KNOWLEDGE-BASED VALIDITY MAINTENANCE FOR PRODUCTION SYSTEMS*

Philip R. Schaefer
Martin Marietta Denver Aerospace
P.O. Box 179, M.S. 0427, Denver, CO 80201

Isil H. Bozma
Yale University, New Haven, CT 06520

Randall D. Beer
Center for Automation and Intelligent Systems Research
Case Western Reserve University, Cleveland, OH 44106

ABSTRACT

In many problem domains, an action may be taken by an expert, which, due to new inferences or a changing domain situation, should be retracted. To this end, an effective problem solver will need to use some kind of validity-maintenance system, so that it can gracefully recover from invalid previous decisions.

Unfortunately, the standard IF/THEN paradigm often used to encode expert behavior does not readily allow the expression and processing of this validity knowledge. We present a new extension to that rule paradigm which can be used to augment production-rule-based systems with validity maintenance capabilities, and demonstrate a straightforward algorithm for its interpretation.

I INTRODUCTION

Through effectiveness for Knowledge Engineering and modularity of Expert Systems construction, Production Rules have become a very popular AI paradigm (Barr and Feigenbaum, 1981). Their advantages as a formalism in Artificial Intelligence are due to several reasons. First, small chunks of knowledge can be incrementally assembled to augment the behavior of the intelligent system. As new knowledge is acquired, the system performance is upgraded incrementally (Winston, 1984). Second, an effective control strategy can handle complex domains using only simple rules. Although the control strategy will need to resolve problems arising from interacting or conflicting rules, achieving this is usually easier than dealing with the corresponding complications that would arise from designing and modifying a single, complex program encoding the same expertise.

*This work was performed at the Center for Automation and Intelligent Systems Research, Case Western Reserve University.

Unfortunately, several problems can arise in systems where a large number of rules may sequentially fire during the problem-solving process:

- -Previous inferences can become inaccurate
- -Old inferences can become inconsistent with the new
- -Intermediate solutions can be non-optimal when compared with new knowledge.

To overcome these problems, a non-monotonic validity maintenance scheme is desirable (Rich, 1983). With such a scheme, the results of rules which have previously fired can be retracted when necessary. In this way, inferences which become inaccurate or inconsistent as a result of new observations or new inferences can be gracefully removed from the system, allowing newer, correct inferences to be made. Additionally, assumptions which were originally made in an effort to achieve a feasible solution can be retracted when contradictions arise or when superior assumptions are found.

Human experts, of course, are proficient at dealing with this kind of validity maintenance in our dynamic everyday world. People seem quite willing to make assumptions or conclusions before complete evidence is available. The important point, however, is that they also are able to correct or reject these inferences as more information becomes available or as the domain situation changes over time. Several systems (called "Truth Maintenance Systems," or TMSs) for introducing similar validity maintenance abilities to computer implementations have been described in the literature. Systems such as Doyle's (1979) require the AI system to maintain a single consistent set of inferences. A newer kind of validity maintenance, an "assumption-based" approach, keeps track of assumption sets supporting the various inferences, and thus allows multiple, possibly contradictory, inference sets to be developed simultaneously (de Kleer, 1984). In each of these systems, a network of reasons or assumptions behind the inferences is constructed, which is manipulated to maintain validity.

However, it is not clear exactly how the validity-maintenance abilities of the human expert can be naturally expressed in terms of such networks. It may be possible for the expert to indicate how the individual working-memory elements in the network are dependent on each other. However, the implementation-level network is at a considerably lower level of representation than the "knowledge level" expert production rules. It would be unsatisfactory to require the expert to think about the domain at both levels.

It is clearly important, therefore, to be able to maintain and process validity knowledge in a Production System in a TMS-like fashion. Just as important, however, is the need to express the human expert's validity knowledge in a natural and straightforward way, to preserve the Knowledge Engineering advantages of the production-rule paradigm.

II EXPRESSING VALIDITY MAINTENANCE KNOWLEDGE IN PRODUCTION RULES

Although a validity-maintenance mechanism is clearly important in many expert systems, it is not immediately apparent how an IF/THEN rule approach can capture the necessary additional knowledge. The usual reading of a rule in a production system is something like

```
IF <this condition is seen>
THEN <make this conclusion>
(Barr and Feigenbaum, 1981,
 Weiss and Kulikowski, 1984)
```

During the reasoning process, if the condition is observed even for an instant, the conclusion is asserted. Although a separate rule might be able to remove the conclusion, the semantics of this rule assumes that it will apply for all time. This kind of rule will unfortunately cause problems if the expert system is reasoning in a world of changing knowledge or data. If the following rule were fired under such an interpretation:

```
IF <the weather appears bad>
THEN <do not have the picnic>
```

the system would not have the knowledge that the conclusion was no longer valid if the weather later became good.

One alternate IF/THEN rule interpretation to handle this kind of problem could be

```
IF <this condition is true>
THEN <this conclusion is true>
```

In that case, when <the weather appears bad> became false, the <do not have the picnic> would become false as well. Such an interpretation would also lead to problems in a dynamic knowledge environment. Consider the effects of the conclusion of the rule

```
IF <the ignition switch is on> and
   <the starter makes no sound>
THEN <there is an electrical problem>.
```

When the ignition switch were turned off to perform some other test, the <electrical problem> conclusion would be forgotten.

Occasionally, knowledge seemingly of a rule-based nature, may defy expression in either of these interpretations. For example, a meeting scheduler might have a rule such as

```
IF <everyone can attend the meeting at time X>
THEN <schedule the meeting at time X>

with the condition
   "this is valid as long as
    conflicts arise for no more
    than 10% of attendees."
```

In this case, the validity condition is not even one of the antecedants of the original decision.

Using either of the above two rule interpretations, it is probably possible to get the desired behavior in the IF/THEN approach by including clever "patches" of several additional rules. From a Knowledge-Engineering perspective, however, this is an unsatisfactory solution. The modularity of the rule-based knowledge representation would be compromised, and rules which are quite straightforward for a human to understand would be stated to the system in a complex, barely comprehensible fashion.

In each case, it is evident that a human expert stating such rules has a clear intention of the validity conditions implied. We propose that the IF/THEN rule paradigm be extended to allow a natural and conceptually "clean" expression of this knowledge. The proposed extension is to use, rather than IF/THEN constructs, an IF/THEN/AS-LONG-AS construct. The interpretation of a rule in this new form is

```
IF <this condition is seen>
THEN <make this conclusion>
AS-LONG-AS <this validity condition
            remains true>.
```

The three above examples could be written as:

```
IF <the weather appears bad>
THEN <do not have the picnic>
AS-LONG-AS <the weather remains bad
            or threatening>

IF <the ignition switch is on> and
   <the starter makes no sound>
THEN <there is an electrical problem>

IF <everyone can attend the meeting
    at time X>
THEN <schedule the meeting at time X>
AS-LONG-AS <conflicts arise for
            <10% of attendees>.
```

With this construct, the Knowledge Engineer states an IF/THEN rule in the usual form, then adds the validity conditions under which the result remains valid. In this way, the reasoning and validity maintenance of the expert system will more closely resemble that as perceived by the expert when stating the rules. There is no longer a need to use elaborate "rule programming" to achieve the desired validity-maintenance behavior.

III PROCESSING IF/THEN/AS-LONG-AS RULES

Processing rules containing AS-LONG-AS parts will, of course, require some additional steps beyond the usual matching and chaining of standard IF/THEN rules. Here, we discuss algorithms that can be used for such processing.

Fortunately, techniques exist today which, although not designed with AS-LONG-AS rules in mind, do provide the dependency processing such rules will require.

Dependency networks, such as found in Truth Maintenance Systems, are one way to internally store the dependence of conclusions on the validity conditions associated with the inference rules. With such storage of dependency information, an efficient rejection of invalidated decisions can take place as soon as any validity condition is violated. Although the full implications of the integration of IF/THEN/AS-LONG-AS rules with a complete TMS, including the associated assumption selection, have not yet been studied, it has been established that at least the retraction portions of TMS mechanisms are effective for these rules. Our validity-maintenance algorithm, most similar to (Doyle, 1979), comprises three steps:

-note the dependency relationships implied by an AS-LONG-AS part when a rule fires to find a value for an element of the working memory

-check for validity of other memory elements when a new value is entered into memory. Invalidate those elements whose AS-LONG-AS conditions are violated

-replace the old dependency information in the system when the value of an element is altered.

The first step can be implemented with the following algorithm:

```
When an data element E is inserted into
memory by rule R,
  1. Examine the AS-LONG-AS part of R
  2. For each working memory element E'
     associated with that AS-LONG-AS
     evaluation
     1. store E as a dependent of E',
        along with R and a list of the
        variable bindings associated
        with R at that time
     2. store E' as A-CAUSE-OF E.
```

Further, if the element had a value previously, the system must check to see if any of its dependents are affected, and invalidate as necessary:

```
1. For each dependent D associated with
   memory element E
   1. Examine the AS-LONG-AS part of the
      rule which depends on D, in the
      context of the variable bindings
      stored, to see if D's validity
      conditions remain true
   2. If it evaluates to "false,"
      invalidate the value of D.
2. For each invalidated dependent of D,
   recursively perform this invalidation
   algorithm until no more invalid working
   memory elements result.
```

Additionally, it is necessary to "clean out" any old dependency relations that the previous value of E had:

```
When a previously-existing element E
is given a new value
  for each A that is A-CAUSE-OF E,
     remove E from the dependencies
     list of A.
```

Using this validity-maintenance algorithm, effective expression and processing of validity knowledge can be performed, as we next demonstrate.

IV AN EXAMPLE OF AS-LONG-AS PROCESSING

To illustrate the effectiveness of rules in the IF/THEN/AS-LONG-AS construct, the following meeting-scheduler system is presented, which makes use of the kinds of validity conditions previously discussed. The following rules show the rules for initially scheduling a meeting, determining when a person has a conflict, and for scheduling a meeting around conflicts:

```
rule-1:
  IF no attendee has a conflict
     at proposed time T
  THEN schedule the meeting at time T
  AS-LONG-AS <= 25% of attendees
             get conflicts

rule-2:
  IF person P has multiple meetings
     scheduled for time T
  THEN person P has a conflict of value T
  AS-LONG-AS those multiple meetings
             remain at time T

rule-3:
  IF meeting requested time = t
     and meeting scheduled time
     cannot = t
  THEN propose meeting for time T+1
```

Figure 1 shows the initial knowledge base, with "forms" to be filled with the meeting and attendee information, just one of many possible knowledge representations.

Suppose that the expert system first considers MEETING-1. It is desired to assign it a scheduled time, so rule-1 and rule-3 are matched as providing scheduling information. In evaluating rule-3, the system tries to schedule the meeting at its requested time, 10:00, and, because of rule-1, succeeds. The "scheduled time" element is therefore filled with the value 10:00. Next, the AS-LONG-AS part of rule-1 is examined. In so doing, the system looks at the list of MEETING-1 attendees and looks for conflicts of each one. Validity-maintenance links are stored to record the dependency of the scheduled time on each of the working memory elements accessed during this AS-LONG-AS evaluation. In this case, links would be stored between the "attendees" element of MEETING-1 and the "scheduled time" element. For the same reason, links would be stored between the MARY, ELLEN, and STAN "conflict" element and the "scheduled time" element. Example dependency links can be seen in Figure 2.

Next, the system considers MEETING-2, and similarly stores 10:00 in its "scheduled-time" element, dependent on the conflicts of JOE, SUE, and BOB, and the "attendees" element of MEETING-2. Additionally, because rule-2 was used to find the possible conflicts of the attendees, all attendees' "conflicts" elements will be stored as dependencies of the respective meeting "scheduled time" elements, due to the rule-2 AS-LONG-AS part.

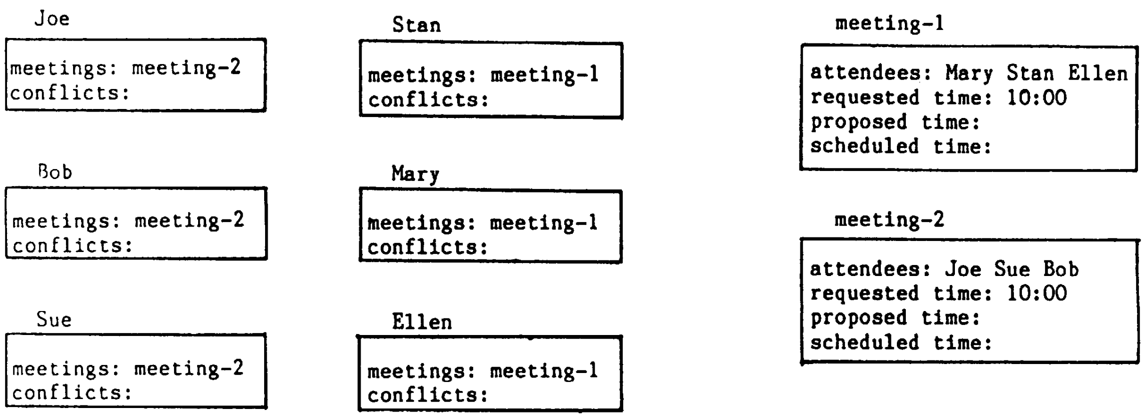

Figure 1. The initial knowledge base for the scheduler example.

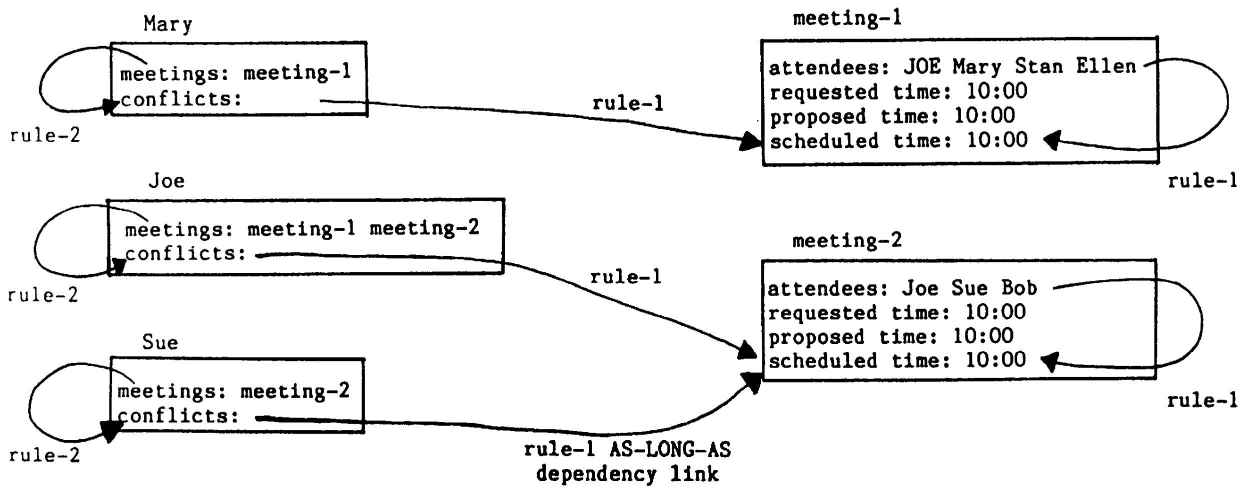

Figure 2. After the initial scheduling, JOE is placed on the meeting-1 attendees list. This causes validity maintenance, which will reschedule meeting-2, as described in the text.

Now, suppose that someone decides that JOE should attend MEETING-1 as well. Correspondingly, MEETING-1 is put on his "meetings" list and JOE is put on the meeting "attendees" list. Figure 2 shows how this new knowledge affects the validity-maintenance network. Validity maintenance processing is invoked whenever a working-memory element upon which other elements depend, is modified. Therefore, because the "scheduled time" of meeting-1 is a dependent of the modified "attendees" entry, its validity conditions, the AS-LONG-AS part of rule-1, are checked. The "conflicts" slots of all the attendees are evaluated, and in the process, JOE receives a "10:00" in his "conflict" element. Despite the conflict, the AS-LONG-AS part of rule-1 remains true for MEETING-1, and its scheduled time remains valid.

However, the MEETING-2 scheduled time is a dependent of the now-modified JOE "conflict" entry, so its validity condition must be checked as well. For MEETING-2, though, 33% of the attendees now have conflicts, violating the AS-LONG-AS part. Therefore, the validity-maintenance mechanism removes the scheduled time from MEETING-2 and its dependent, JOE's conflict. A new value for the scheduled time must be found, and in rescheduling, rule-3 fires, giving MEETING-2 the scheduled time of 11:00. At this point, everyone is happily scheduled without any conflicts.

From this example, it becomes clear that even with simple AS-LONG-AS parts in the rules, quite complex validity maintenance can result. To encode this explicitly without IF/THEN/AS-LONG-AS rules would have required considerably more rule engineering.

V A STUDY CASE FOR THE EXTENDED PRODUCTION RULES

We have implemented a form-filling production-rules-based expert system called DIFF- The Domain Independent Form Filler (Beer, 1986). The knowledge in DIFF is stored in user-defined forms, which contain as their values either knowledge provided to the system a priori or knowledge inferred by the system as it proceeds. The system is goal-driven to fill requested entries of specified forms, using production rules and the other form entry values. Many form-filling tasks, such as a implemented Course Scheduling task, similar to the example above, require validity maintenance on form entries. DIFF uses IF/THEN/AS-LONG-AS rules to accomodate this requirement.

Within the form-filling paradigm of DIFF, it was quite straightforward to implement the validity maintenance algorithm described above. The various form entries in the system correspond to elements of Working Memory. Therefore, each time a form entry is filled, all of the form accesses in the AS-LONG-AS part of the rule are given the new entry as a dependent. A future change in the values of any of these form accesses will cause validity checking of the dependent form.

In addition to the completed course-scheduler example, current work includes using DIFF for cardio-vascular diagnosis and tax-form consulting.

VI CONCLUSION

Validity Maintenance in production systems used in domains of changing knowledge is crucial if the system is to avoid the inaccuracy, inconsistency, and non-optimality problems of the basic IF/THEN formulation. To this end, in addition to the usual knowledge about what domain inferences to make, the knowledge of the context under which these inferences will remain true in the future is necessary. For Knowledge Engineering purposes, the validity knowledge should be expressible in a natural and compact way. The new approach of using rules in the IF/THEN/AS-LONG-AS form meets these requirements. Because of the straightforward algorithms available for its implementation, the IF/THEN/AS-LONG-AS paradigm should provide inroads to more practical rule-based design for complex domains.

ACKNOWLEDGEMENTS

We would like to express our thanks to the Director of CAISR, Professor Yoh-Han Pao, for support of this work, and to Professor Leon Sterling, for his comments on an earlier version of this paper.

REFERENCES

[1] Barr, A. and Feigenbaum, E.A., The Handbook of Artificial Intelligence, Vol. I, 190-199, William Kaufmann, 1981.

[2] Beer, Randall, "Interim DIFF User's Manual," Technical Report TR-101-86, Center for Automation and Intelligent Systems Research, Case Western Reserve University, 1986.

[3] de Kleer, Johan, "Choices without Backtracking," Proceedings of the National Conference on Artificial Intelligence, 1984, 79-85.

[4] Doyle, Jon, "A Truth Maintenance System," Artificial Intelligence, Vol. 12, 231-272, North-Holland, 1979.

[5] Rich, Elaine, "Knowledge Representation Using Other Logics," Artificial Intelligence, McGraw-Hill, New York, 1983.

[6] Hayes-Roth, F., Waterman, D.A., and Lenat, D.B., Building Expert Systems, Addison-Wesley, Reading, MA, 1983.

[7] Weiss, S.M., and Kulikowski, C.A., Designing Expert Systems, Rowman and Allanheld, Totowa, NJ, 1984.

[8] Winston, P.H., "Rule-based Systems for Analysis," in Artificial Intelligence, Addison-Wesley, Chapter 3, 1984.