Defence Science Journal, Vol 43, No 2, April 1993, pp 151-158

Implementing Backtrack in Production Systems: A TMS-based Approach

Jaidev, G. Ravi Prakash

Artificial Intelligence Laboratory, Department of Computer Science and Engineering
Indian Institute of Technology, Madras-600 036

and

N. Parameswaran

School of Computer Science and Engineering
University of New South Wales, P.O. Box 1, Kensington, NSW 2033, Australia

ABSTRACT

OPS5 is one of the most widely used Production System languages. The control strategies provided in OPS-like languages are extremely weak and consequently during problem solving, frequently dead-ends are encountered. We have made an attempt to perform dependency-directed backtracking within the framework of an OPS-like interpreter which employs a Truth Maintenance System for reasoning with past actions. The conditions for backtrack are characterised by the violation of a set of domain-specific constraints and signalled as contradictions in the Dependency Network (D-Net). We have developed a system, called OPS91, which comprises an enhanced match-select-act cycle operating on a D-Net as working memory. This cycle is explained with reference to the D-Net structure and the correlation between the D-Net and Rete-Net operations. The revision algorithm and null conflict set resolution strategy are detailed. The performance of the system is evaluated and broad guidelines on the programming strategy are presented.

PRODUCTION SYSTEMS FOR PROBLEM SOLVING

Problem solving in artificial intelligence (AI) refers to the numerous techniques developed to instil in machines, specific intellectual capacities hitherto performed only by humans. The concept of General Problem Solving started with the advent of the information processing paradigm[1], and has since diversified into many generic frameworks for representation and reasoning. One such framework for general intelligence is the Production System (PS). The PS model has been used to solve a variety of problems[2,3] and these experiments have resulted in many PS languages, OPS5[4] being one of the most widely used ones.

In problem solving using OPS-like languages (as in many problem solving systems), the focus lies in the control of reasoning. Unfortunately, the control strategies provided (such as MEA and LEX) are extremely weak, and suffer from the problem of local maxima unless guaranteed heuristics are available for the problem being solved. This issue is further compounded by the inability of OPS to keep track of the problem solving logic and the history of the working memory (WM) and hence, its inability to explore alternative options during constraint violations and dead-ends.

The advantages of performing a dependency-directed backtrack over utilizing a chronological backtrack have been well established. Dependency-directed backtracking (DDB) is often easily incorporated in backward chaining inference schemes. For instance, in Prolog, variable bindings received from achieved subgoals can be revised in order to satisfy other currently unachievable subgoals[5]. However, in forward chaining systems such as OPS5 (where backtracking is

Received 26 October 1992

particularly necessary), DDB is not directly performable, since data dependencies are not obviously visible.

In the present study, we have made an attempt to perform DDB within the framework of a forward chaining inference scheme by employing a Truth Maintenance System (TMS) for reasoning with past actions. The conditions for backtrack are characterised by the violation of a set of constraints specific to the problem domain and signalled as contradictions in the TMS network. The TMS affords a convenient representation for maintaining a complete history of the actions performed, and consequent changes to working memory, while avoiding multiple copies of the world.

We describe a system called OPS91* that interfaces an OPS-like interpreter with a TMS, and study its performance as a specific problem solver. In Section 2, we present some of the recent work on TMS-based problem solvers and issues involved in selecting a specific truth maintenance mechanism. The modifications necessary in the dependency network due to a common action-belief framework are then discussed. The architecture of OPS91 is detailed and some of the implementation mechanisms are outlined in Section 3. Performance evaluations presented in Sections 4 and 5 show that OPS91 is a better problem solver than OPS5, with some overhead in terms of time and memory.

2. RMS-BASED PROBLEM SOLVING SYSTEMS

2.1 Related Work

Probably the earliest attempt at a common representation for actions and beliefs was the work on Plan Nets[6]. Modelling actions in an Assumption-based TMS (ATMS) was discussed by Morris and Nado[7], while Kulkarni and Parameswaran[8] presented an enhanced RMS with action nodes which had their justifications in preconditions, backtrack and plan decision nodes. The concept of encoding planning events and plan events has been mentioned by Beetz and Lefkowitz[9], and Jaidev and Parameswaran[10] described a Dependency Net (D-Net) representation for coding the logic of a subgoaling planner. The REDUX architecture[11] discusses the use of TMS for maintaining the validity of the reason for context choices and rejections, since problem solving is often based on switching contexts based on heuristic preferences.

*Called so because it was first developed and described in 1991

2.2 Choice of Truth Maintenance System

There are basically two kinds of Truth Maintenance Systems: the Justification-based TMS [hereon simply called Reason Maintenance System (RMS)] and the Assumption-based TMS (ATMS). The issue that next surfaces is obviously the choice of a system for reason maintenance. An RMS is a single context mechanism that is particularly appealing for synthesis tasks. However, there is always a likelihood that contexts may have to be switched for several reasons (such as upon discovering constraint violations) and the RMS does this by a costly and unnatural process. ATMS, on the other hand provides a very efficient way of handling multiple contexts, corresponding to exploring alternatives in parallel. But these systems become very cumbersome when the number of contexts is very large and a single thread of reasoning is required (as in the case of planning).

2.3 ATMS-based Production System Methodologies

Morgue and Chehire[12] describe the efficiency of PSs coupled with an ATMS. Two approaches have been proposed, based on the phase (of the match-select-act cycle) into which the ATMS has been integrated. Morgue emphasizes the inherent problem of ATMS-based systems in the form of control of contexts, and proposes a few solutions with respect to the PS scenario. However, an important issue left unaddressed is that of representing and reasoning with actions[7] in this framework. This issue particularly manifests itself in a problem solving activity such as planning, where the nature of revocable actions brings about obvious non-monotonicity at the level of heuristic search. Freitag and Reinfrank[13] have also described an (A)TMS-based problem solver that makes a restricted use of variables in the productions. Freitag's framework, just as Morgue's, reasons only with beliefs and not with actions.

3. OPS91: AN RMS-BASED PRODUCTION SYSTEM

We propose[14] an enhanced PS architecture based on an RMS mechanism, keeping in view single-context problem solvers (specially planners). An interpreter which accepts an OPS-like syntax is proposed. Concrete problem solver actions are modelled as productions,

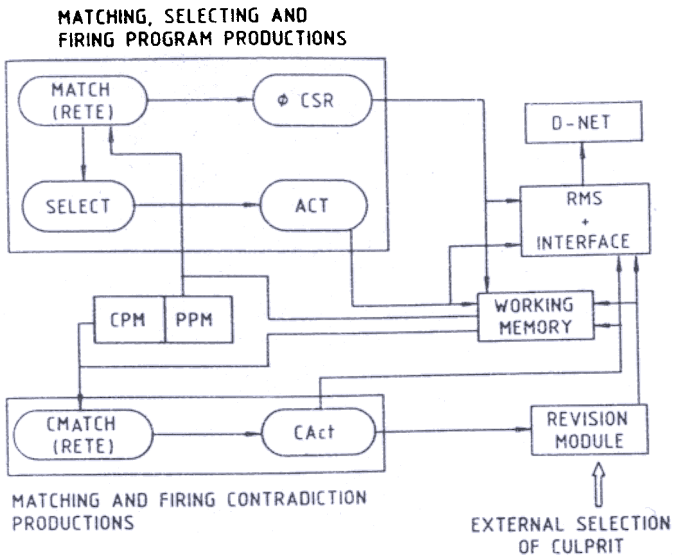


Figure 1. OPS91 architecture.

and the data dependencies are recorded on a dependency network. In the event of a contradiction, an unlabelling and relabelling procedure is utilized to redirect search. A chronological backtrack mechanism forms a backup in the event of a null conflict set situation and provides completeness to the algorithm.

The main components of the OPS91 architecture (Fig. 1) are the OPS91-Interpreter operating under an enhanced match-select-act cycle, the Partitioned Production Memory, the OPS91 Working Memory, and the RMS operating on a D-Net.

3.1 Partitioned Production Memory

A constrained search by the OPS system can be performed by means of explicit representation of domain constraints as productions, and an interleaved contradiction resolution cycle. For this reason, the Production Memory is partitioned into two distinct memory spaces: the Program Production Memory (PPM) and the Contradiction Production Memory (CPM). The PPM is similar to the conventional OPS5 Production Memory.

CPM is also a set of productions having the same structure as PPM. These productions when fired do not cause any change in the state of the world. They, however, are meant to signal new contradictions introduced as a result of the previous ACT phase (actions performed in the last cycle). The LHS of these productions are a collection of condition elements, as in the case of the PPM. The RHS differs in that it is constrained to be an action of one type. This action makes a special working memory element called contradiction and invokes the RMS Interface Module to add a contradiction node in the D-Net.

3.2 OPS91 Working Memory

The structure and functions of the working memory remain unmodified over the original OPS5 system, except for the special element contradiction. Creation of this element triggers the contradiction resolution procedure, which then attempts a minimal revision in order to recover to a consistent state. The normal OPS91 cycle can proceed only from a consistent state.

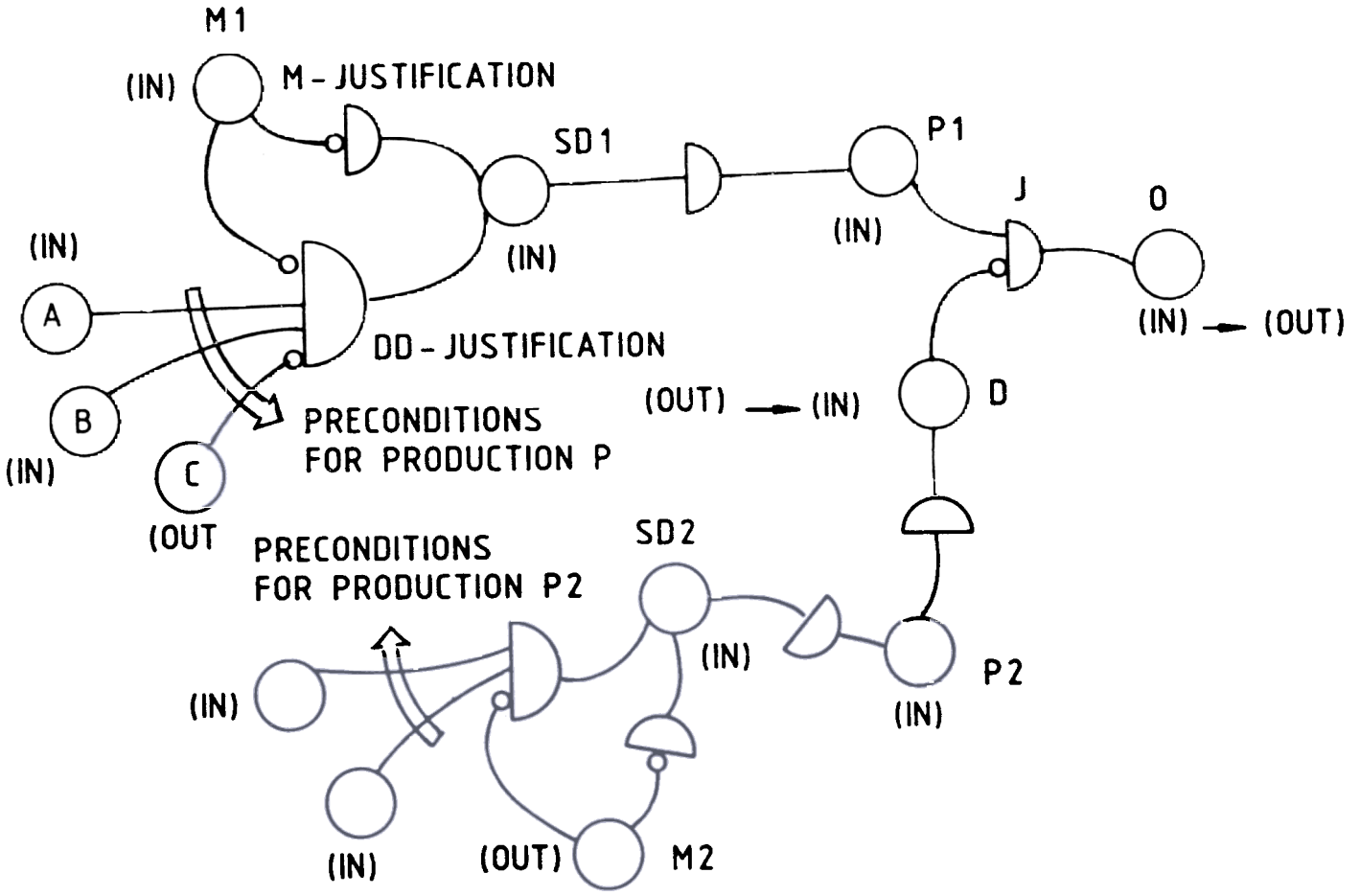
The match on the working memory elements is done using the Rete-Net[15]. When a node is labelled IN (OUT) in the D-Net, it is passed as a token with make (remove) status to the Rete-Net. Thus, the current state (and the history) is stored in the D-Net, and the alpha and beta memories of the Rete-Net point to corresponding object nodes of the D-Net.

It may be noted that OPS matches negated preconditions by checking for the absence of satisfying elements in the working memory, while the RMS models these as OUT labelled nodes in the D-Net. While backtracking it could occur that any of these nodes could be relabelled IN, preventing the previously selected instantiation from being matched (since a negated condition element is present in the WM). It thus becomes imperative to invalidate the data dependency for this instantiation. Consequently, all previously created object nodes* that are currently OUT and are capable of satisfying negated preconditions have to be matched and need to feature as nonmonotonic antecedents for the selection decision node. These nodes appear in the Rete Memory as additional alpha-memory and constitute the overhead incurred in order to perform automatic verification of data dependencies in the event of a revision.

3.3 RMS Module

The RMS module comprises the truth propagation and revision algorithms operating on the underlying dependency network. This dependency network is an enhanced D-Net (over conventional belief networks)[10], because we provide justifications to maintain status assignment on the productions fired and/or revoked.

*There are many nodes for the same object, since OPS creates objects with new time tags each time a modification is made.



M1, M2: Maintenance Nodes SD1, SD2: Selection Decision Nodes
P1, P2: Production Nodes D: Deleter Node
O: Object Node that is added (labelled IN) by Production P1 and and deleted (labelled OUT) by later Production P2

Note: Object Nodes A and B represent objects that matched positive condition elements, while Object Node C represents an object (deleted in the past) that can satisfy the negative condition element, on the LHS of the production represented by Production Node P1.

Figure 2. Representation of the addition/deletion of an object by productions in the D-Net.

The D-Net consists of five categories of nodes: object nodes, production nodes, selection decision nodes, maintenance nodes and contradiction nodes. We briefly describe each of them below and typical representations can be seen in the scenario in Fig. 2.

Object nodes correspond to working memory elements created (these being labelled IN) and deleted (being labelled OUT) by the RHS actions. All object nodes are named by the time tag corresponding to the working memory element they represent. A deleter nodee is used to label an object node OUT (corresponding to deleting an object from the working memory). The deleter node (labelled OUT by default) forms a nonmonotonic antecedent on any supporting justification for an object node. During a delete operation, the deleter node is given a valid justification, thus labelling the corresponding object node OUT. The procedure of adding and deleting an object is shown in Fig. 2.

Production nodes refer to instantiations of productions that were fired anytime in the past. These are supported by a single justification with a selection decision node as monotonic antecedent. Selection Decision nodes signify a decision to select an instantiation from the conflict set. These receive support from two kinds of justifications: a Data Dependency Justification (DD-Justification) and a Maintenance Justification (M-Justification), as shown in Fig. 2. The DD-Justification has the object nodes matching the condition elements, as monotonic or nonmonotonic antecedents (depending on whether the condition elements are positive or negative, respectively) and a

maintenance node (described below) as a nonmonotonic antecedent. The M-Justification has the Maintenance Node as the sole nonmonotonic antecedent, and is used to maintain the status of the Selection Decision node even after its antecedents do not hold. This solution to Nell's problem has been discussed in detail by Jaidev[16]. Contradiction nodes are indicative of an inconsistent state and are introduced when the CAct phase (described below) creates contradiction elements in the Working Memory.

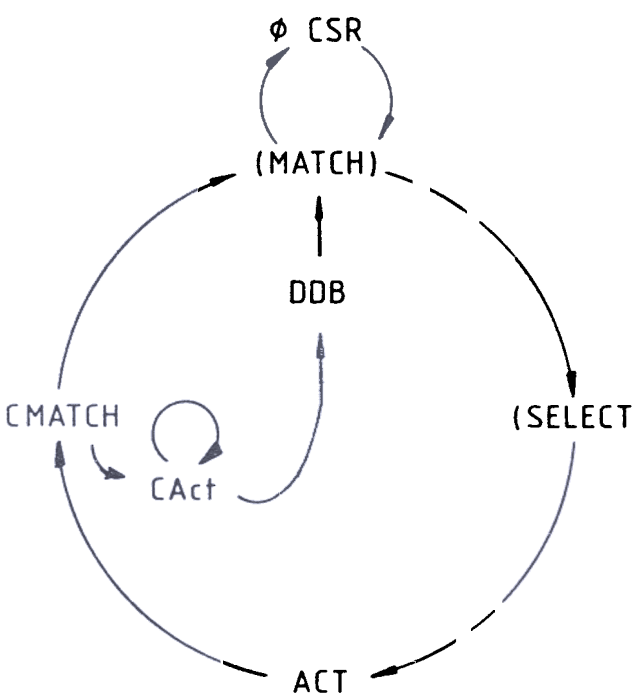
3.4 OPS91 Interpreter

EMSA Cycle

Based on the partitioned Production Memory structure, the Match-Select-Act (MSA) cycle of OPS5 is modified into an Enhanced Match-Select-Act (EMSA) cycle. The EMSA cycle schematic is shown in Fig. 3. As is clearly discernible, three possible cycles are present. The normal OPS91 cycle is one in which no contradiction productions are matched. A summary of the other two cycles is given below.

3.4.2 Domain Contradiction Resolution

The CMatch phase functionally involves matching the working memory elements with the LHS of the productions comprising the CPM. This yields a set of productions whose RHS indicates potential contradictions. The CAct phase fires all the productions output by CMatch. This creates new contradiction elements in the working memory and the corresponding IN labelled contradiction nodes in the D-Net (through the RMS interface). Following this, the DDB procedures are invoked and the contradiction is resolved by making the elective (the only elective being the maintenance node) IN. The complete revision procedure that follows DDB is given below (featuring in parentheses are comments).



Represents the phases of the original OPS5 MSA cycle. All others pertain to the OPS91 EMSA cycle

Figure 3. The OPS91 EMSA cycle.

It may be noted that a backtrack in the current scenario really involves a transition from one point in the state space to an entirely unexplored point in the state space, and hence the new state of the Rete-Net has also to be restored. We do this by passing all the positive tokens in the revised state and the negative tokens with time tags less than the largest time tag in the revised state to the Rete-Net, which then yields the conflict set. Importantly, for negated condition elements, when a token with a remove status appears, the matching token is deleted, but the current token is stored with a remove status.

procedure revise (elective);
a1. *Make–in* (elective, CRJ);
{Make the maintenance node IN by giving it a contradiction resolution justification - CRJ}
a2. *RMS–propagate* (D-Net);
{Propagate the effects of making the elective IN}
a3. *for* all consequent productions *do*
{Make all consequent productions OUT so that the current state of the world (i.e. at the retracted production) is restored}
a3.1 *Make–in* (maintenance–node (production), CRJ)
{The function "maintenance–node" returns the maintenance node for a given production node. Making the Maintenance node IN, labels the corresponding production node OUT}
a3.2 *RMS-propagate* (D-Net);
{Propagate the effects of retracting the production}
a4. *for* all consequent productions in sequence *do*
a4.1 *Remove* (maintenance–node (production), CRJ)
{Remove the contradiction resolution justification for the production so that the validity of the

data dependency can be checked}
a4.2 *if* the DD-Justification for the production is invalid *then*
{If the data dependency does not hold now}
a4.2.1 Make the corresponding maintenance node IN by giving it the same contradiction resolution justification;
a4.2.2 *RMS-propagate* (D-Net);

3.4.3 Null Conflict Set Resolution

Importantly, we consider a null conflict set also as a contradiction, which is resolved by the single loop ∅CSR cycle. The resolution of this form of contradiction is currently done by a chronological backtracking procedure. However, we can add heuristics to the processing of this contradiction, which is backed up by a chronological backtrack in case the former fails. Both the domain contradiction resolution procedures and the null conflict set resolution procedures are costly procedures and rely upon the productions of PPM and CPM being correct and complete. If the program is not correct or complete, OPS91 would continue to perform a large amount of unnecessary time and memory intensive work. Ensuring that PPM and CPM are correct and complete is an important issue and some partial solutions to this problem, have been offered by Ravi Prakash et al[17].

4. PERFORMANCE

We present portions of the trace of a sample session. The problem scenario comprises two rooms, R1 and R2, connected by door D and an agent in room R2 initially. The goal is to paint all the walls of the rooms R1 and R2 and the door D, as well as have the door closed.

The problem specifies that if a door is painted, it cannot be opened or closed. We characterize the dead-ends that can result out of this, in the form of the following constraint production:

```
(p trapped1
    (agent   in <r1>)
    (room    name <r2> < > <r1> ↑ door <d>
    (Wall    painted false ↑ room <r2>)
    (door ↑ name <d> ↑ open false ↑ painted true)
     > (contradiction))
```

OPS91 has been coded in C on a SUN III/60. The data obtained from a run of the OPS91 interpreter for the problem above are given below. (The significance of some of these is discussed in the next section).

Problem Description Specifications		
No. of Object Classes		4
No. of Program Productions		10
No. of Constraint Productions		2
D-Net Statistics		
No. of Nodes at start	:	21
No. of Nodes at completion	:	85
No. of WMEs (deleted + added)	:	21 (11 + 10)
Rete Net Statistics		
No. of 1-input nodes		53
No. of 2-input nodes		17
Av time spent on 1-input nodes		11 ms
Av time spent on 2-input nodes		4 ms
Maximum Time/EMSA cycle (nth cycle)		1400 ms (10)
Minimum Time/EMSA cycle (nth cycle)	:	405 ms (1)
Average Time/EMSA cycle (n cycles)		816.4 ms (12)
Av time/backtrack (no of backtracks)		133 ms (2)
Max ratio of current EMSA cycle time to the previous EMSA cycle time		1.4242
Min ratio of current EMSA cycle time to the previous EMSA cycle time		0.6097
At completion Left Memory size		286
At completion Right Memory size		6
Additional Right Memory size		9
Other		
Worst case no of chronological backtracks		7
Worst case no of DDB		2
Computed time saved on worst case DDB over worst case chronological backtracks		5714.8 ms

5. DISCUSSION

5.1 Results

OPS91 affords a convenient mechanism for programming domain constraints and provides a backtrack mechanism within the interpreter cycle itself.

This leads to a simpler code and reduction in the search due to pruning of non-solution parts of the search space and intelligent revision. One can observe in the previous example that the worst case number of chronological backtracks is 7, while a similar figure for dependency directed backtracks is only 2. An additional constraint that prevents an open door from being painted (since the goal of closing the door can then never be achieved) can further reduce the number of DDBs to only 1.

We found that the additional bookkeeping done in the form of increased right memory is quite low, if the number of modifications to an object are few. As the number of modifications to an object increases, the overhead increases correspondingly. The time spent on 2-input nodes versus the time spent on 1-input nodes varies widely in accordance with the examples (over a number of problems studied, we have observed ratios of 0.2 to 7, the ratio being 0.364 in the above example). The time/EMSA cycle does not vary much over two consecutive cycles, the maximum ratio being 1.4 and the minimum ratio 0.6. The average time taken for a revision of the production history was only 133 ms which is much lower than the average time for a complete EMSA cycle, which is 816 ms. The D-Net growth over a total of 12 cycles shows a change from 21 nodes to only 85 nodes, implying that the increase in D-Net size does not constitute a significant memory overhead.

5.2 Associated Issues

One of the important issues resulting from the method of backtracking adopted in OPS91 is that the justifications for further instantiations cache the preconditions that were satisfied due to which the productions were matched, but do not represent the conditions under which they were selected. That is, OPS91 functions under the assumption that if $P_1, P_2, \ldots, P_i, \ldots, P_k$ is the sequence of productions at the time of backtrack and if P_i is backtracked upon, then all $P_i \in \{P_{i+1}, \ldots, P_k\}$ are selected if their preconditions are satisfied. This assumption is discovered to be weak. However, OPS91 is in any case guaranteed to find a solution (if it exists) because it is backed up by the chronological backtrack of the ∅CSR cycle. The system would be complete if the heuristics utilized in selecting a production were also captured in the D-Net. This would make it possible to validate the new sequence of productions that result from the revision procedure.

The representation of objects as nodes in the D-Net, instead of assertions, appears to model the M-S-A cycle of OPS systems naturally, but leads to a large accumulation of nodes on justifications. For instance, all objects that are capable of satisfying negated preconditions feature on DD-Justifications (as explained in Section 3.3). However, our approach may not be resource intensive if the number of modifications to a particular object is small.

6. CONCLUSION

We had started by noting that there are two approaches to PS using Truth Maintenance: RMS-based and ATMS-based. We can reasonably conclude by stating that the choice of ATMS or an RMS for integrating with a production system is decided essentially by the application on hand (and it is perhaps useful to explore the notion of a Generic or Hybrid TMS) but the utility of either approach to developing an enhanced PS architecture that can reason with revocable actions by performing intelligent backtracking is unquestionably settled.

To utilize the power of the enhanced PS architecture proposed, the conventional problem specifications (such as of OPS5) must be transformed to constraint-based problem specifications, to exploit the facility of specifying domain constraints available in OPS91. An important guideline for performing such a shift in programming strategy is to analyse the problem with a view to determining what domain situations would never lead to a solution state, generalizing patterns of such situations and representing these patterns as domain constraints. OPS91 could then become an extremely efficient programming methodology for AI-based problem solving.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the help rendered by P Viswanath, KG Venkatasubramanian, C R Satyan, T Arul Seelan and Dhrithiman Banerjee in the implementation of OPS91.

REFERENCES

Newell, A. & Simon, H.A. Human problem solving. Prentice-Hall, Englewood Chiffs, 1972.

2 Buchanan, B.G. & Shortliffe, E.H. Rule-based expert systems. Addison-Wesley Pub. Co., 1984.

McDermott, J. R1: A rule-based configurer of computer systems. *AI Journal*, 1982, **19.**

Forgy, C.L., OPS5 users manual. Carnegie-Mellon Univ., CMU-CS-81-135, July 1981.

5 Kumar, Vipin & Lin, Yow-Jian. A framework for intelligent backtracking in logic programs. *In* Proceedings of 6th Conference on Foundations of Software Technology and Theoretical Computer Science, New Delhi, 1986. pp. 108-23.

6. Drummond, M.E., A representation of action and belief for automatic planning systems. *In* Proceedings of Workshop on Reasoning about Actions and Plans, Oregon, 1986.

7. Morris, P.H. & Nado, R.A. Representing actions with an Assumption-based Truth Maintenance System. *In* Proceedings of AAAI-86, 1986.

8. Kulkarni, D. H. & Parameswaran, N. Truth Maintenance System for action representation. *In* Proceedings of 4th Portuguese Conference on AI, EPIA-89. Springer Verlag, 1989.

9. Beetz, M. & Lefkowitz, S.L. Reasoning about justified events: A unified treatment of temporal projection, planning rationale and domain constraints (an extended abstract). MEMO-P-89-21, TA TRIUMPH-ADLER AG, Germany, 1989.

10. Jaidev & Parameswaran, N. PLANET: A tool for representing & generating plans in an RMS Framework. *In* Proceedings of 3rd IEEE International Conference on Tools for Artificial Intelligence, TAI'91, San Jose, 1991.

11 Petrie, Jr., C.J. Context maintenance. MCC Tech. Report ACT-RA-364-90, 1991.

12. Morgue, G. & Chehire, T. Efficiency of production systems when coupled with an assumption-based truth maintenance system. *In* Proceedings of AAAI-91, 1991.

13. Freitag, H. & Reinfrank, M. A non-monotonic deduction system based on (A)TMS. *In* Proceedings of ECAI-88, 1988. pp. 601-06.

14. Jaidev, Ravi Prakash, G., Parameswaran, N. & Mahabala, H.N. OPS91: An RMS-based production system model for problem solving. *In* Proceedings of IJCAI-91 Workshop (W.25) on Advances in Interfacing Production Systems with the Real World, Sydney, 1991.

15. Forgy, C.L. Rete: A fast algorithm for the many pattern/many object pattern match problem. *AI Journal*, 1982, **19**, 17-37.

16. Jaidev. A reason maintenance framework for planning. Dept. of Computer Science and Engineering, Indian Institute of Technology, Madras, 1991. MS Thesis.

17 Ravi Prakash, G.; Subrahmanian, E. & Mahabala, H.N. A methodology for systematic verification of OPS5-based AI applications. *In* Proceedings of IJCAI-91, Sydney, 1991.