A BELIEF REVISION SYSTEM FOR LOGIC PROGRAMS

Taher Ali[1], Ziad Najem[2], and Mohd Sapiyan[1]

[1]Department of Computer Science, Gulf University for Science and Technology, Kuwait
ali.t@gust.edu.kw, sapiyan.m@gust.edu.kw
[2]Department of Computer Science, Kuwait University, Kuwait
najem@cs.ku.edu.kw

ABSTRACT

Search is one of the most important needs of problem solvers. Usually the problem solvers suffer from retracing conclusions. If a problem solver cached its inference, then it would not need to retrace conclusions that it had already derived earlier in the search. By caching the inferences, the problem solver avoid throwing away useful results and avoid wasting effort rediscovering the same things over and over. In this paper we present a belief revision system for logic programs that can work under the non-monotonic logic.

KEYWORDS

Applications of justification-based truth maintenance systems, Belief revision systems, Truth maintenance systems, Justification-based truth maintenance systems, Incremental evaluation of tabled Prolog, Incremental tabulation for Prolog queries, Tabulation for logic programs, Memoing for logic programs.

1. INTRODUCTION

Truth Maintenance Systems [1], or Tms, are used within Problem Solving Systems [2], in conjunction with Inference Engines (IE) such as rule-based inference systems like Prolog (SWI-Prolog [3], Gnu-Prolog [4], B-Prolog [5], XSB [6], Ciao [7] and SICStus-Prolog [8]), to manage the inference engine's beliefs in given sentences as a Dependency Network. Figure 1 gives an an overview of Problem Solving Systems that uses Tms along with IE. A Tms is a knowledge representation method for representing both beliefs and their dependencies. A Tms is intended to satisfy a number of goals. One of these goals is the ability to remember derivations computed previously. It may happen that the same question is being asked from the problem solver over and over. If the previous knowledge is not cached when the question was answered for the first time, then the IE needs to re-compute the knowledge again and again. But if the previous knowledge was in the knowledge base, then there is no need for retracing the same knowledge. The use of Tms can avoid such retracing.

David C. Wyld et al. (Eds) : COSIT, DMIN, SIGL, CYBI, NMCT, AIAPP - 2014
pp. 227–231, 2014. © CS & IT-CSCP 2014 DOI : 10.5121/csit.2014.4922

Jtms is the simplest type of Tms where one can examine the consequences of the current set of assumptions. Jtms is a domain-independent belief revision system [9] which is usually coupled with an inference engine that does the actual inference work. Jtms operates on propositional objects and is used to record and maintain dependencies between deductive inferences. This can be done by representing deductive dependencies as a Jtms network.

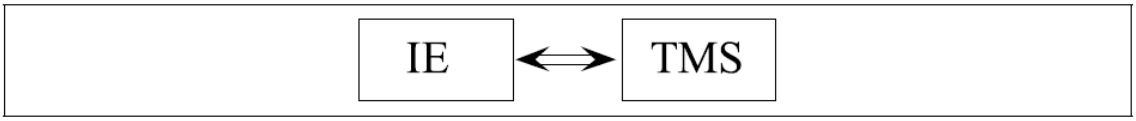

Figure 1: An overview of Problem Solving Systems

2. JTMS NODES AND JUSTIFICATIONS

The basic Jtms specification can be given in terms of two sets: the set of enabled assumptions (domain propositions) and the set of justifications. Propositions of the domain are mapped into nodes where each node is labeled either IN or OUT depending on whether or not it is currently assumed. Nodes corresponding to propositions that the system currently believes in are labeled IN while currently disbelieved propositions are labeled OUT. A node is labeled OUT by default but Jtms may label a node as IN in exactly two cases: either by a request from the inference engine or if there exists an active justification that supports the node.

In order to form a Jtms network the nodes are linked by justifications. A justification is a structure that is responsible for recording a single inference. A justification has two sets of nodes, the in -list and the out - list as its antecedent and a single node as its consequent. A justification is said to support its consequent node. Note that it is possible to have multiple justifications supporting the same node. An active justification is a justification where all the nodes in the in - list are labeled IN and all the nodes in out -list are labeled OUT. The consequent node of an active justification will be labeled IN while the consequent node of an inactive justification will be labeled OUT unless it has another active justification supporting it.

3. JTMS AND NON-MONOTONIC LOGIC

Search is one of the most important needs of problem solvers. Usually the problem solvers suffer from retracing conclusions. If a problem solver cached its inference, then it would not need to retrace conclusions that it had already derived earlier in the search. By caching the inferences, the problem solver avoid throwing away useful results and avoid wasting effort rediscovering the same things over and over. One of the Jtms goals, inherited from the Tms, is that Jtms is able to remember derivations computed previously. Jtms can do this by caching the inferences. This effort of Jtms can help in implementing incremental tabling[10] features for Prolog that will work with non-monotonic logic [11, 12] programs. The idea is that instead of remembering the end results as traditional memoing implementations does, the Prolog inference engine caches its inferences by the help of Jtms. By caching the inferences, Jtms will reflect any change in data through its network to keep the inferences updated. The responsibility of Jtms is answering queries correctly with respect to the contents of Jtms nodes and justifications at the moment the query is made.

Example 1

Figure 2 shows an example of an Sldnf-resolution [13] . The Sld-derivation [14] tree is for the query ?-bachelor(X) with respect to the Prolog program of Figure 2. There are two branches in the main proof structure with only one being successful. The only answer generated for the query ?-bachelor(X) is coming from the second branch. Figure 3 shows

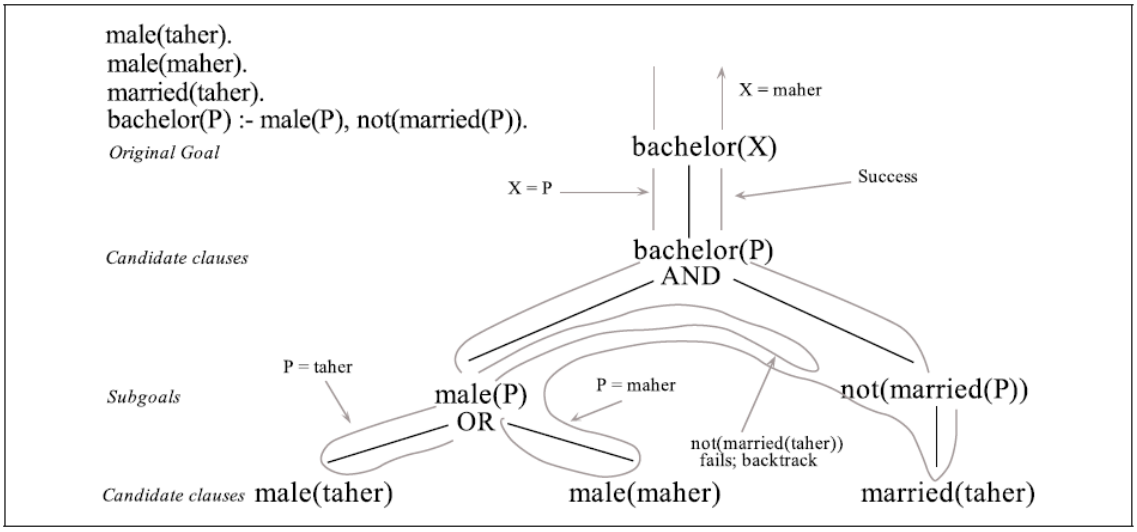

Figure 2: An example of Sldnf-dervation tree.

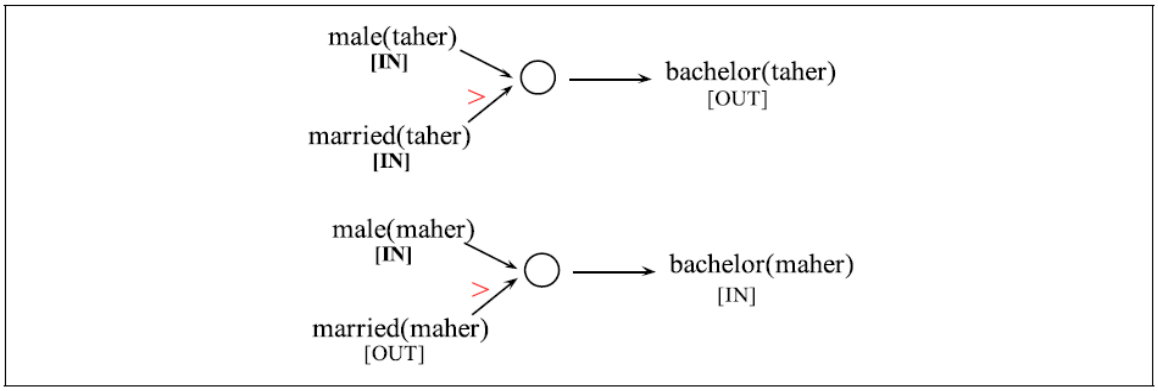

Figure 3: Jtms network for proof tree of Figure 2

an example of a justification network for the proof tree in Figure 2. Nodes are shown by printing their corresponding domain atoms. Justifications are shown as circles. The antecedents of a justification are identified by arrows pointing towards the justification while the consequent is pointing away from the justification. A negative literal in the antecedent (i.e. a member of the justifications out - list) is identified by placing a : sign on top of the arrow pointing to the justification. Figure 3 also shows the current labeling of the nodes. Labels that are printed in bold are specified by the inference engine while the

rest are assigned by the Jtms. Note that Jtms network of Figure 3 has two justifications that correspond to the two proof branches of Figure 2 proof tree, whether or not the proof branch was successful. This will allow the Jtms network to reflect any changes in data. (i.e. the query results are always correct and updated).

Coming back to the example of Figure 3, consider that married(maher) is asserted to the database of Prolog facts in Figure 2. Jtms reflect this change through it's network in order to keep the network updated. Jtms is capable of updating (revising) its belief bachelor(maher) without invoking the inference engine by propagating the changed value through the network. The resultant network is shown in Figure 4.

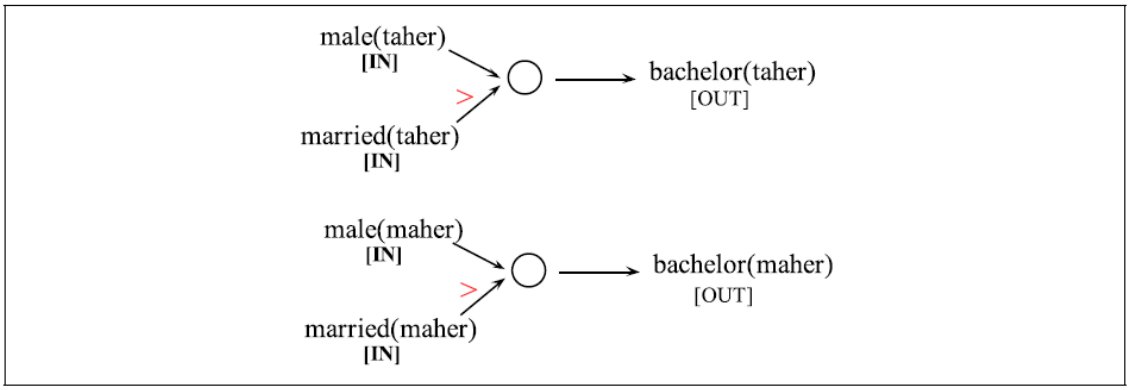

Figure 4: Jtms network of Figure 3 after asserting married(maher) to the database of Prolog facts in Figure 2.

4. CONCLUSION

The main idea of our approach presented in this papaer is to cache the proof generated by the deductive inference engine. The proof structure is converted into a justification based truth-maintenance (Jtms) network. Jtms saves the dependency between deduced facts and the facts used to make the deduction in order to be able to efficiently cache the proof structure. The system translates every successful branch of a query into a Jtms network that links the facts and the rule used in the branch to the answer generated by that branch. A justification is installed for each complete branch of the SLD-tree. When a query is re-evaluated, the system returns the answers of the query by collecting the IN consequences of each query's Jtms justification.When changes in database of facts take place, the system propagtes the effect of the changes through the Jtms network to ensure that the proof structure is both correct and complete.

REFERENCES

[1] Jon Doyle. A truth maintenance system. Artif. Intell., 12(3):231–272, 1979.
[2] Kenneth D. Forbus and Johan de Kleer. Building problem solvers. MIT Press, Cambridge,MA, USA, 1993.
[3] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. Theoryand Practice of Logic Programming, 12(1-2):67–96, 2012.
[4] Daniel Diaz, Salvador Abreu, and Philippe Codognet. On the implementation of gnu prolog.TPLP, 12(1-2):253–282, 2012.

[5] Neng-fa Zhou. The language features and architecture of b-prolog. Theory Pract. Log. Program.,12(1-2):189–218, January 2012.
[6] Konstantinos F. Sagonas, Terrance Swift, and David Scott Warren. The xsb programming system. In Workshop on Programming with Logic Databases (Informal Proceedings), ILPS,page 164, 1993.
[7] Manuel V. Hermenegildo, Francisco Bueno, Manuel Carro, Pedro López-García, Edison Mera,José F. Morales, and German Puebla. An overview of ciao and its design philosophy. CoRR,abs/1102.5497, 2011.
[8] Mats Carlsson and Per Mildner. Sicstus prolog – the first 25 years. CoRR, abs/1011.5640,2010.
[9] Stuart C. Shapiro. Belief revision and truth maintenance systems: An overview and a proposal.Technical report, 1998.
[10] Diptikalyan Saha. Incremental evaluation of tabled logic programs. PhD thesis, Stony Brook,NY, USA, 2006. AAI3258884.
[11] Guido Boella and Leendert W. N. van der Torre. A non-monotonic logic for specifying and querying preferences. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, IJCAI, pages 1549–1550. Professional Book Center, 2005.
[12] Drew McDermott. Nonmonotonic logic ii: Nonmonotonic modal theories. J. ACM, 29(1):33–57, January 1982.
[13] David H. D. Warren. An abstract prolog instruction set. Technical Report 309, AI Center,SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Oct 1983.
[14] Stefan Brass. Magic sets vs. sld-resolution. In Johann Eder and Leonid A. Kalinichenko, editors, ADBIS, Workshops in Computing, pages 185–203. Springer, 1995.

AUTHORS

Tahir M. Ali received his BSc and Ms from Kuwait University and PhD from University of Malaya. He is currently an Assistant Professor of Computer Science in G ulf University for Science and Technology, and also serving as the IT director. His main research interest is in field of Artificial Intelligence (AI), in particular, logic programming and scheduling algorithms.

Ziad H. Najem received his BSc from Kuwait University and Ms and PhD from University of Illinois at Urbana-Champaign. Prior to joining the Department of Computer Science at Kuwait University in 1999, Dr. Najem worked as a Scientific Resear cher at Kuwait Institute
for Scientific Research.

Mohd Sapiyan Baba is currently a Professor of Computer Science in Gulf University for Science and Technology, Kuwait. He was a lecturer in University of Malaya for more than 30 years, teaching Mathematics and Computer Science courses, and supervised numerous students for their research projects at undergraduate and postgraduate levels. His main research interest is in field of Artificial Intelligence (AI), in particular, the application of AI in Education