)).

arch(f,[c,b,a],[e,d,a])  ;

arch(f,[e,d,a],[c,b,a])  ;

no
```