MODEL VERIFICATION AND IMPROVEMENT USING DISPROVER*

by

L. Siklóssy and J. Roach

July 1973 TR-26

The Department of Computer Sciences

University of Texas at Austin

* Work partially supported by NSF Grant GJ-34736.

Abstract

Confidence in the adequacy of a model is increased if tasks that are impossible in the world are shown to correspond to disprovable tasks in the model. DISPROVER has been used as a tool to test, in worlds of robots, the impossibility of tasks related to various conservation laws (objects, position, model consistency, etc.) and time constraints. The adequacy and sufficiency of operators can be established. Interacting with DISPROVER, the model designer can improve his axiomatization.

The frontier between "acceptable" and "ridiculous" axiomatizations is shown, in many examples, to be a most tenuous one.

1. Introduction: the Need for Model Verification.

The simulation of mobile robots has been a continuing area of interest [1-11]. Very early it was noticed [6] that robot simulation and problem-solving systems for robots were not trivial. In particular, the experience of other groups, as well as our own, indicates that errors have an uncanny way of creeping into the models proposed to simulate particular worlds. The purpose of this paper is to describe a technique --that of disproofs-- implemented in a computer program, DISPROVER [8], that will help to verify the adequacy of a proposed model for robot-like worlds. Moreover, DISPROVER can be used to improve the proposed axiomatizations for a model.

Our main line of thought, which will be exemplified several times, is that the frontier between "acceptable" and "ridiculous" axiomatizations of the world is a very tenuous one. What may appear as "insignificant" changes in an axiomatization may make it impossible to accomplish a task that is feasible in the real world, while some other "insignificant" changes may render possible the accomplishment of a task which is impossible in the world.

Let us assume that someone proposes an axiomatization AX for some world W. (An appropriate problem-solver is associated with AX.) How are we to judge of the adequacy of AX? AX will be all the more credible if a variety of problems can be easily formulated and solved in AX, such that the solutions obtained correspond to solutions that we expect to obtain in W. But testing a model by giving it only solvable tasks to solve is insufficient. Taking an extreme position, let us suppose that all problems, meaningful or not, solvable or not, have a solution in AX. Then AX would be inacceptable as a model, since it would give solutions to problems which we know have no solution in W. Hence, to show the adequacy of AX it is necessary to demonstrate that a variety of problems that are impossible in W can be easily formulated and shown to have no solution in AX. The model becomes highly credible if:

a) problems that have, or that we expect or suspect to have, solutions in W can be solved in AX, in such a way that the solution in AX satisfies our knowledge of or intuition about solutions in W;

b) problems that have, or that we expect or suspect to have, no solution in W can be disproved in AX, i.e. shown to have no solution.

We can say that AX is a perfect model of W if any problem is solvable in W if and only if it is solvable in AX, or impossible in W if and only if it is disprovable in AX. If W is the real physical world, then a perfect model does not appear achievable. At best, we can increase our confidence in a model by gaining experience with solutions to solvable or disproofs to impossible tasks.

In the next two sections we review some axiomatizations that have been used for robot worlds and briefly describe DISPROVER. In the subsequent sections we give examples of almost identical axiomatizations which, in one case lead to acceptable results, while in the other they violate some conservation law in W, or lead to results that are inacceptable for other reasons. Finally, we show how DISPROVER can be used to improve already "adequate" axiomatizations.

2. Axiomatizations of the World.

The set theory cum predicate calculus axiomatization introduced in [1] has been popular in axiomatizing robot worlds. It is briefly reviewed here.

The world is described as a set of true facts that hold in the world, for example:

(NEXTTO ROBOT BOX2) (STATUS PAIL3 FULL) (CLOSED DOORAB), etc.

Changes in the world are the result of applications of operators to the world. An operator, for example goto-object(object), is applicable to a world WORLD1, if the preconditions of goto-object are satisfied in WORLD1. One axiomatization of goto-object* would have preconditions: (ONFLOOR) (INROOM ROBOT place) (INROOM object place), where object is the argument of goto-object, and place a free variable that must be bound.

After applying an operator, the old world is changed into a new world by use of the add set and delete set of the operator:
new-world:= ((old-world - delete-set-of-op) + add-set-of-op) where "-" and "+" are set difference and union.
An axiomatization of the add-set and delete-set of goto-object could be:
Delete-set: (ATROBOT $) (NEXTTO ROBOT $);
Add-set: (NEXTTO ROBOT object);
where "$" is a universal variable that will match anything.

A task is specified as a set GOAL of conditions that the world must satisfy. The goal is achieved if we find a sequence of operators op_1, op_2, ... , op_n, which transform the initial state of the world into

*A restriction of goto2 in [1], called goto2a in [6].

a world that contains GOAL as a subset.

The above type of axiomatization is far from perfect. For the purposes of this work, we have extended the axiomatization in the following directions:

a) conditional statements and functions are allowed in the precondition, add and delete sets.

b) the bindings of free variables can be forced, or can result from computation.

However, the world is still a set of true facts.

Hendrix [10] has proposed an axiomatization that incorporates explicit time references and spontaneous activities (such as the growing of grass), while Siklóssy and Dreussi [11] allow negative facts --such as: BOX2 is not in ROOM1-- to be present in the description of worlds, preconditions and goals.

The robot problem-solver used in this study is a modification of LAWALY [7] that incorporates the extensions a) and b) above to the axiomatization. It will not be discussed here, since our main emphasis is on DISPROVER, which will be briefly sketched in the next section.

3. Sketch of DISPROVER.

DISPROVER is an implementation of the technique of hereditary partitions [8] to disprove statements about robot-like worlds. DISPROVER can be applied to find disproofs in any system which describes successive states by a set of true and false conditions, independently of the form of the operators that transform one state into another.

An example will help to illustrate the working of DISPROVER. We take the world of Figure 1 (as used in [1,6,7]) and change it so that if the robot goes through the door to the room containing the location G, then the light turns off. To turn the LIGHTSWITCH1 on, the robot must be (ON ROBOT BOX1) and the BOX1 must be (NEXTTO BOX1 LIGHTSWITCH1). We wish to show that the task:

(STATUS LIGHTSWITCH1 ON) (ATROBOT G), is impossible, i.e. we wish a disproof of this task.

DISPROVER uses two sets of predicates for disproofs:

a) anchor predicates, which are always the predicates (made positive if necessary) in the statement of the problem to be disproved. Here the anchor predicates are P1 = (STATUS LIGHTSWITCH1 ON); P2 = (AT ROBOT G).

b) refinement predicates, which are additional predicates used in the disproof. In a particular disproof (see section 9), the following refinement predicates were used:

P3 = (INROOM ROBOT ROOMD); P4 = (INROOM ROBOT ROOMA); P5 = (ON ROBOT BOX1); P6 = (ONFLOOR); P7 = (NEXTTO ROBOT BOX1).

The union of anchor and refinement predicates is the set of relevant predicates.

We are developing automatic ways of generating refinement predicates. For the purposes of this work, it will be assumed that the refinement predicates are furnished by the model designer.

The partitions generated by DISPROVER are described by the presence or absence of each relevant predicate in the partition. The initial partition is obtained by intersecting the set of relevant predicates and their negatives with the initial state of the world. In our example, the initial partition is: -P1 -P2 -P3 P4 -P5 P6 -P7.

From the initial partition, all relevant operators to the problem --that is, which can modify one of the relevant predicates-- are applied to generate new partitions. The disproof terminates if either:

a) the goal set is a subset of one of the partitions; or

b) no new partitions can be generated.

In case a), the disproof fails; while in case b) the disproof succeeds: the goal is not achievable, since the partitions are closed under all legal operators, and the goal is not a subset of any partition.

It should be noted that DISPROVER allows initial, intermediary and final states to be specified using the negation of predicates. For example, we could have tried to disprove:

(STATUS LIGHTSWITCH1 ON) ¬(AT ROBOT G); however the disproof would have failed (case a) above), and indeed the goal state thus specified is achievable. In general, though, the failure of a disproof does not imply that the goal state represents a solvable problem. DISPROVER always terminates, since the maximum number of partitions that can be generated is: $N = 2^{(\text{number of relevant predicates})}$. In successful disproofs, the number of partitions generated is at most (N-1), and in actual practice, whether DISPROVER succeeds or not, it usually generates far fewer partitions than

N. The disproof of the above example was obtained using only 7 partitions.

We now turn our attention to applications of DISPROVER to the verification of the adequacy of models.

4. Adequacy Verification: Conservation Tests.

DISPROVER makes it possible to attempt to verify that a goal is impossible. We shall concentrate on _negative tests_ of a model, that is, on interesting problems that should not be solvable in the model. By contrast, _positive tests_ would be (hard) problems that should be solvable in the model. Positive tests will be ignored here.

A large class of interesting negative tests can be categorized under the label of _conservation tests_. Similarly, in physics, many conservation laws hold: conservation of energy, of momentum, of parity, of charge, etc. (at least in many physical systems.)

4.1 Conservation of Objects.

Unless specifically permitted, objects cannot appear or disappear in the world.* For example, in the worlds of [1, 6, 7], we can disprove (NEXTTO ROBOT BOX5), where BOX5 is not in the world. The disproof took 2.3 seconds in interpreted LISP on the CDC 6600.

Returning to the axiomatization of goto-object (section 2), one could have argued cleverly as follows: the boxes cannot move out of ROOMA, hence the test (INROOM object place) is superfluous. Furthermore, since the only case when the robot can go next to an object is when she is in ROOMA, we could replace the precondition (INROOM ROBOT place) by (INROOM ROBOT ROOMA). The precondition set of goto-object would become:

(ONFLOOR) (INROOM ROBOT ROOMA).

However, the task (NEXTTO ROBOT BOX5) is now solvable by LAWALY [7] in 0.3 seconds (interpreted LISP on the CDC 6600). For that matter she could also be (NEXTTO ROBOT RAQUEL-WELCH), whether RAQUEL-WELCH is a box or not.

*We assume the world fully known to the robot.

4.2 Conservation of Position.

The robot and objects in its world cannot be simultaneously in two positions at the same time. We give two examples of models where in fact the conservation of position could not be proved --since it could be violated-- and how minimal changes in the axiomatization permitted the conservation to be verified using DISPROVER.

4.2.1 Robot next to two boxes at the same time.

In [1], a box could be pushed next to another box by means of the operator:

push(object1 object2);

Preconditions: (PUSHABLE object1) (ONFLOOR) (NEXTTO ROBOT object1) (INROOM object1 place) (INROOM object2 place);

Delete set: (AT ROBOT $) (AT object1 $) (NEXTTO ROBOT $) (NEXTTO object1 $) (NEXTTO $ object1);

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

The goal (NEXTTO ROBOT BOX1) (NEXTTO BOX2 ROBOT) --where BOX1 and BOX2 need not be next to each other-- is achievable, and is solved by LAWALY in 3.1 seconds [7]. The same goal becomes impossible if (NEXTTO $ ROBOT) is added to the delete sets of goto-object and push(object1 object2). The disproof took 3.4 seconds.

4.2.2 Box blocking two different doors at the same time.

In [2], a block operator was introduced. Its axiomatization was:

block (door room box);
Preconditions: (INROOM ROBOT room) (INROOM box room)
(PUSHABLE box) (UNBLOCKED door room)
(JOINSROOM door room room2);

Delete set: (AT ROBOT \$) (AT BOX \$)
(UNBLOCKED door room) (NEXTTO ROBOT \$)
(NEXTTO box \$) (NEXTTO \$ box);
Add set: (BLOCKED door room box) (NEXTTO ROBOT box).

It is then possible to block two (and more) doors with the same box. Blocking two doors took 6.9 seconds [7]. If the axiomatization of block is changed, and we add (BLOCKED \$ \$ box) to its delete set, the same task is disproved in 0.7 seconds.

4.2.3 Box in two different places at the same time.

Extending the above axiomatizations to allow boxes to be pushed to locations, we disproved the statement: (AT BOX1 LOC1) (AT BOX1 LOC2) in 0.5 seconds.

It should be noticed that a variety of predicates are used to indicate the position of the robot. She can be AT a location, ON a box, NEXTTO a door, IN a room, ON the FLOOR. Similarly, boxes could be AT a location, BLOCKing a door, NEXTTO another box, etc. The variety of these predicates makes it difficult to check the conservation of position in the model. Hence, other axiomatizations should be investigated. Although we shall come back to this point, we can already draw a lesson from the above examples: the axiomatization should make it easy to disprove negative tests.

4.3 Conservation of model consistency.

When the world includes mutually exclusive predicates, interesting negative tests are immediate. For example, a door must be open or closed [12], a light must be on or off (but not both), etc. In the world of [1, 6, 7], a disproof to the second statement is found in 0.6 seconds.

4.4 Conservation of asymmetry.

Certain properties of the world are asymmetric. For example, (ON BOX1 BOX2) and (ON BOX2 BOX1) cannot be true at the same time. We have not experimented with this type of conservation, although it might be a good negative test in some worlds.

4.5 Conservation of identity.

A box cannot become a door, and a certain door may not initially connect ROOMA and ROOMB, and later ROOMC and ROOMD. This conservation of identity appears difficult to violate in our experience.

5. Deadlock Conditions.

Besides conservation tests, interesting negative tests for models of the world include tests of deadlocks, of the necessity of operators and of time constraints.

A deadlock arises when resources are needed to continue a task, but none of these resources are available. When deadlock occurs, a task cannot be achieved, so one might be able to disprove the task. Sometimes a small change in resources makes it possible to break the deadlock. An example will illustrate the problem.

Suppose that a book is on a bookshelf, and the robot needs to climb on a ladder to reach the book. One ladder, LADDER1, is complicated and the book is needed for instructions to climb LADDER1. The task given to the robot is to have climbed LADDER1.

If LADDER1 is the only ladder in the world, the task is disproved in 0.6 seconds. If a second ladder exists in the world, a solution is found in 4.8 seconds. The solution is time-consuming since first the robot planned to carry LADDER1 to the bookshelf. However, she could not climb LADDER1, missing the book; so she jumped back in the past of the plan, and planned to carry the second ladder to the bookcase. From then on no difficulties are encountered.

The set axiomatization used for robot worlds is weak in arithmetic capabilities, and impossible tasks based on a more accurate count of resources --such as those described by Simon [13]-- are not presently within its capabilities.

6. Necessity of Operators.

In a model, operators are sometimes introduced to accomplish certain subgoals. To test the adequacy of an operator, we wish to:

a) accomplish examples of the subgoals with the help of the operator;

b) disprove examples of the subgoals if the operator is removed from the set of operators available to DISPROVER.

For example, in LAWALY's world [7] a dirty room cannot be made clean unless it is swept.

7. Time Constraints.

Although time is not explicitly represented in the representation of the world used, it is represented implicitly. If WORLDJ is obtained from WORLDI by the application of one or more operators, then WORLDJ has an implicit time label greater than WORLDI. A number of interesting negative tests can be built around requirements for time sequences. One example occurred to us when axiomatizing a robot which carried boxes. After she went through a door from ROOMA to ROOMB, the box she was carrying was still in ROOMA!

Let us assume that, by the use of DISPROVER described in section 6, we have assured ourselves that the operator putdown(object) is necessary to put down an object held by the robot. It now suffices to disprove the goal:
(INROOM ROBOT ROOMB) (INROOM BOX ROOMA), from the initial world:
(INROOM ROBOT ROOMA) (INROOM BOX ROOMA) (HOLDING BOX), where DISPROVER is not given putdown as an operator.

8. Local Actions.

Many actions by the robot are possible only if the robot is in close proximity to the objects upon which she is going to act. For example, she cannot push a box unless she is close to it; she cannot turn a light on unless she is near the corresponding lightswitch. The border which separates apparently correct axiomatizations from faulty ones, in which actions are performed from a distance, is very thin. Let us illustrate by an example.

In [1, 6, 7] the turnonlight(lightswitch) operator is axiomatized as:
Preconditions: (TYPE lightswitch LIGHTSWITCH) (ON ROBOT BOX1)
(NEXTTO BOX1 lightswitch);
Delete set: (STATUS lightswitch OFF);
Add set: (STATUS lightswitch ON).

Let us assume that the lightswitch is triggered as soon as the robot is high enough, that is after it has climbed on BOX1.* In this way, the precondition (NEXTTO BOX1 lightswitch) is no longer necessary. But then, if BOX1 happens to be in another room, for instance ROOMD, nothing prevents the robot from turning the light on from that ROOMD!

9. Modelling Criteria and Local Adequacy.

Even though a particular model appears adequate in the sense that appropriate negative tests have been disproved, a consideration of the disproofs may lead the designer to further improve the axiomatization. Let us consider an example.

The impossible task described in section 3:
(STATUS LIGHTSWITCH1 ON) (AT ROBOT G), cannot be disproved using only the two anchor predicates. Let us demonstrate how the disproof fails: from the initial

*Devices using photoelectric cells could make this arrangement quite feasible.

partition -P1 -P2, the goto operator permits a transition to the partition -P1 P2. The fact that the robot is at G does not violate the preconditions of the operator turnonlight; hence we pass to the partition P1 P2, which contains the goal that we are trying to disprove. Hence the disproof fails.

With the axiomatization of [1, 6, 7], the refinement predicates P3, P4, P5 and P7 appear necessary for a disproof. Intuitively though, we feel that the reason that the two tasks cannot be achieved simultaneously is that the robot must be in ROOMD to go to G, and as soon as she enters ROOMD the light is off, and cannot be turned on without leaving ROOMD. The disproof should have nothing to do with the robot's being on BOX1 or not, etc! Hence, what appears crucial is information about the two rooms, ROOMA and ROOMD. Indeed, a shorter disproof can be obtained using only the refinement predicates (INROOM ROBOT ROOMD) (INROOM ROBOT ROOMA) <u>if</u> the axiomatization of turnonlight is changed: the preconditions set of turnonlight must be augmented with (INROOM lightswitch room) (INROOM ROBOT room).

The above example shows an improvement in the axiomatization resulting from a successful, yet intuitively unsatisfactory, disproof. Unsuccessful disproofs can also help the model designer. When a disproof fails, he is given indications of the weaknesses of his axiomatization. Returning to the above unsuccessful disproof without refinement predicates, the designer can see that the disproof fails because in a world reduced to a description in terms of only P1 and P2, nothing prevents the robot from turning the light on from G. His knowledge of the physical world shows the fallacy of such an action, and he may be led to add the refinement predicate P3 = (INROOM ROBOT ROOMD), and require turnonlight to include the two added preconditions we have just mentioned. As DISPROVER is re-entered, the disproof fails again: from the partition ¬P1 ¬P2 P3, she can turn the light on. Indeed, the preconditions to turnonlight are not contradicted by the partition, since the precondition (INROOM ROBOT ROOMA) is not contradicted by P3 = (INROOM ROBOT ROOMD); it could only be contradicted by ¬(INROOM ROBOT ROOMA), which is not known in the partition. As before, the designer would be led to add the refinement predicate (INROOM ROBOT ROOMA).

10. Conclusions.

To increase our confidence in the adequacy of a model, a variety of problems solvable in the world should be solvable in the model, and a variety of impossible goals in the world should be disprovable in the model. DISPROVER is a tool that can be used to verify that impossible goals in the world do indeed correspond to disprovable goals in the model.

Among the many impossible goals that should be disproved in a model of a mobile robot, we have given examples of tests involving conservations --of objects, position, model consistency, asymmetry and identity--, cases of deadlock, and time constraints. DISPROVER can also help verify that operators really do what they are supposed to do, and no more.

As DISPROVER generates, or fails to generate, disproofs, the designer of the model may be led to modify his axiomatization. When DISPROVER fails, it indicates the possibility of transitions (between states) which, although undesirable, have been overlooked. Even a successful disproof may be intuitively unsatisfactory if it requires details peripheral to the designer's intuition of the situation.

The use of DISPROVER was exemplified with the help of a variety of very simple examples. Even in such simple examples, the designer's intuition was often found at fault. Perhaps he is not to be blamed, since the dividing line is most thin between "correct" axiomatizations and "ridiculous" axiomatizations that permit the accomplishment of forbidden tasks.

11. References.

1. Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, 2, 3/4, 189-208, 1971.

2. Raphael, B. et al., Research and Applications-- Artificial Intelligence, Technical Report, SRI Project 8973, Stanford Research Institute, Menlo Park, California, December 1971.

3. Fikes, R. E., Hart, P. E. and Nilsson, N. J., "Learning and Executing Generalized Robot Plans," Artificial Intelligence, 3, 4, 251-288, 1972.

4. Fikes, R. E., Hart, P. E. and Nilsson, N. J., "New Directions in Robot Problem Solving," Machine Intelligence 7, Michie, D. and Meltzer, B. (Eds.), Edinburgh University Press, Edinburgh, Great Britain, 1972.

5. Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," Third International Joint Conference on Artificial Intelligence, Palo Alto, California, 1973.

6. Siklóssy, L., Modelled Exploration by Robot. Technical Report TR-1, Computer Sciences Department; University of Texas, Austin, 1972.

7. Siklóssy, L. and Dreussi, J., "An Efficient Robot Planner which generates its own Procedures," Third International Joint Conference on Artificial Intelligence, Palo Alto, California, 1973.

8. Siklóssy, L and Roach, J., "Proving the Impossible is Impossible is Possible: Disproofs based on Hereditary Partitions," Third International Joint Conference on Artificial Intelligence, Palo Alto, California, 1973.

9. Siklóssy, L. and Dreussi, J., Simulation of Executing Robots in Uncertain Environments, Technical Report TR-16, Computer Sciences Department; Univer-

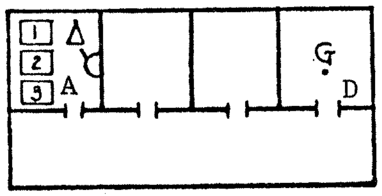
sity of Texas, Austin, May 1973.

10. Hendrix, G., Beyond Omnipotent Robots, Technical Report Nl-14, University of Texas, Austin, 1973. (To appear under the title "Modeling Simultaneous Actions and Continuous Processes," in _Artificial Intelligence Journal_.)

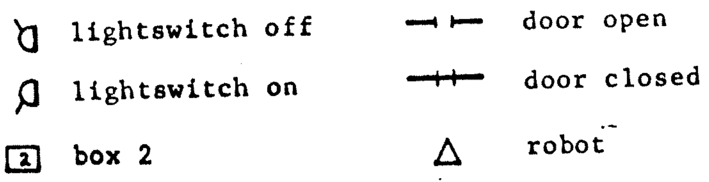
11. Siklóssy, L. and Dreussi, J., Robot problem-solving with negative goals, Technical Report TR-23, University of Texas, Austin, 1973.

12. Musset, Louis Charles Alfred de, "_Il faut qu'une Porte soit Ouverte ou Fermée_,: in _OEuvres Complètes_, Editions du Seuil, Paris, 1963.

13. Simon, H. A., "On Reasoning about Actions," in: Simon, H. A. and Siklóssy, L. (Eds.) _Representation and Meaning_, Prentice-Hall, Englewood Cliffs, N.J., 1972.



A Robot World



LEGEND

Figure 1.