AN EXTENDED ATMS FOR DECOMPOSABLE PROBLEMS

by

Hardeep Johar
Leonard N. Stern School of Business
New York University
40 West 4th Street
New York, New York 10003

and

Vasant Dhar
Leonard N. Stern School of Business
New York University
40 West 4th Street
New York, NY 10003

March 1991

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-91-3

ABSTRACT

When dealing with nearly decomposable problems such as those described by Simon (1973), the problem components may be worked on by different problem solvers that are spatially and temporally separated, with each problem solver constrained by assumptions it makes about the activities and choices of other problem solvers, that is by partial knowledge of the global problem. There are advantages to maintaining multiple solutions locally for as long as possible, even though a single final solution is desired. When it becomes less desirable to retract certain assumptions, these become constraints for other problem solvers and can be communicated to them via a truth maintenance system. We describe an extended architecture for an ATMS for these kinds of decomposable problems.

1. Introduction.

In his analysis on the structure of ill-structured problems, Simon [12] describes various planning and design problems where the entire problem is subdivided into tasks and separate agents work on the decomposed tasks. Since there is often only partial information exchange between problem solvers, each problem solver has to make working assumptions about the activities of the other related solvers. Further, some constraints might be realized during the problem solving process instead of being specified a-priori. At some point in time, the outputs of the different problem solvers must be integrated into an overall solution. Examples of these kinds of decomposable problems include the creation and execution of plans, and problems in engineering design.

Most often, problem solving requires some sort of search for finding a solution. Problem solvers that use search as a mechanism for finding solutions make assumptions and then test the validity of these assumptions in providing a consistent solution to a problem. Truth Maintenance Systems (TMS) attempt to further reduce this complexity by separating specific problem-solver activities from belief maintenance activities (see [9] for a summary of truth maintenance) The utility of Truth Maintenance Systems could be greatly enhanced if they could somehow exploit the reduced complexity coming from decomposition.

In this paper we examine the implications of having many problem solvers working towards a common global objective on the design of TMSs. In particular we show that these kinds of problems can be handled by an Extended ATMS architecture. Extensions to the ATMS architecture are necessary because more semantic information content in nodes is required so that the TMS can recognize the existence of multiple problem solvers and accordingly enforce a partitioning in the search space, and the ATMS has to be tailored to provide single solutions while still maintaining multiple contexts (since problem solver commitments may be tentative). Such problems can, in principle be modeled using integer programming techniques, but controling the search process in such systems is difficult. In comparision, generate-and-test solvers coupled to a TMS do much better [4].

2. Truth Maintenance Systems.

Two important types of TMS are justification-based TMS (JTMS) and assumption-based TMS (ATMS). JTMS are designed to maintain a single consistent state [6, 10, 11], while ATMS [3, 7] maintains several possible contexts simultaneously. An ATMS is suited for finding multiple solutions to a problem, and a lot of extra effort may be required to find a single solution. Ways to reduce this effort have been proposed by DeKleer, although the issue has in no way been resolved completely.

Both types of TMSs are designed to work under a problem solvers that searches for a solution. For many problems, complexity reduction requires decomposing the problem into component problems, with different problem solvers working quasi-independently on each component. For these problems,

the TMS can be viewed as working under multiple problem solvers.

In decomposable problems, involving different problem solvers, the activities of each solver may be separated temporally, in the sense that one problem solver begins only after another has finished, or spatially, in the sense that two or more problem solvers work simultaneously on different parts of the same problem. In either case, the activities of the solvers are interrelated. In the case of the temporally separated problem solvers, commitments made by the first problem solver impose constraints on later problem solvers. In the case of spatially separated problem solvers, each problem solver has to make assumptions about the activities of other problem solvers and to delay its own commitments as much as possible in order to retain flexibility. Commitments by a problem solver are treated as constraints by other solvers. These are less desirable to retract than assumptions. All assumptions and commitments are conveyed to the TMS and therefore constitute a globally shared repository. In both cases, the different problem solvers operate on some amount of shared knowledge.

Of the two kinds of TMS, the JTMS has certain inherent limitations for handling multiple problem solvers. Some of the limitations identified by De Kleer [3] are also relevant in the context of decomposable problems. Since a problem solver must delay commitment and examine alternate assumptions about the activities of other problem solvers, it becomes important for the TMS to maintain the results of alternate paths followed in the search. These may be useful later. A JTMS, however, can maintain only one consistent context and switching states is difficult. In addition, the JTMS resolves contradictions by dependency-directed backtracking. However, when the choices of different problem solvers are conflicting, the problem solvers should have the ability to examine the different choices possible (this is analogous to the *single-state* problem reported by De Kleer). Finally, the requirement that temporary inconsistency be permitted in the TMS while the different problem solvers are working on the problem, argues against out the use of a JTMS.

An ATMS on the other hand, allows the existence of multiple contexts. This permits the problem solver to examine different sets of assumptions, and to postpone commitment. In addition, a Dressler ATMS [7] permits the use of nonmonotonic justifications. An ATMS architecture would therefore appear to be suitable for decomposable problems. However, as we see in the next section, additional information needs to be given to the ATMS to find single solutions without unnecessary backtracking, and equally important, to provide a reasoning system where multiple solvers can negotiate conflicts instead of relying on search.

3. Requirements for Decomposable Problems.

Decomposable problems share certain characteristics. Chandrasekaran [1] describes design proposal methods that use decomposition to reduce the size of the search space. All constraints for subproblems may not be known before hand, constraint generation and problem solving go on simultaneously, and

there might be complex processes of commitments and backtracking.

A simple example is the design of a fluid pump (for example a pump used to push oil through a pipeline). The design of the pump may take several weeks and may in fact be completed after the production process has started. Two important design decisions are the choice of the fluid and the choice of the metal to be used as a casing. Each decision depends on and constrains the other. It is possible (though not usual) for the choice of the metal to be made, and production on the casing to have started, before the choice of fluid is made. In this example, both problem solvers can work independently. Also, it is possible that some constraints may not be known before hand, for example, a constraint of the form *fluid x cannot be used with metal p* may only be discovered during the problem solving process; the problem solver that commits first (for example the choice of the metal) effectively constrains the options of the second problem solver.

The characteristic that constraint generation and problem solving go on simultaneously in the different problem solvers implies that problem solvers have to make assumptions about the commitments made by other problem solvers. However, in a practical setting, a problem solver may not be able to wait for other problem solvers to terminate before it makes a commitment. In the fluid pump example, if other activities depend on the fluid choice (perhaps pipelines have to be constructed), then the problem solver whose task is to choose a fluid may do so before the metal has been chosen. The fluid choice will constrain the metal choice, but if at some later point when the metal choice is attempted, it is discovered that no remaining metal choice is consistent with the fluid choice, the fluid assumption will have to be retracted. At this point the problem solver should have access to other feasible fluid options it may have considered.

While the ATMS can keep knowledge about other feasible options since it maintains multiple contexts, extra work is required for finding single solutions. De Kleer [2] suggests the use of *one-of* disjunctions to force the ATMS to find a single solution. This is not, however, desirable since it places the major burden of determining a consistent labeling on the TMS instead of it being more of a negotiated process among the solvers (based on domain knowledge). In order to make this possible, the TMS must be able to distinguish among different problem solvers data and to abdicate control to them whenever backtracking is required. Since there may be a complex process of commitments and backtracking amongst the various subproblems and it becomes important to manage this process properly. Problem solvers should not be permitted to easily retract choices made by other problem solvers. Apart from complexity reduction, there could also be practical reasons for not allowing one problem solver to retract commitments made by another. In the fluid pump example, if the fluid has already been chosen, and production on pipelines to carry it has begun, then the problem solver that selects the metal should not be permitted to easily retract the fluid choice. One way of handling this problem is to change the fluid choice from an assumption to a premise. However, this is overly restrictive since under some circumstances, it may be necessary, though not desirable, to

retract a choice (for example if no metal can be used with the selected fluid). If the choice has been changed to a premise then the justification for the choice will no longer be accessible. It therefore becomes necessary for the second problem solver to have access to the fact that the fluid was chosen by a problem solver, that it is not a inviolable constraint, and the justification for choosing the metal.

The implication of this requirement is that the search space should somehow be partitioned for the different problem solvers to prevent them from encroaching on commitments of other solvers. An ATMS has no way of doing this. This problem is best clarified by an example.

Assume a simple choice problem like the fluid pump example described earlier. There are four choices x, y, p and q, where x and y represent fluids, and p and q represent metals. There are two problem solvers, PS1 and PS2 working on the problem with PS1 selecting a fluid and PS2 selecting a metal. The problem can be encoded as follows:

PS1: Find values for x and y given the following constraints:
x + y = 1
x belongs to {0,1}
y belongs to {0,1}

and

PS2: Find values for p and q and x given the following constraints
p + q = 1
p belongs to {0,1}
q belongs to {0,1}
x + p = 1

The interpretation of the above formulation is exactly one of x or y, one of p or q, one of p or x will be selected. Only PS2 knows of the constraint involving x and p, perhaps because PS1 has incomplete knowledge about the problem or because the constraint was discovered by PS2 while it was working on the problem. Now assume that the problem solvers are temporally separated, i.e. first PS1 starts and completes its work and later PS2 starts. In an ATMS problem solving architecture, as a result of the activities of PS1, node **(x + y = 1)** would be labeled as follows:

<x+y=1,{{x=0,y=1},{x=1,y=0}},{{x=0,y=1,0+1=1},{x=1,y=0,1+0=1}}>

where the first component is the datum, followed by the label and justifications. The above provides two consistent environments for the node. Now, if the need for a single solution was expressed in the form of a one-of disjunction, one of the two environments would figure in the label, i.e the node would contain

<x+y=1,{{x=1,y=0}},{x=1,y=0,0+1=1}}>

which can be interpreted as fluid x being selected.

Now PS2 goes to work. Suppose it first tries the assumptions x=0, p=1, and the constraint x+p=1. Since in the current state of the ATMS x has a value of 1, the ATMS determines an alternate consistent environment under which x+p=1 holds. The one-of disjunction would be consistent in the alternate environment {x=0,y=1,p=1,q=0}, and this could be the one consistent solution found by the TMS. This may not be desirable since it would raise the complexity (PS2 has to redo the work of PS1) of the problem, or it may violate practical constraints. If work on the pipeline to carry the fluid has already begun, it might be advisable to choose x=1, q=1, y=0, p=0 rather than the new solution found by PS2. The only way to prevent PS2 from selecting the x=0 solution (i.e. changing the solution found by PS1) is to change status of x=1 from an assumption to a premise. But making this change would mean loosing the information that alternate consistent environments were found by PS1. This could be fatal if there is no feasible solution with x=1 (for example if q=0 or x+q=1 are added as constraints to PS2). The ATMS would have no knowledge about why x was set to 1. In the best case PS2 would have to redo all the work of PS1, and in the worst case PS2 would have no knowledge about the fact that x=1 was an assumption and not a premise. In the latter case PS2 may conclude that there is no feasible solution, when in fact one exists. Figure 1 illustrates the difference between treating x=1 as an assumption and as a premise.

The problem arises arises because the ATMS (and the JTMS) does its label propagation work (contradiction resolution in the case of the JTMS) syntactically, not on the semantics of the problem, which are known only to the problem solver. In the case of decomposable problems, it may be both impractical and undesirable (for complexity reduction reasons) for every problem solver to have a complete understanding of the problem. In the next section we introduce some additional semantics in ATMS nodes that effectively serve to partition the problem space for different problem solvers.

4. An Extended ATMS for Decomposable Problems.

4.1. Extending the ATMS.

We propose using the extended ATMS described by Dressler (see [7] for a description). The extended ATMS allows the use of non-monotonic justifications in the form of **out-assumptions** and also allows the use of default reasoning. The mechanics of Dressler's ATMS is not important here so we refer the reader to Dressler's paper for more detail. We suggest an adaptation to the extended ATMS to improve it's usefulness in finding single solutions, and so that the ATMS can help partition the search space for multiple problem solvers. We introduce two special kinds of nodes **Objective nodes**, and **Commit nodes**. Objective nodes are necessary to represent the objective of each problem solver and to maintain alternative consistent environments, while commit nodes record the commitments made by problem solvers, so that these commitments can be communicated to other problem solvers. **There is exactly one**

objective node and one commit node associated with each problem solver.

4.1.1. Objective Nodes.

Objective nodes are treated in exactly the same way as other ATMS derived nodes, i.e. they have justifications, hold in environments, and belong to contexts. Each objective node is, however, explicitly identified with a problem solver. In a manner analogous to derived ATMS nodes, the structure of an objective node is as follows

<datum,label,justifications,tag>

where datum represents the objective node, label consists of the set of consistent environments for the node, justifications are the justifications (given by the problem solver) for the node, and tag is additional semantic information provided to the TMS which associates an objective node with a problem solver.

We illustrate the concept of a objective node through the following (simple) example. Assume that the purpose of the problem solver is to solve the problem PS1 as defined in the previous section. In this problem the objective node is the node corresponding to the datum **(x+y=1)**. The problem solver can, for example, use the following rule to identify an objective node

x=N,y=M,N+M=1 -> x+y=1, mark-node-objective(PS1).

where commas represent conjunctions and *mark-node-objective* tells the TMS to mark the node corresponding to the datum **(x+y=1)** as an objective node for PS1. It is easy to see that, if all environments are computed, in this case the structure of the objective node in the TMS is

<x+y=1,{{x=0,y=1},{x=1,y=0}},{{x=0,y=1,0+1=1},{x=1,y=0,1+0=1}},Objective(PS1)>

The tag *Objective(PS1)* indicates that this node represents the solution set for problem solver PS1.

The purpose of the objective node is to allow the TMS to inform the problem solver of the solution choices available to it. Since the label of an ATMS node consists of minimal sets of assumptions under which the datum is consistent, each environment in the label of the objective node corresponds to a possible solution. In the above example there are two possible solutions, {x=0,y=1} and {x=1,y=0}.

4.1.2. Commit nodes.

Commit Nodes are used to reflect solution choices made by the problem solver. The primary purpose of commit nodes is to allow knowledge about a commitment made by one problem solver to be carried over to another problem solver and hence constrain the possibilities for the second problem solver. Commit nodes are constructed by the problem solver by providing the TMS with the selected solution, the environment for the solution, and any justifications it may want to record as reasons for the choice. The structure of the commit node is as follows

<datum, label, choice, justification, tag>

where the tag identifies the node as being a commit node.

In the above example, suppose the problem solver chooses {x=1,y=0} as the desired solution, and does so because there happens to be more of x than y available. It would inform the TMS accordingly which would create the following node:

<Resource-choice(x,y),{{x=1,y=0}},{{x=1,y=0}},{{more(x,y)}},commit(PS1)>.

The tag tells the problem solver that the node represents a commitment made by problem solver PS1, and tells the TMS that the node is of a special type.

At this point it is useful to reiterate that commit nodes reflect problem solver commitments and are constructed only at the request of the problem solver. Unlike objective nodes, commit nodes cannot be treated like derived ATMS nodes. In addition, it is the responibility of the ATMS to ensure that the environment provided by the problem solver is in fact consistent with the choice, and is present in the label of the problem solvers objective node.

4.1.3. Dealing with multiple problem solvers.

The ATMS must fulfill two requirements for dealing with more than one problem solver. It should help the problem solver pick and record one solution while maintaining knowledge about alternate feasible environments, and it should control problem solver choices by enforcing a partitioning of the search space. The first requirement is handled by the objective and commit nodes for an individual problem solver. The second requirement requires the use of commit nodes as partitioning devices. The use of commit nodes differs slightly depending upon whether problem solvers are temporally or spatially separated. Below we outline two examples illustrating how commit nodes are used in the two cases.

4.1.3.1. Example 1: Temporally Separated Problem Solvers.

Consider a simple resource selection problem of the type described in section 3. There are four resources x, y, p, q, and the problem first requires the selection of one resource from x and y and then requires the selection of one resource from amongst p and q. Assume that the two problem solvers are temporally separated, i.e. first PS1 does some work, and then PS2.

In the case of PS1, the objective node and commit nodes would be formulated as in the earlier discussion. Assuming that the commit node formed would be

<Resource-choice(x,y),{{x=1,y=0}},{{x=1,y=0}},{{more(x,y)}},commit(PS1)>.

this commit node serves as a constraint for PS2 as follows. Suppose PS2 tries to construct node x+p=1 with x=0 and p=1 as assumptions. The ATMS would examine all current commitments (commit nodes) to see if x=0 or p=1 violates

some current commitment. Since, in this case, x=0 violates the commitment made by PS1, the ATMS informs PS2 that x+p=1 is nogood because of Commit(PS1). Conversely, if PS2 attempts to construct the node x+p=1 with assumptions x=1 and p=0, the ATMS determines that x=1 is currently a commitment made by PS1 and places Commit(PS1) in the label of x=1. The node x=1 is therefore represented as:

<x=1,{{Commit(PS1)}},nil>

where nil implies that there are no justifications for the node. Node x+p=1 is represented as

<x+p=1,{{Commit(PS1),p=0}},{{p=0,x=1,1+0=1}}>

Next PS2 provides the TMS with the assumption q=1 to create the node p+q=1 along with the information that this is the objective of PS2. The ATMS computes the label for p+q=1 and the node is represented as

<p+q=1,{{p=0,q=1,Commit(PS1)}},{{x+p=1,q=1}},Objective(PS2).

and the commit node is

<res-choice(p,q),{{p=0,q=1,Commit(PS1)}},{{p=0,q=1}},nil,Commit(PS2)>.

By ensuring that Commit(PS1) is present in the label of both Objective(PS2) and Commit(PS2), the ATMS records the dependency of PS2's activity on commitments made by PS1. PS2 is prevented from changing the commitment made by PS1, but, if the need to change the commitment arises, the possibility is open to PS2 since it can examine the assumptions, premises and justifications used by PS1. Figure 2 illustrates the environment lattice for this example.

4.1.3.2. Example 2: Spatially separated problem solvers.

Assume the same example as in the previous section with one difference. Instead of the two decisions being temporally separated, they are spatially separated, i.e. they work simultaneously on their problems. The implication is that the PS2 (or PS1) does not have full information on the choices made by PS1 (or PS2) while it is attempting to arrive at a solution.

One way in which this problem can be resolved is for each problem solver to ignore the existence of the other, and to negotiate a solution after both have arrived at tentative solutions. This, however, may not be desirable because expensive recomputation may be required when choices made by other problem solvers are known. As an example suppose that the value of p was also set by PS1. The dependence of the value of p on PS1 would not be reflected in the objective node of PS2, and if the dependence is complex, i.e. it affects nodes in PS2 not directly related with the objective node, then expensive recomputation would be necessary.

The real problem here is the non-monotonicity of the deKleer ATMS. The solution is to bring non-monotonicity into the ATMS by using Dressler's [7] extended ATMS. Non-monotonic assumptions are coded as out assumptions. In our example the only assumption not in the PS2's control is the value of x. The

addition of two new assumptions, *out(x=1)* and *out(x=0)* reflect problem solver PS2's uncertainity about the outcome of PS1. This would lead to the creation of two new assumptions, out(x=0) and out(x=1) which would be present in the labels of p=0 and p=1 respectively. The objective node for PS2 becomes

<p+q=1,{{out(x=1),q=0},{out(x=0),q=1}},{{p=1,q=0},{p=0,q=1}},Objective(PS2)>.

This can be used to arrive at a temporary commitment for PS2. When, after negotiation, x=1 is confirmed, and Commit(PS1) is added, the first option in the objective node becomes a nogood because out(x=1) becomes a nogood, and the new objective node is the same as in the end of example 1.

4.2. Computational Requirements.

The extended form of the ATMS described above can be implemented with minimal extensions to the available ATMS algorithms. Objective nodes are exactly the same as derived nodes in a deKleer or Dressler ATMS and no extra work has to be done to compute labels for these nodes. All the ATMS algorithm needs to know is that the node in question is a objective node. It can use this information to tag the node accordingly, inform the problem solver if the label for such a node changes, and it has to ensure that there is only one objective node for each problem solver. This can be done with a minimal extension of the ATMS label update algorithm.

Commit nodes are computed a little differently and cannot be treated as ordinary derived nodes. However, for these nodes too, a simple extension of the label update algorithm suffices. This is because commit nodes are intrinsically linked to objective nodes and need only be modified when the corresponding objective node is modified, or when the problem solver explicitly modifies it. In the latter case the ATMS would work in two steps. In the first step it would make the necessary modification in the commit node, and in the second step, this modification would be propagated to other objective nodes created by other problem solvers. In the former case, i.e. when the objective node changes, the ATMS has to evaluate the effect on the commit node (has the current choice become a nogood?) and inform the problem solver accordingly. If the problem solver decides to make a change (necessary if the current choice has become a nogood), then the label update algorithm can proceed in the same manner as in the first case. In addition to the regular work of an ATMS, the extended ATMS has to examine current commitments and determine if any of these are violated.

That this method works can be seen by the following example. If, in the case of temporally separated problem solvers, PS1 revises its choice of x=1 because of some new constraint, its commit node changes to

<Res-choice(x,y),{{x=0,y=1}},{{x=0,y=1}},{},commit(PS1)>

The ATMS notes the change in a commit node and recomputes labels for nodes which have Commit(PS1) as a justification, in this case for p=0. The old label for Objective(PS2) will become a nogood since Commit(PS1) now expands to (x=0,y=1) and PS2 will be forced to look for a new solution.

5. Related and further research.

Decomposing problems to reduce complexity has long been examined in AI and Operations Research [12]. A recent issue of AI Magazine (Winter 1990) is entirely devoted to design problems, where decomposition is an important issue. Chandrasekeran [1] examines the issue of design proposal by constraint satisfaction and concludes that unless problems are decomposed, this method is computationally expensive. Our paper is an attempt to extend TMSs to explicitly take advantage of the computational gains from decomposition. In the fluid pump example described in this paper, if the problem is treated as a single problem there are 16 combinations that can be examined. Since the problem lends itself to easy decomposition, the maximum number of combinations that may need to be examined reduces to 8, a significant gain in complexity reduction.

Further research needs to be done on identifying ways to decompose problems. Real problems are often worked on by decomposing them using ad-hoc techniques. Mathematical programming problems are often decomposed by examining the set of constraints [8]. Perhaps an analysis of the relationships between assumptions can help in problem decomposition. One possible approach is to look for minimally connected subsets of the graph formed by assumptions as nodes and relationships as arcs. This approach, however, raises new issues about complexity and needs to be examined in greater detail.

Much of the motivation for this research comes from the viewpoint of organizational decision making where problem solving tends to be an ongoing process. Research in this area views knowledge used for decision making as being available in knowledge bases which are somewhat akin to databases (see for example [5]). Since problem solving is an ongoing process, and the knowledge base is in a constant state of flux, any such knowledge base has to be nonmonotonic, and a TMS for belief maintenance seems to be appropriate.

References

1. B. Chandrasekaran, "Design Problem Solving: A Task Analysis," *AI Magazine*, vol. 11, pp. 59-71, Winter 1990.
2. Johann DeKleer, "Problem-Solving with the ATMS," *Artificial Intelligence*, vol. 28, no. 2, pp. 163-196, 1986.
3. Johan deKleer, "An Assumption-based TMS," *Artificial Intelligence*, vol. 28, no. 2, pp. 127-162, 1986.
4. Vasant Dhar and Nicky Ranganathan, "Integer Programming vs. Expert Systems: An Experimental Comparision," *Comm. of the ACM*, vol. 33, no. 3, pp. 323-336, March 1990.
5. Daniel R. Dolk and Benn R. Konsynski, "Knowledge Representation for Model Management Systems," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, pp. 619-628, November 1984.

6. Jon Doyle, "A Truth Maintenance System," *Artificial Intelligence*, vol. 12, pp. 231-272, 1979.

7. Oskar Dressler, "An Extended Basic ATMS," *Proceedings of the 2nd Intl. Workshop on Non-Monotonic Reasoning*, Springer-Verlag, 1988.

8. C. Lemarechal, "Nondifferentiable Optimization," in *Handbook in Operations Research and Management Science: Volume 1 Optimization*, ed. Nemhauser et. al., pp. 529-572, Elsevier, Amsterdam, 1989.

9. Joao P. Martins, "The Truth, the Whole Truth, and Nothing But the Truth," *AI Magazine*, vol. 11, pp. 7-26, January 1991.

10. D. McAllester, "Reasoning Utility Package," *MIT AI Memo 667*, Cambridge, Ma., 1982.

11. C. Petrie, D. Russinoff, and D. Steiner, *Proteus 2: System Description. MCC Technical Report AI-136-87*, May 1987.

12. H. A. Simon, "The Structure of Ill-structured Problems," *Artificial Intelligence*, vol. 4, no. 1, pp. 181-202, 1973.

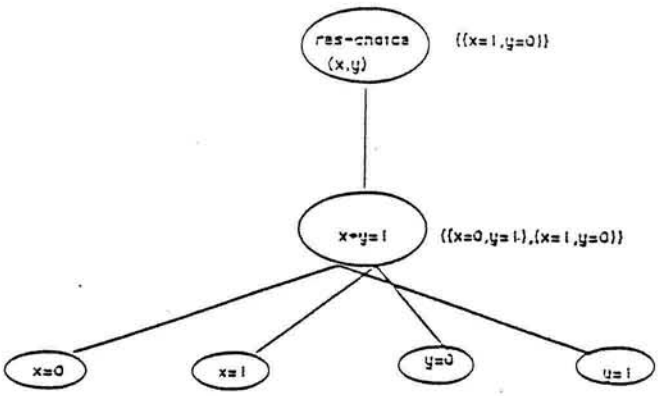

Figure 1a. x=1 as an assumption.

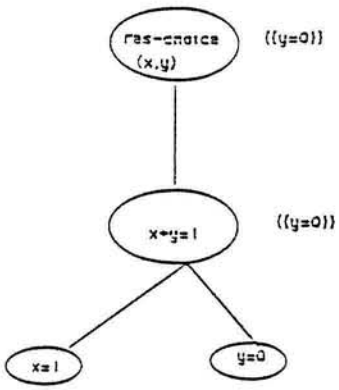

Figure 1a. x=1 as a premise

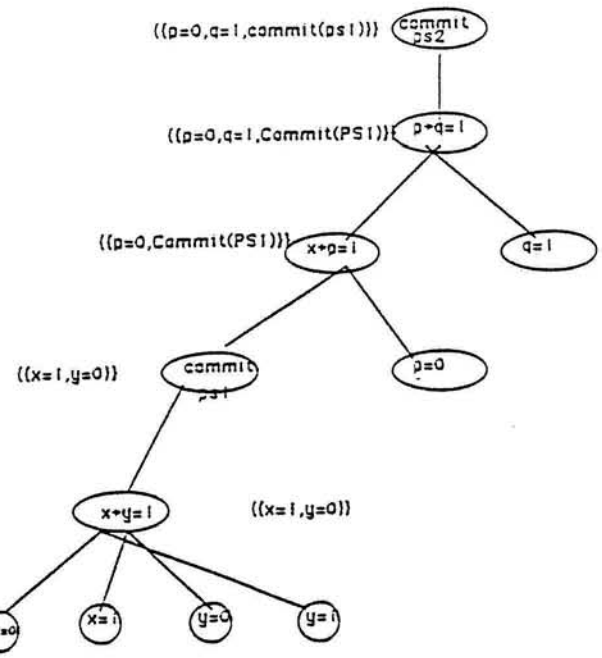

Figure 2: Environment lattice for example 1