PROVING THE IMPOSSIBLE IS IMPOSSIBLE IS POSSIBLE: DISPROOFS BASED ON HEREDITARY PARTITIONS*

L. Slklossy and J. Roach

Department of Computer Sciences

University of Texas, Austin, U.S.A.

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.

Key words: Robot Planning; Impossible Tasks; Theorem Prover; Statement Disprover; Disproof; Hereditary Properties; Hereditary Partitions; Problem Solving.

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, for a given set of axioms, operators and an initial 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.

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 [l], the robot can be in two different places at the same time.[2,3]

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

3. Some 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. The automatic building of disproving models appears beyond the present state of the art.

Since predicate calculus has been of great interest to workers in artificial intelligence, some additional comments ore in order. Given a well-formed formula wff in predicate calculus, three exclusive cases are possible:

a) wff is a theorem.
b) iwff is a theorem.
c) neither a) or b). In this case, -iwff is also in category c), This last case is by far the most common, in the sense that, given a finite vocabulary of constants, variables, function and predicate names, the number of well-formed formulae of a given length in category c) far surpasses the number of formulae in categories a) and b),

Given a wff in categories a) or b), a theorem prover based on resolution, for example, can determine after a finite time -which, in general, cannot be calculated in advance- in which of the two categories the wff belongs. To obtain this result, the theorem prover could time-share its efforts between trying to prove the inconsistencies of both wff and -|wff. However, if the wff Is in category c), then usually the theorem prover will go on forever. In only very few cases will the theorem prover stop because no new clauses are generated: a disproof of the wff has then been obtained.

A disproving program in predicate calculus would attempt to determine that a wff is not in category a). The above discussion shows that present theorem provers are at best very weak disproving programs.

It should also be noticed that a proof of the attainability of the negation of a statement does not imply that the statement itself can be disproved. For example, if a light can be turned on and off (at different times) then the statement STATUS(LIGHT ON) and its negation -STATUS(LIGHT ON) are both attainable, and hence neither can be disproved.

While the theorem proving approach gives results in far too few cases, and model building is too often ad-hoc, a disproving technique with some generality has been called hereditary properties (see [4] for some examples). Consider a checkerboard from which we remove two opposite squares. This mutilated checkerboard cannot be covered by dominoes. 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 ae 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. W.W.W. (what we wanted'.)

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 disjoint 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. Even further, to the partitions containing some attainable stares we can add some partitions containing no attainable states. The disproof is still complete if this expanded set of partitions is closed under legal moves -i.e. from one partition in the set any legal move leads us to the same or some other partition in the set- and if the goal state that we are trying to disprove is not in any of the partitions.

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 [l]. The world is described as a set of predicates, for example HEXTT0(R0BOT B0X2) , 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 [l]: *a* robot and three boxes, B0X1, B0X2 and B0X3, in a room. The only relevant operators for our problem are (somewhat simplified from [l]).
goto(object), meaning: robot goes next to an object.
Preconditions: ONFLOOR.
Delete set: ATROB0T($) NEXTTO(ROBOT $)*.
Add set: NEXTTO(ROBOT object).
push(objectl object2), meaning: robot pushes objectl next to object2.
Preconditions: PUSHABLE(objectl) A 0NFLOOR * NEXTTO (ROBOT objectl).

*$ means everything. At least that was the meaning before a series of devaluations'.

Delete set: ATROBOT($) AT(objectl $) NEXTTO(R0BOT $) NEXTTO(objectl $) NEXTTO($ objectl).
Add set; NEXTTO(objectl object2) NEXTT0(object2 objectl) NEXTT0(R0BOT objectl).

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(B0X1 Al) AT(B0X2 A2) AT(BOX3 A3)
to a goal state which includes:
NEXTT0(B0X1 B0X2) NEXTT0(B0X2 B0X3).
A solution is: goto(BOXl), push(BOXl B0X2), goto(B0X3), push(B0X3 B0X2).

However, a more symmetric description of the goal state answering the statement "the three boxes are next to each other" would be:
NEXTTO(BOXl B0X2) NEXTT0(B0X2 B0X3) NEXTTO(BOX3 BOXl).
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 negations. 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;
Partitionl; "IP12 ^ "P23 A -P31.
This partition contains all states, whether attainable or not, which satisfy -P12 A nP23 and AiP31, i.e. which do not contain any predicate of the form:
NEXTTO(BOX., B O X .^ $_{3)+1}$), 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(BOXl B0X2). This operator jl£ applicable to partitionl -although not to the original world, because the robot does not start next to B0X1-, since the state:
NEXTT0(R0BOT BOXl) *1V12 -T23* -P31
is a member of partitionl. Hence by applying push(BOXl B0X2) we move out of partitionl, and must create a new partition2, specified by:
Partition2: P12 A 1P23 A 1P31.
Similarly, we create partition3
Partition3: TP12 AP23 A *31, and
Partition^: TP12 A iP23 A P31. (see Figure 1.)
From partition2, we can either go to partitionl by doing, for example, push(B0X2 B0X3); or stay in partition2 by doing, among other possibilities, goto(BOX2); or move to a new
Partition5: P12 A P23 A -P31,
by applying push(B0X3 BOX2) to the state including:
NEXTTO(R0BOT B0X3) P12 P23 TP31
of partition2. Similarly, we create:
Partition6: P12 TP23 P31, and
Partition?: -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. W.W.W.

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

Another disproof, in the world of [l], concerns the goal: AT(B0X1 A) STATUS(LIGHTSWITCHI ON), starting from the original state which includes: AT(B0XL A) STATUS(LIGHTSWITCH1 OFF). The robot needs to climb on BOXl to turn on LIGKTSWITCH1, but she is then incapable of returning BOXl to its original location. The disproof, by DISPROVER, took about 3 seconds; three partitions were built using the *anchor* predicates from the goal state.

5.1 Classes of Impossible States.

Given some axiomatization of a world, namely an initial state and a set of legal moves -operators-, we can distinguish broadly between two main classes of impossible goal states:
a) goals including a subgoal which is not satisfied in the initial state and does not occur on the add set of any operator. For example, ii the robot cannot paint, and if COLOR(BOX1 BLUE) is true initially, then a goal including COL0R(B0XI PINK) cannot be achieved, and is disproved easily by DISPROVER using only one partition.

A variation of this class of impossible tasks occurs when a subgoal, subgoal., of the goal is satisfied in the initial world but must be destroyed, i.e. deleted, to achieve some other subgoal$_n$. Moreover, once deleted, subgoal cannot be added by any operator. The last example in section 5 above is a case in point.
b) Each subgoal. of the goal can be achieved independently, but the simultaneous realization of all subgoals is impossible, due to "side effects" caused by the delete sets. The main example in section 5 of the symmetric NEXTTO(BOX. BOX.) is a case in point.

In practice, what is most fun to do is to take someone else's world and disprove goal states which "intuitively" should be realizable!

5.2 Selection and Testing of Operators.

Given a set {U } of anchor predicates in an attempted disproof, DISPROVER first discards all operators which do not add or delete some predicate in {U} , since these operators are clearly irrelevant. The initial world is intersected with {V'} " {P|P'e{u)v IPE(U}} to give the first partition. All remaining (relevant) operators are applied to this partition in an attempt to create new partitions. Whenever a new partition is created, a test is made to see whether the goal state is in the partition. If it is, the disproof fails, and DISPROVER will try to bootstrap itself (section 6.) Otherwise, the partition is kept, and all relevant operators will be applied to it to try and obtain yet further new partitions. In artificial intelligence jargon, DISPROVER generates partitions breadth-first until the goal state is found -meaning the disproof failed for the given set of anchor predicates (see section 6)- or until no new partitions are found, which would then complete the disproof.

It should be noticed that with the form of preconditions used, only a few simple set operations are needed to determine whether an operator is applicable to a partition. If preconditions are generalized *to* arbitrary predicate calculus expressions, or include procedures, then It may well be impossible to decide whether an operator is applicable to a partition,

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 tn room rm.
Preconditions: ONFLOOR A 3rm(L0CINR00M(loc rm)).
Delete set: ATROBOT($) NEXTTO(ROBOT $).
Add set: ATROBOT(loc).
We will disprove the state ATROBOT(AI), where AI was used in AT(B0X1 AI). As in [I], 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(AI).
The initial state is contained tn the partition:
Partition! : lATROBOT(AI) .
The state: ONFLOOR LOCINR00M(AI x), for x anything, is a member of this partitionl -even though it is unattainable-, and the operator gotoloc(AI) can be applied to this state, to obtain
Partition2: ATROBOT(AI).
Since the goal we are trying to disprove is a member of partitton2, the disproof 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(AI) A ONFLOOR A L0CLNR00M(A1 x) .
DISPROVER tries again (and will succeed with the disproof, otherwise we would not have chosen this example'.) The original state is in:

Partitionl: 1ATR0B0T(A1) A ONFLOOR A -L0CINR00M(AI x) .
From this partition, if the robot climbs on something, we can go to:
Partition2: -ATROBOT(AI) ft 10NFL00R ft "IL0CINR00M(A1 x) .
However, no further partitions can be generated, completing the disproof, W.W.W.

DISPROVER finally fails to find *a* disproof if -besides economic reasons of time limit or memory exceeded- no new sets of anchor predicates are generated during bootstrapping.

7. LAWALY helped by DISPROVER.

Up to tlits point we have seen DISPROVER working alone. Now we will consider cooperation between LAWALY and DISPROVER. In some cases, LAWALY, the path-finder we use to solve robot planning problems , 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) A NEXTTO(ROBOT BOX), from the initial state: INR00M(R0B0T A) A CLOSED (DOOR) A 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 NEXTT0(R0BOT 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 BOX), 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:
Partitionl: -iNEXTT0(R0B0T BOX) CLOSED(DOOR).
Pa$_T$tttion2: 1NEXTT0(R0B0T BOX) ICLOSED(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(ob.ject), Preconditions: (ONFLOOR) A 3x(INR00M (ROBOT x) A INROOM(object x)), to the state intermedtary-state: ONFLOOR A INROOM(R0B0T B) A INR00M(B0X B). DISPROVER suggests to LAWALY that the original problem might be solved by splitting It up into two successive problems. The first problem ie 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. IAWALY 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) A INROOM (ROBOT B). Again, for essentially the same reasons *as* before, LAWALY fails to solve the problem. DISPROVER cannot find a disproof (of course, since none exists), but suggests the intermediary-state:
INROOM(R0BOT B) A NEXTT0(ROB0T BOX). Once more, LAWALY fails, again due to her stubbornness in insisting on finishing a subtask completely before starting another one. DISPROVER finds no disproof (rightly so, since none exists) with the anchor predicates:
INRO0M(R0BOT BOX) NEXTT0(R0BOT BOX) ON(R0B0T BOX). Moreover, DISPROVER suggests the same intermediary-state as before, hence the system would begin to repeat itself, and failure is accepted.

We are investigating the possibility of LAWALY helping DISPROVER by communicating information on why she failed to find a path, thereby helping DISPROVER *to* build a more adequate set of anchor predicates. The results are still sketchy and will not be discussed here.

9. The Importance of Axiomatlzation.

The problem of section ? 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:
imOQM(ROBOT 3) 0N(ROBOT BOX) INROOM(BOX *S).* If *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 e model. We give examples of collaboration between DISPROVER and a powerful robot planning system, LAWALY. The axiomatization chosen for the world greatly influences the performance of the described systems.

11. References.

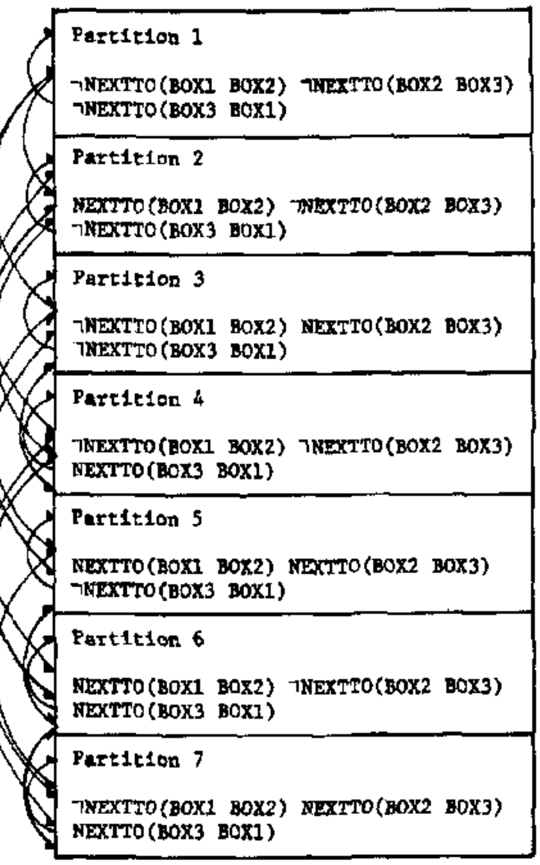
[1]Fikes, R. E. and Nilson, N. J. "STRIPS: A New Approach to the Application of Theorem Proving in Problem Solving," Artificial Intelligence. *2,* 189-208, 1971.

[2]Review of [l]. Computing Reviews. _13, 5, 216-217, 1972.

Slklossy, 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 Stklossy, L. (Eds.) Representation and Meaning: Experiments with Information Processing Systems. Prentice-Hall, Englewood Cliffs, N.J., 1972.

Slklossy, L. and Dreussi, J. A Hierarchy-Driven Robot Planner which Generates its own Procedures. Technical Report 11, Computer Sciences Dept., University of Texas Austin, 1973.



Partition 1
¬NEXTTO(BOX1 BOX2) ¬NEXTTO(BOX2 BOX3) ¬NEXTTO(BOX3 BOX1)
Partition 2
NEXTTO(BOX1 BOX2) ¬NEXTTO(BOX2 BOX3) ¬NEXTTO(BOX3 BOX1)
Partition 3
¬NEXTTO(BOX1 BOX2) NEXTTO(BOX2 BOX3) ¬NEXTTO(BOX3 BOX1)
Partition 4
¬NEXTTO(BOX1 BOX2) ¬NEXTTO(BOX2 BOX3) NEXTTO(BOX3 BOX1)
Partition 5
NEXTTO(BOX1 BOX2) NEXTTO(BOX2 BOX3) ¬NEXTTO(BOX3 BOX1)
Partition 6
NEXTTO(BOX1 BOX2) ¬NEXTTO(BOX2 BOX3) NEXTTO(BOX3 BOX1)
Partition 7
¬NEXTTO(BOX1 BOX2) NEXTTO(BOX2 BOX3) NEXTTO(BOX3 BOX1)

*Figure 1. Disproof of NEXTT0(B0Xl BOX2) * NEXTT0(B0X2 BOX3) * NEXYT0(B0X3 B0X1).*

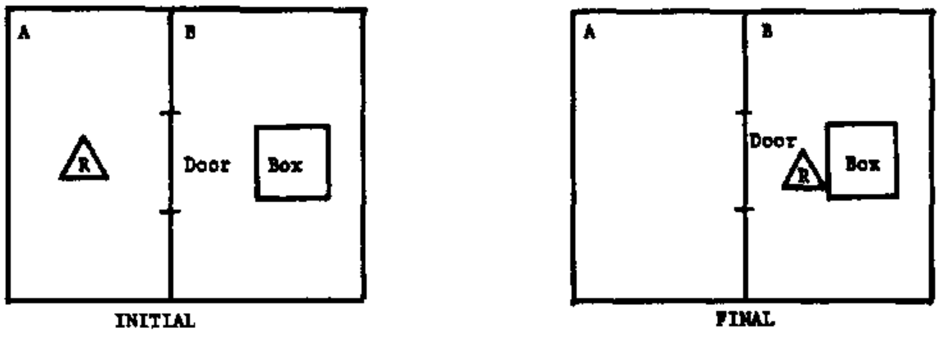


Figure 2.

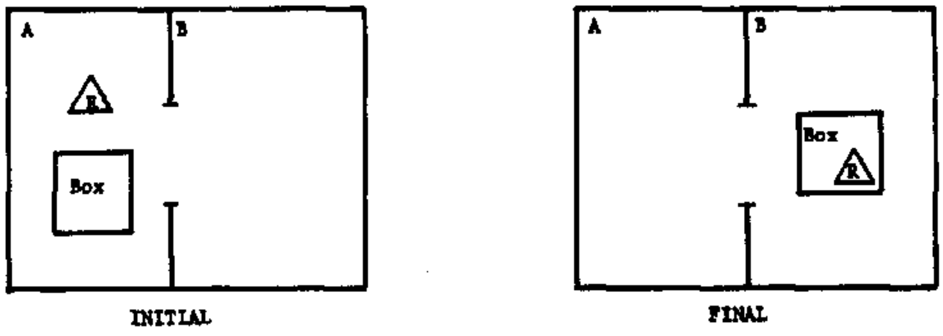


Figure 3.