Efficiency of Production Systems when Coupled with an Assumption based Truth Maintenance System.

Geneviève Morgue and Thomas Chehire

Thomson-CSF/RCC
160 Boulevard de Valmy, BP 82
92704 Colombes Cedex, France
gmorgue@eurokom.ie

Abstract

Expert systems in complex domains require rich knowledge representation formalisms and problem solving paradigms. A typical framework may involve a blackboard architecture and a Reason Maintenance System (RMS) to guarantee the consistency of the links between the blackboard nodes. However, in order to satisfy computational feasibility and become operational, the resulting expert system must often be rewritten using less expressive tools.

We propose an architecture integrating efficiently an OPS-like inference engine and an Assumption based Truth Maintenance System (ATMS). These paradigms have been separately investigated and extended. Roles distribution between an ATMS and an inference engine integrated in a single framework is one of the major issues to obtain good overall performance.

Two architectures will be studied : loose coupling, where the ATMS and the inference engine are clearly separated, and tight coupling where the ATMS is intimately integrated with the match phase of a RETE-based inference engine. The advantages and drawbacks of both solutions are described in details. Finally, future work is discussed.

Introduction

Expert systems in complex domains require rich knowledge representation formalisms and problem solving paradigms. Commercially available expert system shells provide some compromise between expressiveness and tractability.

A forward chaining engine with an OPS-like rule language is one of the key components of such shells. Its operation involves a match-select-act cycle:

1. **Match** : The condition part of each rule is compared to the content of the fact base (or working memory). If a set of facts conjointly satisfy all the conditions, the rule is said to be instantiated. One or many rule instantiations may thus be found, and are queued in a list of executable operators, called the conflict set or the agenda.
2. **Select** : One or more rule instantiations are selected from the agenda for future execution of their action part. Selection is done according to some user defined conflict resolution strategy. Predefined strategies usually include FIFO, LIFO, highest priority, and more.
3. **Act** : The right-hand-side actions of the selected rule, or rules, are executed. These actions may modify the fact base, which will possibly instantiate new rules.

This work has been supported in part by the DRET (French DARPA) under grant number 89/568.

Having the possibility to retract facts from the working memory is necessary in many applications. When allowing this, one should be aware that the conclusions derived from the removed facts are not necessarily valid anymore. And when there are contradictions in the fact base, the system may not be able to pursue its reasoning process. To avoid handling these problems manually, Reason Maintenance Systems (RMS) have been developed.

Expert systems using a RMS store justifications : a justification is a link between a fact created on the right-hand-side of a rule and the facts which instantiated this rule. Let us illustrate this through an example in an OPS-like syntax :

Rule base: (Rule birds_fly (Bird ?x) → (assert (fly ?x)))

Fact base: (Bird Tweety)

In this example, the justification
(Bird Tweety) → (Fly Tweety) will be created.

When retracting a fact, the system follows the links established by the justifications, to retract not only the desired fact, but also all the facts it enabled to derive.

Among the different RMSs, the ATMS (Assumption based Truth Maintenance System) became very popular in the last few years. ATMSs are a convenient way of exploring many choices in parallel when solving a problem.

With an ATMS, the user does not need to program the expansion of the search space, as he would usually have to in a framework with control primitives and a backtracking mechanism.

Inference engines and ATMSs have been separately investigated and extended. Their integration and interfacing in a single framework implies many design choices. Roles distribution between the ATMS and the inference engine is one of the major issues.

In this article, we propose an architecture enabling to efficiently integrate an OPS-like inference engine with blackboard-like control, and a reason maintenance system. Different architectures will be studied : loose coupling, where the ATMS and the inference engine are clearly separated, and tight coupling where the ATMS is intimately integrated within the match phase of the inference engine. The advantages and drawbacks of both solutions are described in details.

ATMS background

For a better understanding of the remaining of this article, basic concepts of an Assumption Based Truth Maintenance System are listed.

ATMSs make the distinction between **assumptions** and other data (or **facts**). Assumptions are data which are presumed to be true, unless there is evidence of the contrary. Other data are primitive data always true, or that can be derived from other data or assumptions. The ATMS records such dependencies through **justifications.** It is then in charge of determining which combinations of choices (assumptions) are consistent, and which conclusions they enable to draw.

To achieve this, each datum is stamped with a **label** consisting of the list of **environments** (i.e. sets of assumptions) under which it holds. When a new justification for a datum is provided, its label is updated with the label of the left-hand-side of the justification (i.e. list of environments under which all facts or hypotheses supporting the datum through this justification, are simultaneously true).

An environment is inconsistent if it enables to derive a special datum representing the contradiction (usually noted ⊥). It is then called a **nogood.** When such an environment is discovered, it has to be removed from all the labels.

The **context** of a consistent environment is the set of facts that can be derived from the assumptions of that environment. A problem with many possible solutions will thus generate many contexts. The main advantage of an ATMS is that all solutions are developed in parallel, and maximum work is shared between solutions.

However, nogoods handling and labels updating are costly operations and their efficient implementation is a key point in the successful use of ATMSs in real world problems.

Loose coupling of ATMS and inference engine

The first and simplest way of combining an OPS-like inference engine and an ATMS is to modify the select and act steps of the inference engine cycle. The match step remains unchanged.

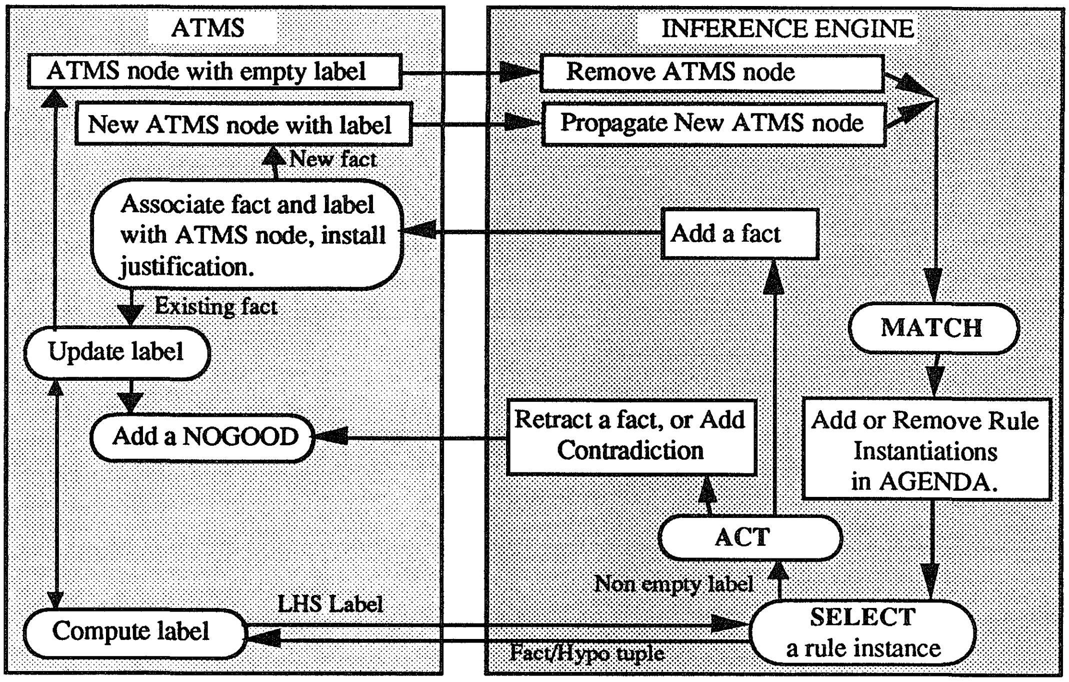

Figure 1 : Loose Coupling of an ATMS and an inference engine.

The ATMS is responsible of all the truth maintenance computations (updating labels, handling contradictions ..). The inference engine transmits facts, assumptions and justifications to the ATMS.

Justifications are created when a rule is fired, maintaining the dependency of the fact (or assumption) created by the action part over the facts (or assumptions) that instantiated the left-hand-side of the rule. This overhead occurs in the act step of the inference engine and is not CPU intensive, unless the justification is installed on an already existing fact, since the ATMS has to update its label, which may cause a chain of label modifications of other connected facts.

An important situation occurs when the label of the justifications's left-hand-side is empty. In this case, updating labels does not modify the labels of existing ATMS node, and creating an ATMS node with an empty label is useless, since the associated datum does not hold in any context. Such justifications bring no new information, and firing a rule whose left-hand-side has been matched by a contradictory fact tuple can thus be prevented.

Thus, in the loose coupling approach, we slightly modify the select phase of the inference engine. When a rule instantiation is selected from the agenda the ATMS is called to compute the label of the facts or hypotheses that matched its left-hand-side. If the computed label is empty, the selection is invalidated and the selection phase has to try another rule instantiation.

Other justifications of interest are those that support the ⊥ datum, since they may create new nogoods. Such justifications are created using special rules called contradiction-rules, the sole implicit action of which is to derive ⊥.

Firing contradiction-rules will possibly prevent some other rule firing, by augmenting the nogoods.

Moreover, if the contradiction rules are fired too late, expensive label updating may occur due to justifications introduced by rules that would have been otherwise prevented from firing. One of the crucial problems in interfacing an OPS-like inference engine and an ATMS is thus to discover the contradictions, thereby creating nogoods, before firing any rule instantiated by a tuple, the label of which contains a superset of such nogoods.

Contradiction rules are thus given a special priority, and the select phase will always pick them first in the agenda. This loose coupling approach has some advantages, but can quickly become intractable in a combinatorial application.

Let us illustrate this with the well known 4_Queens problem :

```
(rule initialize
- >
  (for ?i from 1 to 4 do
       (assume (queen ?i ?j)))))
```

```
(contradiction queen_attack
  ?q1:(queen ?x ?y)
  ?q2:(queen ?u ?v)
  (test (and  (not (eq ?q1 ?q2))
              (or  (and (= ?x ?u)
                          (not (= ?y ?v)))
                      (and (= ?y ?v)
                          (not (= ?x ?u)))
              (=  (abs (- ?y ?v))
                  (abs (- ?x ?u))))))
```

```
(rule find_solution
(queen 1 ?i)
(queen 2 ?j)
    (queen 3 ?k)
    (queen 4 ?l)
 ->(assert
 (solution ?i ?j ?k ?l)))
```

Rule *find_solution* will be instantiated by the following fact tuples:

```
(queen 1 1) (queen 2 1) (queen 3 1) (queen 4 1)
(queen 1 1) (queen 2 1) (queen 3 1) (queen 4 2)
(queen 1 1) (queen 2 1) (queen 3 1) (queen 4 3)
(queen 1 1) (queen 2 1) (queen 3 1) (queen 4 4)
etc...
```

One clearly sees that 256 instantiations of the *find_solution* rule are queued in the agenda whereas only 2 will get fired to find the 2 solutions to the 4_queens problem. The match process of the inference engine has done useless work, because the overall label of a tuple instantiating the *find_solution* rule is computed only after the rule is completely instantiated, and not while the inference engine is trying to match its condition part with facts in the fact base.

Another important issue is that the ATMS will recompute the label of the fact tuple {(queen 1 1) (queen 2 1)} 4 times, to compute the label of the tuples which created the first 4 instantiations of the *find_solution* rule, listed above.

The problem with this loose coupling is therefore that a lot of work is done either repeatably by the ATMS or uselessly by the match step of the inference engine.

We clearly need to integrate the ATMS label computation with the match step of the inference engine, and store intermediate label computation in order to share label computations between many rule instantiations. The RETE algorithm has been chosen for such an approach, since it is a state saving and node sharing algorithm and one of the most efficient ones for OPS-like inference engines.

Tight coupling between an ATMS and a RETE network

Tight coupling of the ATMS and the inference engine involves some modifications to the RETE algorithms. We presume some familiarity with production systems and the RETE algorithm. We invite the reader to refer to the book "Programming in OPS 5" (Brow.nston et al. 1985), to articles on the RETE (Forgy 1982) (Scales 86), (Schorr et al..1986) and (Chehire 1990).

The basic idea is to make the ATMS intervene earlier in the inference engine cycle : we will thus modify the match step instead of the selection step. This method will enable to discover the contradictions much earlier, and thus to shortcut a great amount of computations (join operations) in the RETE network. Label computations are stored in the RETE memory nodes, and possibly shared among different rule instantiations.

Creating partial justifications

Let us fire the rules in the following example :

```
Rules :   (Rule R2
              (employee ?name ?department)
              (location ?department  ?floor)
              (test (≤ ?floor 2))
           → (assert (takes-stairs ?name)))
          (Rule R3
              (employee ?name ?department)
              (location ?department  ?floor)
              (test (≤ ?floor 2))
              (age ?name ?age & > 50)
           → (assert (warn ?name take-lift)))
Facts : (employee Betty research)  (location research 1)
        (age Betty 51)
```

In the loose coupling approach, justifications provided by the inference engine consist in two completely instantiated rules :

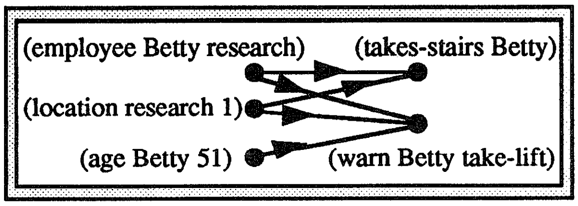

Figure 2 : ATMS nodes and links.

The label of the fact *(takes-stairs Betty)* is computed from the labels of the *(employee Betty research)*, and *(location research 1)* facts. The label of the fact *(warn Betty take-lift)* is then computed from the labels of the *(employee Betty research)*, *(location research 1)*, and *(age Betty 51)* facts.
The computation of the first justification could be used for the second one.

To insure the sharing of labels computations and thus improve global performance, we introduce **partial justifications** and new ATMS nodes.

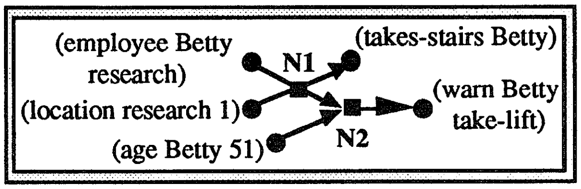

Figure 3 : Partial justifications.

We replace the previous two total justifications by the following four partial justifications :

(Employee Betty research) ∧ (location research 1) → N1
N1 ∧ (age Betty 51) → N2
N2 → (warn Betty take-lift)
N1 → (takes-stairs Betty)

The label of N1 corresponds to the label of the tuple {(employee Betty research) (location research 1)}. It will be used to compute the label of *(takes-stairs Betty)*, and the label of N2, which in turn will be used to compute the label of *(warn Betty take-lift)*. Some of the computation is thus factorized.

Furthermore, this structure is very easily matched on the RETE architecture : it can be built incrementally while propagating the fact tuples in the RETE network. The RETE memory nodes now contain not only the tuples instantiating the joined patterns, but also the **intermediate label** of this tuple (label of the corresponding partial justification).
The only modification to the basic RETE propagation algorithm is to compute the label of each created fact tuple, and if this label is empty, the tuple is discarded and not transmitted to successor nodes in the network.

Labels recordings in the memory nodes significantly reduces labels re-computations for a single rule. Moreover, labels recomputations for different rules can be reduced when carefully coding the rules, thanks to the node sharing algorithm of the RETE network.

Another important issue for global performance of the RETE network, is that the memory nodes can have a significantly smaller size when label computations are included in the network, since inconsistent fact tuples are discarded early in the network.

Optimizing the update label algorithm, and nogoods

Nevertheless, a problem arises when a new nogood is discovered. If this environment has already been used in some intermediate labels, all memory nodes where it appears have to be updated. A similar problem arises when a new justification is installed on an existing fact. Label updating is a costly operation in an ATMS, and is made even worse with tight coupling, since we have added new ATMS nodes that are stored in the RETE memory nodes. If a fact (or a fact tuple) becomes inconsistent, it has to be removed from the RETE memory nodes, together with all facts or tuples connected to it.

In order to optimize label updating, and possible fact or tuple removal from the RETE memory nodes, we do more bookkeeping than in the standard RETE operations :

- each fact records the α-memory nodes where it is stored,
- each fact tuple records the β-memory node where it was created,

- each environment records all facts and fact tuples, in the label of which it appears,
- each fact and fact tuple records the links through partial justifications to other facts and fact tuples.

When a nogood is discovered, the label of each fact or fact tuple recorded in this environment has to be updated, propagation of label updating follows the partial justifications links. When the label of a fact or a fact tuple becomes empty, it has to be removed from the RETE memory nodes where it appears.

The important point is that this retract operation is made very efficient.

- The remove-fact procedure : if a fact has to be removed, we just add the recorded α-memory nodes in a list of modified RETE nodes. We then follow the partial justification links to remove the fact tuples directly connected to this fact, calling the remove-tuple procedure. No updating of the RETE nodes has yet taken place.
- The remove-tuple procedure : if a fact tuple has to be removed, we just add the recorded β-memory node in the list of modified RETE nodes. We then follow the partial justification links to remove the tuples directly connected to this fact. No updating of the RETE nodes is done.

When the previous two procedures are done, we iterate on the list of modified RETE nodes to simply remove the marked facts or fact tuples.

The retract operation is here very different from the add operation, which is not the case in the standard RETE technique. The memory nodes that really need to be updated, and only those, are accessed. Moreover, they are accessed only once for a single retract operation.

However, label updating and nogood handling remain costly operations and great attention should be paid not to uselessly transmit tuples which will later be discovered inconsistent, and will thus have to be removed from the memory nodes.

This problem arises for example when a fact tuple instantiates both a contradiction rule and other rules. As soon as the contradiction is fired, all such instantiations will be removed from the agenda, and intermediate fact tuples removed from the RETE memory nodes.

In order to avoid this, the user needs to tune the propagation of fact tuples in the RETE network. The classical solution consists in adding control facts. This will result in less readable rules, where domain knowledge is mixed with control knowledge.

In the 4-queens problem, facts created by the *initialize* rule are transmitted to the RETE nodes of the *find_solution* rule before any contradiction rule is fired. All the work done to instantiate the rule 256 times and compute all the labels will have to be defeased. This problem disappears if we split the *find_solution* rule :

```
(Rule end_solution priority 10
  (start_finding_solutions)
  (queen 1 ?i)   (queen 2 ?j)
  (queen 3 ?k)   (queen 4 ?l)
  →   (assert (solution ?i ?j ?k ?l)))

(Rule start_solution priority 50
  →   (assert (start_finding_solutions)))
```

Combination of tuples in the memory nodes of the *find_solution* rule will be delayed until the *(start_finding_solutions)* fact is created. This control fact will be added by another rule that will be fired only after all contradiction rules are fired. Thus, only consistent tuples will be created and transmitted, nogood ones will be discarded. This technique results in important performance gains.

The multiple agendas mechanism that is described in (Chehire 1990) is a more convenient way of controlling fact propagation in the RETE network. However, this mechanisms is aimed at providing blackboard like control in OPS-like systems. The control over fact propagation is only a by-product and optimizes the RETE network by focussing it on the most promising nodes with regard to the solution under evaluation. If this mechanism is used to optimize the ATMS computations , conflicts may arise with its use for pure control over rule packets scheduling. We thus need to provide another mean of optimizing contradiction handling.

Optimizing the contradiction handling.

When a memory node is shared between a contradiction and other rules, we need to fire the contradiction before transmitting the fact tuples to the other rules. Therefore, in such a case, all the tests for a contradiction have to be executed before the tests for transmitting the fact tuples to the successor nodes. The contradiction rules will not be queued in the agenda with the other rules, but will have to be fired as soon as instantiated.

In order to stress the important gains of the proposed optimization, let us rewrite the *find_solution* rule of the 4_Queens problem in the following form :

```
(rule find_solution
    ?q1:(queen ?i ?j)  ?q2:(queen ?k ?l)
    (join   ?q3:(queen ?s ?t)  ?q4:(queen ?u ?v))
    (test (different-facts ?q1 ?q2 ?q3 ?q4))
 ->  (print-solution (?i ?j) (?k ?l) (?s ?t) (?u ?v)))
```

The join keyword in this rule transforms the comb shape of the RETE in a balanced tree. This optimization technique enables to ensure a better sharing of nodes in the network. We show the size of the memory nodes in the cases of loose coupling, and tight coupling with special contradiction handling:

Loose coupling approach

The *find_solution* rule is instantiated 43680 times, and the select step of the inference engine will discard all but 48 of them.

It is important to note that the tight coupling approach without special contradiction handling directly in the RETE is even worse than loose coupling, in this example. Indeed, all the work done in the loose coupling approach has still to be done, but on top of that, all the fact tuples stored in the memory nodes and that become inconsistant after the firing of the contradiction rules, have to be removed.

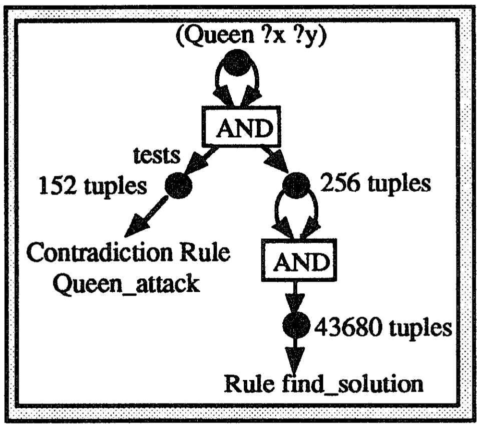

Figure 4 : Size of memory nodes using the loose coupling approach.

Tight coupling with special contradiction handling approach

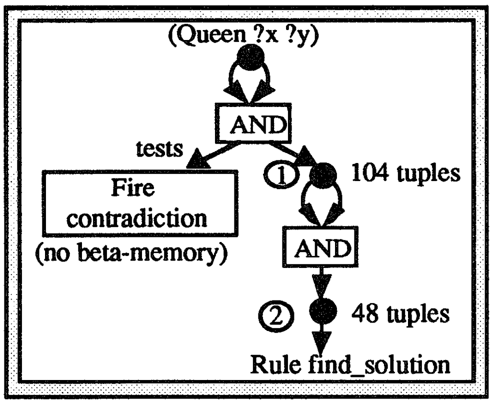

Figure 5 : Size of memories using tight coupling approach, and handling contradictions in join nodes.

When a new hypothesis is sent to the RETE network, it is combined with previous queen hypotheses. If a pair of queens satisfies the contradiction tests, a nogood is created; otherwise the pair is stored for further transmission in the network. This transmission occurs when all tuples involved in the current fact transmission have been tested for contradiction. The 152 contradictions are fired as soon as satisfied, and only 104 tuples are stored in the first β-memory node, instead of 256. These tuples are then combined, and since all contradictions have been first discovered, only the 48 valid combinations are stored in the final β-memory node. The number of labels computations does not exceed 256 in the first AND-node, and 104*104=10816 in the second. In fact, only 6192 label computations occurred in the match step of the inference engine (due to the elimination of tuples containing several time the same queen), whereas 43680 were needed in the select step of the loose coupling approach. Moreover, label computations are much more efficient in the tight coupling approach since they involve only two labels at a time.

Conclusions and future work

In this paper, we have shown that the choice of a method for integrating an OPS-like inference engine and an ATMS has important consequences on the performances of the overall system.

Determining when loose or tight coupling should be used is greatly application-dependant. The bookkeeping and memory-nodes updating needed in the tight coupling approach are significant overheads.

However, in a combinatorial application involving many contradictions, such as the N_Queens problem, the loose coupling approach may become intractable, and exponential performance gains can be obtained using tight coupling with special handling of contradictions.

Therefore both possibilities should be offered in a generic expert system shell. A careful analysis of the problem then enables to choose the most appropriate coupling approach for a specific application.

We are currently investigating extensions of this work to the domain of contextual control of the inference engine over the ATMS (Dressler & Farquhar 1990). The tight coupling approach, associated with the multiple agendas mechanism, provide a good framework for implementing an efficient focus of attention for the ATMS, allowing to guide rule execution and limit label propagation.

Tight coupling also offers interesting possibilities in the domains of non-monotonicity and of default reasoning in OPS-like inference engines.

Acknowledgements

This work has been implemented in XIA, which is the Thomson-CSF environment for developping and delivering expert modules that can be embedded in conventional applications. XIA results partly from ESPRIT project P96 and from Thomson Strategic Project on AI.

The work described in this paper has been partially funded by the DRET.

We would like to thank Pr Michel Cayrol and Dr Pierre Tayrac of Paul Sabatier University (Toulouse, France) for valuable discussions and comments on ATMSs.

References

Brownston, Farrell, Kant & Martin. 1985. *Programming in OPS5 : an introduction to Rule-based Programming.* Addislon-Wesley Series in Artificial Intelligence.

Cayrol M. and Tayrac P. 1989. Les résolutions CAT-correctes et CCT-correctes, la résolution CAT correcte dans l'ATMS. In Proceedings of the Colloque International sur l'informatique cognitive des organisations.Québec

Charpillet F, Théret P.and Haton J.P. 1989. X-TRA : un moteur d'inférence comportant deux modes de compilation de règles TREAT ou RETE et un systeme de maintien de la vérité de type ATMS" In Proceedings of the ninth International Workshop on Expert Systems and their Applications, 285-299. Avignon, France.

Chehire T. 1990. Augmenting the RETE network to efficiently compile a blackboard system. In Proceedings of Expert Systems 90. 253-262. Ed. T.R Addis and R. Muir, British Computer Society Conference Series, London.

Dermott J.Mc, Newell A. and Moore J. 1978. The Efficiency of certain Production System Implementations. *Pattern-directed Inference Systems.* Academic Press.

Doyle J. 1979 A Truth Maintenance System. *Artificial Intelligence* 12:231-272.

Dressler O., and Farquhar. 1990. Putting the Problem Solver Back in the Driver Seat : Contextual Control over the ATMS. In Proceedings of ECAI 90 Workshop on Truth Maintenance Systems, European Conference on Artificial Intelligence,Stockholm Sweden.

Dressler O. 1989. An extended basic ATMS, In Proceedings of the 2nd International Workshop on Non-Monotonic Reasoning, Springer LNCS 346

Dressler O. 1988. Extending the ATMS. In Proceedings of the European Conference on Artificial Intelligence, 535-540, Munich,.

Forgy C.L. 1982. RETE : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence* 19:17-37.

Ghallab M. 1989. Optimisation de processus décisionnels pour la robotique. Thèse d'état, UPS, Toulouse.

Gupta A., Forgy C. and Newell A. 1982. High-Speed Implementations of Ruled-Based Systems. *ACM Transactions on Computer Systems*

de Kleer J. 1986. An Assumption based Truth Maintenance System. *Artificial Intelligence* 28:127-224.

de Kleer J. 1988. A general labeling algorithm for assumption-based Truth Maintenance. In Proceedings of the sixth National Conference on Artificial Intelligence, 188-192, Saint-Paul MN.

Miranker D.P. 1987. TREAT : A Better Match Algorithm for AI Production Systems. In Proceedings of the Fifth National Conference on Artificial Intelligence, 42-47, Seattle.

Scales D.J. 1986. Efficient Matching Algorithms for the SOAR/OPS5 Production System, Report N° KSL 86-47, Knowledge System Laboratory, Stanford Univ.

Schorr M.I., Daly, T.P., Lee H.S. and Tibbits B.R. 1986. Advances in RETE pattern matching. In Proceedings of the Fourth National Conference on Artificial Intelligence.

Reiter R. and de Kleer J., 1987. Foundations of Assumption-based Truth Maintenance System. Preliminary report. In Proceedings of the Fifth National Conference on Artificial Intelligence, 227-234, Seattle.

Tayrac P. and Cayrol M. 1990. ARC : an extended ATMS based on directed CAT-correct resolution. In Proceedings of ECAI 90 Workshop on Truth Maintenance Systems, European Conference on Artificial Intelligence,Stockholm Sweden.