PROVING THE IMPOSSIBLE IS IMPOSSIBLE IS POSSIBLE, WITH APPLICATIONS TO ROBOT WORLDS*

by

L. Siklóssy and J. Roach

February 1973 TR-9

The Department of Computer Sciences

University of Texas at Austin

*Work partially supported by grant GJ-34736 from the National Science Foundation.

Abstract

A novel technique, called hereditary partitions, is introduced. It permits the rigorous proof that, in a given axiomatization, certain states can never be reached. The technique is implemented in a computer program, DISPROVER, and is applied to robot worlds. DISPROVER cooperates with a path-finding program when the latter encounters difficulties.

1. Introduction.

Theorem proving and problem solving programs are sometimes successful in finding a proof to an actual theorem, or in solving a problem which does have a solution. On the other hand, presented with an expression which is not a theorem, or with an unsolvable problem, the programs are generally incapable of discovering, in a positive sense, that the expression is not a theorem, or that the problem is indeed unsolvable. The usual diagnostic would be: "I cannot solve the problem, because my resources are exhausted, or because I am stuck somewhere. However, the problem may be solvable. I just don't know."

We are interested in developing programs which, given a problem, will try to solve it. If they cannot solve the problem, they will try to show that it cannot be solved. A program which performs the last function will be called a disproving program. To build such programs, we use a technique which we named hereditary partitions. This technique has some generality and forms our basis for a program, DISPROVER, which has been applied to disprove goals in robot worlds, i.e. DISPROVER proves rigorously that, in a given world, there is absolutely no way to attain some particular goal.

DISPROVER also cooperates with another program, LAWALY, which tries to find paths to goal states. Sometimes, when LAWALY cannot solve a (solvable) problem, DISPROVER - which clearly cannot disprove the solvable problem - gives LAWALY additional information which permits a solution to be found. In other cases, when DISPROVER cannot immediately disprove an impossible problem, LAWALY sometimes can give DISPROVER additional information for a disproof (i.e. a proof that something cannot be proved) to be found.

We shall now describe some additional motivation for our work, discuss the related technique of hereditary properties - which is a degenerate case

of hereditary partitions - and give several examples of disproofs, terminating with examples of cooperation between DISPROVER and LAWALY.

2. Why Disproofs?

Somehow, it is much more romantic and challenging to show that, in the whole wide world, there is absolutely no way that something can be proved, than to find one, of possibly many, proofs to some theorem. Moreover, work on robots introduces a practical - and necessary - application of disproving programs. In robot problem solving systems, parts of the physical world, including some of the robot's capabilities, are simulated as a model. It is desirable that the model conform reasonably closely to the physical world, otherwise the robot may "think" that it can do some things - which are in fact impossible - or cannot do some other things - which are in fact possible. For example, in [1], the robot can be in two different places at the same time! (see [2] and [3].)

3. Techniques for Disproofs.

Many systems of interest, such as predicate calculus, are undecidable, i.e. for a given statement in the system it is not possible to determine, in general, whether a given statement in the system can or cannot be proved. If, in a particular case, we wish to show that a statement is not a consequence of some axioms, the standard procedure is as follows: find a model in which the axioms are true, but the statement is known to be false. Such model building is truly an art, and is acquired through much experience.

A technique with some generality has been called <u>hereditary properties</u> (see [4] for some examples). Consider a checkerboard from which we remove two opposite squares. This mutilated checkerboard cannot be covered by domi-

noes. To obtain a disproof, we notice that whenever we add another domino on the checkerboard, the number of white and black squares that have been covered remains the same. This property - the equality between the number of black and white squares that have been covered - is hereditary, that is it does not change as any allowable move - putting a domino on the checkerboard - is performed. The disproof is complete when we notice that, in the mutilated checkerboard, the number of black squares does not equal the number of white squares, (the difference is two.)

The technique of hereditary properties can be summarized as follows:
-the original state(s) of the model has (have) some property.
-whenever a state has this property, all states obtained from it by all allowable moves still have this property.
-the goal state which we are trying to attain does not have this property. Hence the goal is unattainable.

4. Hereditary Partitions.

Hereditary partitions generalize the basic idea of hereditary properties. The technique of hereditary partitions can be summarized as follows:
-call the set of all states that can be achieved from the original state(s) by all legal sequences of moves the attainable world.
-the attainable world can be partitioned into desjoint partitions. Hence each original state is in some partition.
-the goal state which we are trying to attain does not belong to any of the partitions. Hence the goal is unattainable.

Obviously, hereditary properties correspond to the case where there is only one partition.

We notice that if we apply a legal move to a state in some partition,

we obtain a state in the same or some other partition. We can say that partitions are closed under legal moves. In fact, as long as this closure property is maintained, we might just as well add to the partitions some unattainable states (i.e. states which are "meaningless") if that makes life simpler; as long as closure is maintained, and the goal state is not in any of the partitions, the disproof is valid.

In practice, the problem is, of course, to build the appropriate partitions. We shall see an example in the next section.

5. Example of a Disproof using Hereditary Partitions.

We shall consider robot worlds axiomatized in a manner similar to that used in [1]. The world is described as a set of predicates, for example NEXTTO(ROBOT, BOX2). Moves in the world are operators which must satisfy some preconditions, and their effect on the world is specified by a delete set and an add set.

Let us consider a subworld of the world in [1]: a robot and three boxes, BOX1, BOX2 and BOX3, in a room. The only relevant operators for our problem are (somewhat simplified from [1]).

goto(object), meaning: robot goes next to an object.

Preconditions: ONFLOOR.

Delete set: ATROBOT($), NEXTTO(ROBOT, $).

Add set: NEXTTO(ROBOT, object).

push(object1, object2), meaning: robot pushes object1 next to object2.

Preconditions: PUSHABLE(object1) $\wedge$ ONFLOOR $\wedge$ NEXTTO(ROBOT, object1).

Delete set: ATROBOT($), AT(object1, $), NEXTTO(ROBOT, $), NEXTTO(object1, $), NEXTTO($, object1).

Add set: NEXTTO(object1, object2), NEXTTO(object2, object1), NEXTTO(ROBOT,

object1).

We assume that boxes are PUSHABLE, that the robot could climb on and off boxes, and possibly do a lot of other mischievous actions, none of which would help her get two boxes next to each other. We now wish to solve the problem: get the three boxes next to each other, i.e. find a path from an original world which includes:

ONFLOOR ATROBOT(A) AT(BOX1, A1) AT(BOX2, A2) AT(BOX3, A3) to a goal state which includes:

NEXTTO(BOX1, BOX2) NEXTTO(BOX2, BOX3).

A solution is: goto(BOX1), push(BOX1, BOX2), goto(BOX3), push (BOX3, BOX2).

However, a more symmetric description of the goal state answering the statement "the three boxes are next to each other" would be:

NEXTTO(BOX1, BOX2) NEXTTO(BOX2, BOX3) NEXTTO(BOX3, BOX1).

We shall now give a disproof of this goal, i.e. show that it cannot be achieved.

The partitions are described in terms of some anchor predicates and their negatives. As a first choice, DISPROVER chooses the three predicates from the goal as anchors. We shall abbreviate these predicates as P12, P23 and P31. The original state belongs to the partition:

Partition1: $\neg$P12 $\neg$P23 $\neg$P31.

This partition contains all states, whether attainable or not, which satisfy $\neg$ P12 $\neg$P23 and $\neg$P31, i.e. which do not contain any predicate of the form: NEXTTO(BOX_i, $BOX_{i\ (\mathrm{mod}\ 3)+1}$), i = 1,2,3.

If the robot could juggle she would move into a new state which would presumably still be in the partition. All goto operations do not, in turn, make her go out of the partition. But let us consider: push(BOX1, BOX2). This oper-

ator is applicable to partition1 -although not to the original world, because the robot does not start next to BOX1-, since the state:

NEXTTO(ROBOT, BOX1) ¬P12 ¬P23 ¬P31

is a member of partition1. Hence by applying push(BOX1, BOX2) we move out of partition1, and must create a new partition2, specified by:

Partition2: P12 ¬P23 ¬P31

Similarly, we create partition3

Partition3: ¬P12 P23 ¬P31, and

Partition4: ¬P12 ¬P23 P31. (see Figure 1.)

From partition2, we can either go to partition1 by doing, for example, push(BOX2, BOX3); or stay in partition2 by doing, among other possibilities, goto(BOX2); or move to a new

Partition5: P12 P23 ¬P31,

by applying push(BOX3, BOX2) to the state including:

NEXTTO(ROBOT, BOX3) P12 ¬P23 ¬P31

of partition2. Similarly, we create:

Partition6: P12 ¬P23 P31, and

Partition7: ¬P12 P23 P31.

At that point, however, no new partitions can be created! Every legal move either leaves the robot in the same partition, or takes her to one of the other partitions. Since the goal state is not in any of the partitions, the disproof is complete.

DISPROVER, programmed in LISP 1.5 and run interpreted on the University of Texas CDC 6600 found the above disproof, for the world of [1], in about 7 seconds.

Another disproof, in the world of [1], concerns the goal: (AT BOX1 A) (STATUS LIGHTSWITCH1 ON), starting from the original state which includes: (AT BOX1 A) (STATUS LIGHTSWITCH1 OFF). The robot needs to climb on BOX1 to turn on LIGHTSWITCH1, but she is then incapable of returning BOX1 to its original location. The disproof, by DISPROVER, took about 3 seconds; three partitions were built using the anchor predicates from the goal state.

6. Bootstrapping in DISPROVER.

The anchor predicates -which determine the partitions- are crucial for DISPROVER to be successful. In some cases, DISPROVER can change its set of anchor predicates. We shall use a disproof as an example of this capability. We expand the world discussed previously via an operator gotoloc(loc), meaning: robot goes to location loc in room rm.

Preconditions: ONFLOOR $\wedge$ $\exists$rm(LOCINROOM(loc, rm)).

Delete set: ATROBOT($), NEXTTO(ROBOT, $).

Add set: ATROBOT(loc).

We will disprove the state ATROBOT(A1), where A1 was used in AT(BOX1, A1). As in [1], there is no predicate LOCINROOM(A1, x) for any x, hence the task is obviously impossible in the axiomatization.

The initial anchor predicate is obtained from the goal: ATROBOT(A1).

The initial state is contained in the partition:

Partition1: $\neg$ATROBOT(A1).

The state: ONFLOOR $\wedge$ LOCINROOM(A1, x), for x anything, is a member of this partition1 -even though it is unattainable-, and the operator gotoloc(A1) can be applied to this state, to obtain

Partition2: ATROBOT(A1).

Since the goal we are trying to disprove is a member of partition2, the dis-

proof fails.

At this point, DISPROVER tries to extend its set of anchor predicates by adding to those already used, all those that were preconditions of the operator(s) that permitted a transition to the partition (here partition2) which we were hoping not to reach in the disproof. The new set of anchor predicates is:

ATROBOT(A1) $\wedge$ ONFLOOR $\wedge$ LOCINROOM(A1, x).

DISPROVER tries again (and will succeed with the disproof, otherwise we would not have chosen this example!) The original state is in:

Partition1: $\neg$ATROBOT(A1) $\wedge$ ONFLOOR $\wedge$ $\neg$LOCINROOM(A1, x).

From this partition, if the robot climbs on something, we can go to:

Partition2: $\neg$ATROBOT(A1) $\wedge$ $\neg$ONFLOOR $\wedge$ $\neg$LOCINROOM(A1, x).

However, no further partitions can be generated, completing the disproof. W.W.W. (what we wanted!)

7. LAWALY helped by DISPROVER.

In some cases, LAWALY, the path-finder we use to solve robot planning problems [5], does not find a path even though one exists. An example will help to illustrate the difficulties she encounters. Figure 2 shows the initial and final states of the robot world. The robot must achieve: CLOSED(DOOR) $\wedge$ NEXTTO(ROBOT, BOX), from the initial state: INROOM(ROBOT, A) $\wedge$ CLOSED(DOOR) $\wedge$ INROOM(BOX, B). (See Figure 2.) LAWALY may decide to work first on the CLOSED(DOOR) condition, or first on the NEXTTO(ROBOT, BOX) condition. Consider the first case. LAWALY finds the door already closed in the initial state, so she wants to obtain the NEXTTO condition. To do that, she must enter Room B, and to do that go through the DOOR. But that would mean opening the DOOR, and hence undoing what she had already achieved, -CLOSED(DOOR)-, and so she

decides to try the conditions in the reverse order. To be NEXTTO(ROBOT, BOX), she goes to DOOR, opens it, goes through it, and then goes next to BOX. At that point, she realizes that she must still close the DOOR. However, that would make her undo something she wanted, namely, NEXTTO(ROBOT, DOOR), so she quits, having failed.

The problem is now passed to DISPROVER. The anchor predicates are taken from the goal, and the following partitions are built:

Partition1: ¬NEXTTO(ROBOT, BOX) CLOSED(DOOR).

Partition2: ¬NEXTTO(ROBOT, BOX) ¬CLOSED(DOOR).

Partition3: NEXTTO(ROBOT, BOX) ¬CLOSED(DOOR).

Partition4: NEXTTO(ROBOT, BOX) CLOSED(DOOR).

Since Partition4 includes our goal, the disproof fails.

Partition4 was obtained by applying the operator: gonext(object), Preconditions: (ONFLOOR) $\wedge$ $\exists$x (INROOM(ROBOT, x) $\wedge$ INROOM(object, x)), to the state intermediary-state: ONFLOOR $\wedge$ INROOM(ROBOT, B) $\wedge$ INROOM(BOX, B). DISPROVER suggests to LAWALY that the original problem might be solved by splitting it up into two successive problems. The first problem is to go from the initial state to a state containing intermediary-state above; the second problem is to go from there to the final state. LAWALY does in fact solve the original problem in this way.

8. A Collaborative Failure.

We now describe a solvable task which is not solved by the collaboration between DISPROVER and LAWALY.

The initial and final states of the task are shown in Figure 3. The final state is: ON(ROBOT, BOX) $\wedge$ INROOM(ROBOT, B). Again, for essentially the same reasons as before, LAWALY fails to solve the problem. DISPROVER

cannot find a disproof, but suggests the intermediary-state: INROOM(ROBOT, B) $\wedge$ NEXTTO(ROBOT, BOX). Once more, LAWALY fails, again due to her stubborness in insisting on finishing a subtask completely before starting another one. DISPROVER finds no disproof (rightly so, since none exists) with the anchor predicates: INROOM(ROBOT, BOX) NEXTTO(ROBOT, BOX) ON(ROBOT, BOX). Moreover, DISPROVER suggests the same intermediary-state as before. Failure is accepted.

9. The Importance of Axiomatization.

The problem of section 7 could have been solved immediately by LAWALY, without DISPROVER's help, if it had been further specified as: CLOSED(DOOR) NEXTTO(ROBOT, BOX) INROOM(ROBOT, B). The problem of section 8 could have been solved immediately by LAWALY if it had been further specified as : INROOM(ROBOT, B) ON(ROBOT, BOX) INROOM(BOX, B). It could also be solved immediately by LAWALY if the climbon operator had specified as parts of its preconditions that the robot could climb on an object only if both she and the object were in the same room. Thus, we can see that the difficulties encountered may be due to the axiomatization used.

Another way of resolving the difficulties is to "patch" the goal descriptions to include consequences such as: a robot is in the same room as the object she is on, etc. Such a "patch" is a trivial program.

10. Conclusions.

The technique of hereditary partitions permits the disproofs of statements that cannot be made true. We have applied this technique to a disproving

program (perhaps the first such program in existence) which operates in simulated robot worlds. DISPROVER can be used to ascertain that physically undesirable states cannot occur in a model. We give examples of collaboration between DISPROVER and a powerful robot planning system, LAWALY. The discovery of appropriate anchor predicates to build the hereditary partitions, and the use of axiomatizations in which paths to or disproofs of states are facilitated, will require much additional research.

11. References.

[1] Fikes, R. E. and Nilsson, N. J. "STRIPS: A New Approach to the Application of Theorem Proving in Problem Solving," Artificial Intelligence, 2, 189-208, 1971.

[2] Review of [1]. Computing Reviews, 13, 5, 216-217, 1972.

[3] Siklóssy, L. Modelled Exploration by Robot. Technical Report 1, Computer Sciences Dept., University of Texas, Austin, 1972.

[4] Simon, H. A. On Reasoning about Actions, in: Simon, H. A. and Siklóssy, L. (Eds.) Representation and Meaning: Experiments with Information Processing Systems, Prentice-Hall, Englewood Cliffs, N.J. 1972.

[5] Siklóssy, L. and Dreussi, J. A Hierarchy-Driven Robot Planner which Generates its own Procedures. Technical Report. Computer Sciences Dept., University of Texas, Austin, 1973.

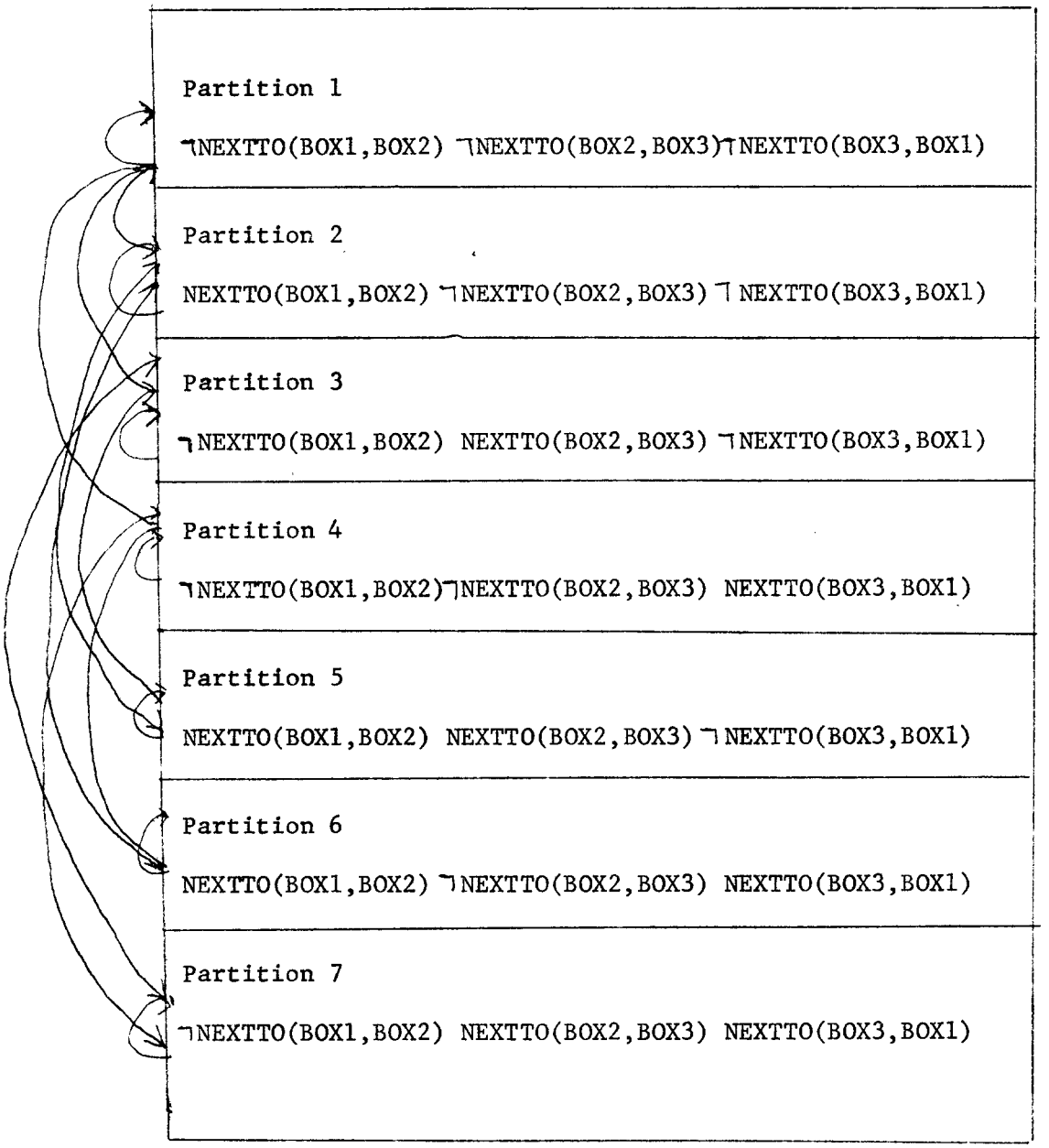

Figure 1. Disproof of NEXTTO(BOX1,BOX2)∧NEXTTO(BOX2,BOX3)∧ NEXTTO(BOX3,BOX1).

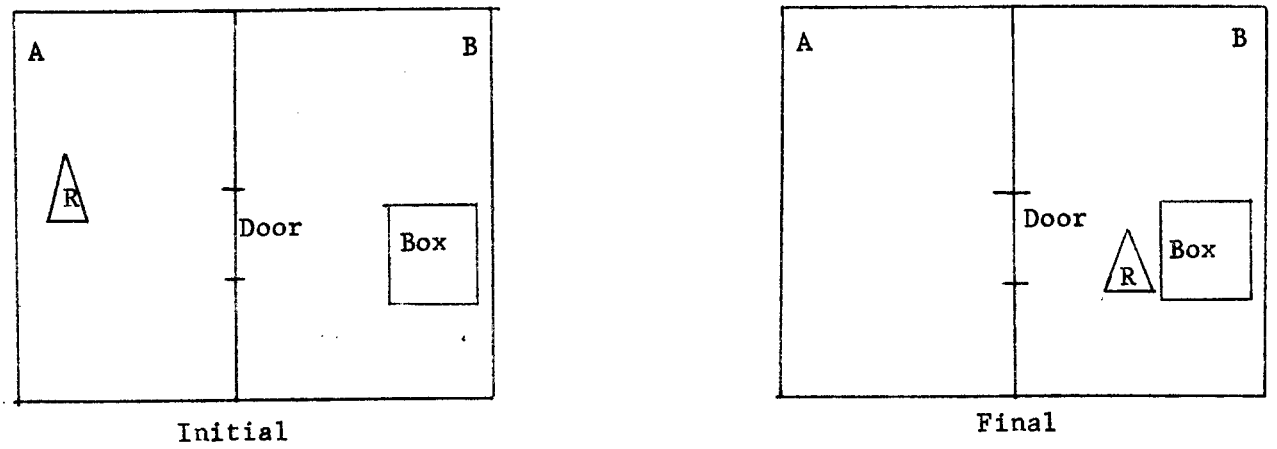

FIGURE 2.

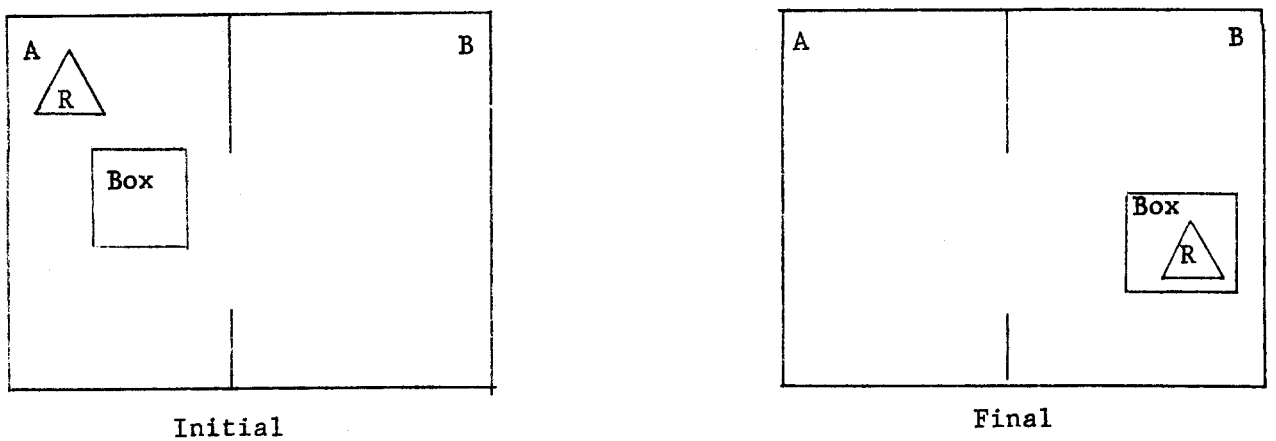

FIGURE 3.