Final Report

Covering the Period 8 October 1970 to 7 October 1971

RESEARCH AND APPLICATIONS—ARTIFICIAL INTELLIGENCE

By: B. RAPHAEL R. O. DUDA R. E. FIKES P. E. HART
N. J. NILSSON P. W. THORNDYKE B. M. WILBER

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
OFFICE OF ADVANCED RESEARCH AND TECHNOLOGY
RESEARCH DIVISION
WASHINGTON, D.C. 20546

CONTRACT NASW-2164

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

Final Report *December 1971*

Covering the Period 8 October 1970 to 7 October 1971

RESEARCH AND APPLICATIONS— ARTIFICIAL INTELLIGENCE

By: B. RAPHAEL R. O. DUDA R. E. FIKES P. E. HART
N. J. NILSSON P. W. THORNDYKE B. M. WILBER

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
OFFICE OF ADVANCED RESEARCH AND TECHNOLOGY
RESEARCH DIVISION
WASHINGTON, D.C. 20546

Attention: MR. JAMES ALBUS

CONTRACT NASW-2164

SRI Project 8973

Approved by:

DAVID R. BROWN, *Director*
Information Science Laboratory

BONNAR COX, *Executive Director*
Information Science and Engineering Division

Copy No. 18

ABSTRACT

This is the final report for the most recent year of a program of research in the field of artificial intelligence. The focus of recent work has been the design, implementation, and integration of a completely new system for the control of a robot that plans, learns, and carries out tasks autonomously in a real laboratory environment. The report includes sections that describe the computer implementation of low-level and intermediate-level actions; routines for automated vision; and the planning, generalization, and execution mechanisms. Section II of the report contains a scenario that demonstrates the approximate capabilities of the current version of the entire robot system.

CONTENTS

ABSTRACT iii

LIST OF ILLUSTRATIONS vii

LIST OF TABLES ix

I INTRODUCTION 1

A. General 1

B. Background 1

C. Report Outline 2

II OVERVIEW AND EXPERIMENTS 3

A. Overview 3

B. Experiments 5

III LOW-LEVEL ACTIONS 25

A. Introduction 25

B. Measurement and Control 26

C. The LLA Portion of Shakey's Model 29

D. The LLAs 30

E. Summary 33

IV INTERMEDIATE-LEVEL ACTIONS 37

A. Introduction 37

B. The Markov Language 37

C. The Actions 40

D. The Pathfinding Algorithm 46

V VISION ROUTINES 51

A. Introduction 51

B. Object Location 51

C. ILA Vision Routines 54

VI PLANNING, GENERALIZATION, AND EXECUTION. 55

A. Introduction. 55

B. Storage of a Generalized Plan 57

C. Planning with Generalized Plans 59

D. Monitoring the Execution of Plans 70

VII PUBLICATIONS AND PRESENTATIONS 75

A. Publications. 75

B. Presentations 77

REFERENCES 79

ILLUSTRATIONS

Figure 1 Map of Shakey's Experimental Environment 6

Figure 2 Map of Shakey's World after Completion of the First Task 21

Figure 3 Control Structure of Low-Level Activities. 35

Figure 4 Control Structure of the Intermediate Level. 42

Figure 5 Obstacle Configuration for FINDPATH. 48

Figure 6 Search Tree for Configuration of Figure 5. 48

Figure 7 Typical MACROP 57

Figure 8 MACROP with Marked Clauses 60

Figure 9 MACROP after Editing 62

Figure 10 Generalized Plan for Two-push MACROP 65

Figure 11 MOP: A Nested MACROP. 69

TABLES

Table 1 Axiom Model 8

Table 2 STRIPS Operators 13

Table 3 Triangle Table for MACROP1(PAR3,PAR1,PAR2,PAR4,PAR5, PAR7,PAR6) 17

Table 4 Triangle Table for MACROP2(PAR3,PAR1,PAR6,PAR7,PAR5, PAR4,PAR2) 23

Table 5 Low-level Activities of Robot 34

Table 6 Markov Table for the Lowest-level Pushing ILA 41

I INTRODUCTION

A. General

This is the final report for the most recent year of a program of research in the field of artificial intelligence. The work reported here began in October 1970 as a direct continuation of work performed under a previous contract.[1]* The work is currently being continued under new support.† Therefore this is a report on the recent accomplishments and status of a continuing research program.

A Semiannual Progress Report[2] was prepared in April 1971 that describes activities during the first six months of this project. This present report therefore emphasizes more recent work and is designed to augment and update, rather than replace, the Semiannual Report.

B. Background

For many years our work has been focused on the application of techniques of artificial intelligence to the control of a mobile automaton--a "robot"--in a realistic laboratory environment. This work reached a plateau in 1969 with the completion of the first integrated robot system, consisting of a mobile vehicle, TV camera, and other sensors, and a set of programs (on an SDS-940 computer) enabling the system to understand, solve, and physically perform a few simple but significant tasks.

---

*References may be found at the end of this report.

†Contract DAHC04-72-C-0008 with the Army Research Office and ARPA.

For the past two years we have been developing a new, more powerful robot system. The robot vehicle itself has remained largely intact. However, the other significant components of the system are completely new. The computer has been replaced by a PDP-10/PDP-15 facility, with considerably more capability than the old SDS-940. The software has been redesigned from top to bottom. This new design includes a library of basic action operators; a new general-purpose, problem-solving system; a new executive for monitoring the progress of the system; special visual perception routines to be coordinated with the problem-solving process; a method for generalization learning; and various error detection and recovery mechanisms.

From late 1969 until early 1971, our principal activities were assembling the new computer hardware and studying, designing, and implementing major components of the new system's software. The basic organization of the system was well established and its major components described in detail in our Semiannual Report last April.[2] Since then we have been fitting together the pieces, filling gaps, and completing the design, as well as continuing some of the separate research studies that grow out of robot system work.

C. Report Outline

Section II of this report presents an overview of the new robot system, and then describes some experiments that will demonstrate the principal new capabilities. Technical details of the four major system components that make these experiments feasible are contained in Sections III, IV, V, and VI. Finally, Section VII lists publications and presentations that were prepared or presented during the project period.

II OVERVIEW AND EXPERIMENTS

A. Overview

The robot system is a hierarchical structure in which we shall identify five major levels. Although some of these levels are much more clearly defined than others and some have considerable substructure, the five levels described below constitute a useful division for this exposition. Also, the effectiveness of the system is largely derived from the clear specifications for these levels and their interconnections.

The bottom level of the system consists of the robot vehicle and its connection to the user programs. This connection includes radio and microwave communication links, a PDP-15 peripheral computer and its software, and a communications channel, with its associated software, between the PDP-15 and the PDP-10. This bottom level may be thought of as defining the elementary physical capabilities of the system. The vehicle itself was described in several reports of previous projects, and the PDP-15/PDP-10 interface is described in Appendix G to the Semiannual Report.[2]

The second level consists of what we call Low-Level Actions, or "LLAs." These are the lowest-level robot control programs available to user programs in the LISP language, our principal programming tool. The LLAs are programatic handles on the robot's physical capabilities such as "ROLL" and "TILT." They are described in detail in Section III.

So that it can exhibit interesting behavior, our robot system has been equipped with a library of Intermediate-Level Actions, or "ILAs." These third-level elements are preprogrammed packages of LLAs, embedded in a Markov table framework with various control and error-correction features. Each ILA represents built-in expertise in some significant physical capability, such as "PUSH" or "GO TO." The ILAs might be thought of as instructive abilities of the robot, analogous to such built-in complex animal abilities as "WALK" or "EAT." Section IV contains a description of the present set of ILAs, along with the conditions under which they are applicable and how they each can affect the state of the world.

The principal sensor of the perceptual system is the TV camera. Programs for processing picture data have been restricted to a few special "vision" routines, which are incorporated into the system at either the ILA or LLA level. The algorithms in these routines are described in Section V.

Above the ILAs we have the fourth level, which is concerned with planning the solutions to problems. The basic planning mechanism is STRIPS, the problem solver described in Appendix C of the Semiannual Report and in Ref. 3. STRIPS constructs sequences of ILAs needed to carry out specified tasks. Such a sequence, along with its expected effects, can be represented by a triangular table called a MACROP ("macro operation"). Section VI describes how such MACROPs can be generated in generalized form, thereby enabling an interesting form of learning to take place.

Finally, the fifth, or top, level of the system is the executive, the program that actually invokes and monitors executions of the ILAs specified in a MACROP. The current executive program, called PLANEX, is briefly described at the end of Section VI. Some additional information about the PLANEX design may be found in Ref. 4.

B. Experiments

In this section we shall describe some experiments now being planned that will illustrate several features of the robot system, which we call, informally, "Shakey." Specifically these will show how Shakey generates a plan to perform a task, and how it then uses part of this plan later as a component of a plan for performing another task. Saving plans for later use might be regarded as a form of learning. The experiments also show how the various levels in Shakey's hierarchical control structure function to enable Skakey to recover gracefully from several kinds of unexpected failures.

1. Shakey's World and Model

We must first describe the environment in which Shakey operates and Shakey's model of this environment. In Figure 1, we show a floor plan of some rooms and doorways in which our experiments with Shakey will be conducted. We can place several large boxes and wedge-shaped objects in these rooms; three boxes are depicted in room RCLK* of Figure 1. Initially Shakey is in room RUNI. The doorways all have mnemonic names indicating the rooms they connect; e.g., DMYSPDP connects RMYS and RPDP.

Shakey's model of this environment is represented by a set of formulas or axioms in the first-order predicate calculus. The rooms,

---

*The room names are mnemomics for properties of the physical environment:

RHAL	=	Hallway
RRIL	=	Rilla's office
RCLK	=	Room with the clock on the wall
RRAM	=	Room with ramp to hallway
RPDP	=	PDP-10 room
RUNI	=	Unimate room
RMYS	=	Mystery room, i.e., room with unknown contents.

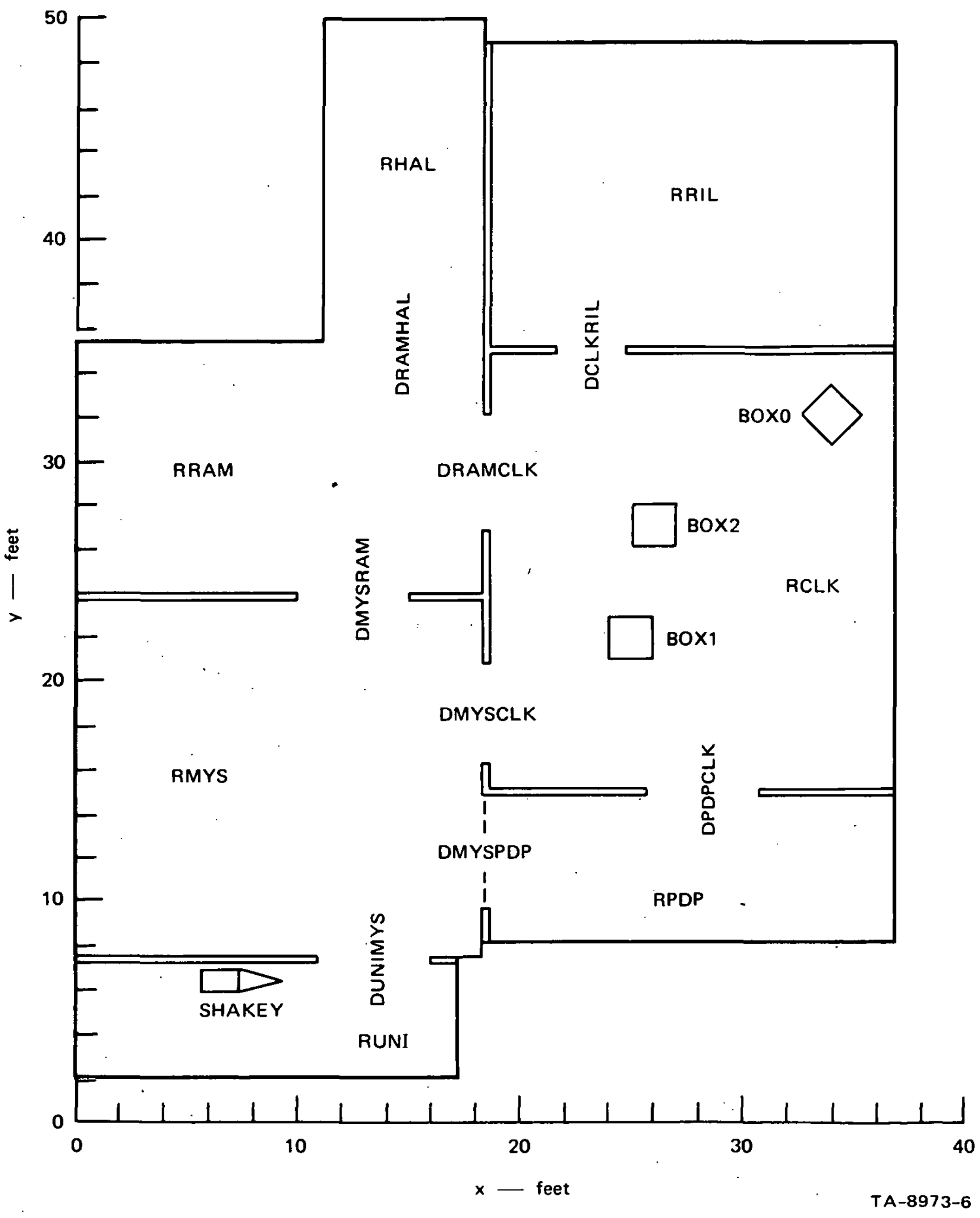


FIGURE 1 MAP OF SHAKEY'S EXPERIMENTAL ENVIRONMENT

doorways, boxes, walls, and other entities occur as terms in formulas that describe important properties of the environment. The axiom model representing the environment from planned experiments is listed in Table 1.

The meanings of most of the predicate symbols are obvious. AT gives coordinate location information referenced to the coordinate system of Figure 1. DAT gives information about the probable error in this coordinate information. The RADIUS predicate is used to give rough size information. THETA and DTHETA give information about Shakey's heading and probable heading error, respectively. The UNBLOCKED predicate tells which doorways are unblocked (i.e., free of obstructing objects such as boxes). The predicate ROOMSTATUS is used to tell whether the contents of a room are known or unknown. The model listed in Table 1 indicates that the contents of all rooms are assumed to be known except for RMYS. By this we mean that Shakey knows that he will never encounter any new objects except perhaps in RMYS. This knowledge is used to guide certain picture-taking behavior, as we shall see later. The LANDMARKS predicate gives the locations of various landmarks such as corners and doorjambs that Shakey can take pictures of to update its position. The axioms at the end of the model in Table 1 (beginning with the predicate WHISKERS) give information about the status of various lower-level motor and sensing activities, e.g., the status of the catwhisker switches and camera control settings. These are further explained in Section III-B.

Altogether there are 170 axioms in the model initially, which makes this model quite large in comparison with those used by any previous automatic problem-solving systems.

2. Shakey's Action Repertoire

In order to perform the tasks described below, Shakey has available a repertoire of ILAs. The operation of these ILAs is described

Table 1

AXIOM MODEL

```
AT(ROBOT,7,5)
DAT(ROBOT,0.1,0.1)
INROOM(ROBOT,RUNI)
AT(BOX0,34,32)
INROOM(BOX0,RCLK)
AT(BOX1,25,22)
INROOM(BOX1,RCLK)
AT(BOX2,26,27)
INROOM(BOX2,RCLK)
SHAPE(BOX0,BOX)
SHAPE(BOX1,BOX)
SHAPE(BOX2,BOX)
RADIUS(BOX0,1.7)
RADIUS(BOX1,1.5)
RADIUS(BOX2,1.5)
DAT(BOX0,0.1)
DAT(BOX1,0.1)
DAT(BOX2,0.1)
THETA(ROBOT,-90)
DTHETA(ROBOT,1)
PUSHABLE(BOX1)
PUSHABLE(BOX2)
UNBLOCKED(DRAMHAL,RHAL)
UNBLOCKED(DRAMHAL,RRAM)
UNBLOCKED(DCLKRIL,RRIL)
UNBLOCKED(DCLKRIL,RCLK)
UNBLOCKED(DRAMCLK,RCLK)
UNBLOCKED(DRAMCLK,RRAM)
UNBLOCKED(DMYSRAM,RMYS)
UNBLOCKED(DMYSRAM,RRAM)
UNBLOCKED(DMYSCLK,RCLK)
UNBLOCKED(DMYSCLK,RMYS)
UNBLOCKED(OPOPCLK,RCLK)
UNBLOCKED(DPDPCLK,RPDP)
UNBLOCKED(DMYSPDP,RPDP)
UNBLOCKED(DMYSPDP,RMYS)
UNBLOCKED(DUNIMYS,RMYS)
UNBLOCKED(DUNIMYS,RUNI)
BOUNDSROOM(FSRAM RRAM SOUTH)
BOUNDSROOM(FERAM RRAM EAST)
BOUNDSROOM(FWRAM RRAM WEST)
BOUNDSROOM(FNCLK RCLK NORTH)
BOUNDSROOM(FSCLK RCLK SOUTH)
BOUNDSROOM(FECLK RCLK EAST)
BOUNDSROOM(FWCLK RCLK WEST)
BOUNDSROOM(FNMYS RMYS NORTH)
BOUNDSROOM(FSMYS RMYS SOUTH)
BOUNDSROOM(FEMYS RMYS EAST)
BOUNDSROOM(FWMYS RMYS WEST)
BOUNDSROOM(FNPDP RPDP NORTH)
BOUNDSROOM(FSPDP RPDP SOUTH)
BOUNDSROOM(FEPDP RPDP EAST)
BOUNDSROOM(FWPDP RPDP WEST)
BOUNDSROOM(FNUNI RUNI NORTH)
BOUNDSROOM(FSUNI RUNI SOUTH)
BOUNDSROOM(FEUNI RUNI EAST)
BOUNDSROOM(FWUNI RUNI WEST)
FACELOC(FNHAL 50.0)
FACELOC(FSHAL 35.5)
FACELOC(FEHAL 18.200000)
FACELOC(FWHAL 11.200000)
FACELOC(FNRIL 49.0)
```

Table 1 (continued)

```
FACELOC(FSRIL 35.400000)
FACELOC(FERIL 36.800000)
FACELOC(FWRIL 18.799998)
FACELOC(FNRAM 35.5)
FACELOC(FSRAM 24.0)
FACELOC(FERAM 18.200000)
FACELOC(FWRAM 0.0)
FACELOC(FNCLK 35.0)
FACELOC(FSCLK 15.200000)
FACELOC(FECLK 36.800000)
FACELOC(FWCLK 18.599997)
FACELOC(FNMYS 23.599997)
FACELOC(FSMYS 7.6000000)
FACELOC(FEMYS 18.200000)
FACELOC(FWMYS 0.0)
FACELOC(FNPDP 14.799998)
FACELOC(FSPDP 8.2000000)
FACELOC(FEPDP 36.800000)
FACELOC(FWPDP 18.600000)
FACELOC(FNUNI 7.1999999)
FACELOC(FSUNI 2.1999998)
FACELOC(FEUNI 17.200000)
FACELOC(FWUNI 0.0)
JOINSROOMS(DRAMHAL RRAM RHAL)
JOINSROOMS(DRAMCLK RRAM RCLK)
JOINSROOMS(DCLKRIL RCLK RRIL)
JOINSROOMS(DRAMHAL RHAL RRAM)
JOINSROOMS(DRAMCLK RCLK RRAM)
JOINSROOMS(DCLKRIL RRIL RCLK)
TYPE(BOX1 OBJECT)
TYPE(BOX2 OBJECT)
TYPE(BOX0 OBJECT)
TYPE(RHAL ROOM)
TYPE(RRIL ROOM)
TYPE(RRAM ROOM)
TYPE(RCLK ROOM)
TYPE(RMYS ROOM)
TYPE(RPDP ROOM)
TYPE(RUNI ROOM)
TYPE(DRAMHAL DOOR)
TYPE(DRAMCLK DOOR)
TYPE(DCLKRIL DOOR)
TYPE(DMYSRAM DOOR)
TYPE(DMYSCLK DOOR)
TYPE(DMYSPDP DOOR)
TYPE(DPDPCLK DOOR)
TYPE(DUNIMYS DOOR)
BOUNDSROOM(FNHALL RHAL NORTH)
BOUNDSROOM(FSHAL RHAL SOUTH)
BOUNDSROOM(FEHAL RHAL EAST)
BOUNDSROOM(FWHAL RHAL WEST)
BOUNDSROOM(FNRIL RRIL NORTH)
BOUNDSROOM(FSRIL RRIL SOUTH)
BOUNDSROOM(FERIL RRIL EAST)
BOUNDSROOM(FWRIL RRIL WEST)
BOUNDSROOM(FNRAM RRAM NORTH)
JOINSROOMS(DMYSRAM RNYS RRAM)
JOINSROOMS(DMYSCLK RMYS RCLK)
JOINSROOMS(DMYSPDP RMYS RPDP)
JOINSROOMS(DPDPCLK RPDP RCLK)
JOINSROOMS(DUNIMYS RUNI RMYS)
JOINSFACES(DRAMHAL FNRAM FSHAL)
JOINSFACES(DRAMCLK FERAM FWCLK)
```

Table 1 (continued)

```
JOINSFACES(DCLKRIL FNCLK FSRIL)
JOINSFACES(DMYSRAM FNMYS FSRAM)
JOINSFACES(DMYSCLK FEMYS FWCLK)
JOINSFACES(DMYSPDP FEMYS FWPDP)
JOINSFACES(DPDPCLK FNPDP FSCLK)
JOINSFACES(DUNIMYS FNUNI FSMYS)
DOORLOCS(DRAMHAL 11.200000 18.200000)
DOORLOCS(DRAMCLK 26.799998 32.0)
DOORLOCS(DCLKRIL 21.700000 24.799998)
DOORLOCS(DMYSRAM 10.0 15.200000)
DOORLOCS(DMYSCLK 16.200000 20.799998)
DOORLOCS(DMYSRDP 9.7000000 14.799998)
DOORLOCS(DPDPCLK 25.799998 30.799998)
DOORLOCS(DUNIMYS 10.799998 16.0)
ROOMSTATUS(RHAL KNOWN)
ROOMSTATUS(RRIL KNOWN)
ROOMSTATUS(RRAM KNOWN)
ROOMSTATUS(RCLK KNOWN)
ROOMSTATUS(RMYS UNKNOWN)
ROOMSTATUS(RPDP KNOWN)
ROOMSTATUS(RUNI KNOWN)
LANDMARKS(RHAL (COORDS (4. 11.200000 35.5 0.)))
LANDMARKS(RRIL
         (COORDS (4. 21.700000 35.400000 -1.)
                 (3. 24.799998 35.400000 -1.)
                 (2. 18.799998 49.0 4.)
                 (2. 36.800000 49.0 3.)
                 (2. 36.800000 35.400000 2.)
                 (2. 18.799998 35.400000 1.)))
LANDMARKS(RRAN
         (COORDS (4. 18.200000 26.799998 0.)
                 (3. 18.200000 32.0 0.)
                 (1. 11.200000 35.5 2.)
                 (4. 10.0 24.0 -1.)
                 (3. 15.200000 24.0 -1.)
                 (2. 0.0 35.5 4.)
                 (2. 18.200000 24.0 2.)
                 (2. 0.0 24.0 1.)))
JOINSROOMS(DMYSRAM RRAM RMYS)
JOINSROOMS(DMYSCLK RCLK RMYS)
JOINSROOMS(DMYSPDP RPDP RMYS)
JOINSROOMS(DPDPCLK RPDP RCLK)
JOINSROOMS(DUNIMYS RUNI RMYS)
LANDMARKS(RCLK
         (COORDS (4. 24.799998 35.0 -1.)
                 (3. 21.700000 35.0 -1.)
                 (4. 25.799998 15.200000 -1.)
                 (3. 30.799998 15.200000 -1.)
                 (4. 18.599997 20.799998 0.)
                 (3. 18.599997 16.200000 0.)
                 (4. 18.599997 32.0 0.)
                 (3. 18.599997 26.799998 0.)
                 (2. 18.599997 35.0 4.)
                 (2. 36.800000 35.0 3.)
                 (2. 36.800000 15.200000 2.)
                 (2. 18.599997 15.200000 1.)))
LANDMARKS(RMYS
         (COORDS (4. 18.200000 9.7000000 4.)
                 (1. 18.200000 14.799998 1.)
                 (4. 18.200000 16.200000 0.)
                 (3. 18.200000 20.799998 0.)
                 (4. 15.200000 23.599997 -1.)
                 (3. 10.0 23.599997 -1.)
```

Table 1 (concluded)

```
                      (4. 10.799998 7.6000000 -1.)
                      (3. 16.000000 7.6000000 -1.)
                      (2. 0.0 23.599997 4.)
                      (2. 18.200000 23.599997 3.)
                      (2. 18.200000 7.6000000 2.)
                      (2. 0.0 7.6000000 1.)))
LANDMARKS(RPDP
          (COORDS (4. 30.799998 14.799998 -1.)
                  (3. 25.799998 14.799998 -1.)
                  (4. 18.200000 14.799998 -1.)
                  (3. 18.600000 9.7000000 0.)
                  (2. 36.800000 14.799998 3.)
                  (2. 36.800000 8.2000000 2.)))
LANDMARKS(RUNI
          (COORDS (4. 16.000000 7.1999999 -1.)
                  (3. 10.799998 7.1999999 -1.)
                  (2. 16.0 7.1999999 3.0)
                  (2. 17.200000 2.1999998 2.)
                  (2. 0.0 2.1999998 1.)))
WHISKERS(ROBOT,0)
IRIS(ROBOT,1)
OVERIDE(ROBOT,0)
RANGE(ROBOT,30)
TVMODE(ROBOT,0)
FOCUS(ROBOT,30)
PAN(ROBOT,0)
TILT(ROBOT,0)
DPAN(ROBOT,3.12)
DTILT(ROBOT,0.7)
DIRIS(ROBOT,0)
DFOCUS(ROBOT,0)
PICTURESTAKEN(ROBOT,0)
JUSTBUMPED(ROBOT,NIL)
```

in Section V. The problem-solving system, STRIPS, must be aware of the properties of the available ILAs. Therefore each ILA is represented for STRIPS by an operator with specified preconditions and effects. These operators and their descriptions are given in Table 2 using the add and delete lists employed by STRIPS.

We shall now describe the planned experiments that will use the model of Table 1 and the operators shown in Table 2. The description will be in terms of the expected results of these experiments.

a. Task 1

Starting with the configuration of Figure 1 (represented by the model in Table 1), Shakey will perform two tasks. Each of these tasks is stated in English and entered into the system via teletype. The first task is stated as "USE BOX 2 TO BLOCK DOOR DPDPCLK FROM ROOM RCLK." This statement is converted by the English language system ENGROB[5] to a goal expressed by a well-formed formula (wff) of the first-order predicate calculus: BLOCKED(DPDPCLK,RCLK,BOX2). The STRIPS problem-solving system is then called to compose a sequence of operators whose execution will create a world model in which this goal wff is true. In terms of the operators in Table 2, we can show that the following sequence would solve this problem:

GOTO2(DUNIMYS),GOTHRUDR(DUNIMYS,RUNI,RMYS),
GOTO2(DMYSCLK),
GOTHRUDR(DMYSCLK,RMYS,RCLK),
BLOCK(DPDPCLK,RCLK,BOX2) .

Rather than generating this specific solution, STRIPS generates a <u>generalized</u> plan that involves going from an arbitrary initial room through an intermediate room, and into a third room and then blocking a doorway in the third room. The rooms, doorways, and blocking object

Table 2

STRIPS OPERATORS

BLOCK(DX,RX,BX)

Preconditions:

```
INROOM(ROBOT,RX) ∧ INROOM(BX,RX)
∧ PUSHABLE(BX) ∧ UNBLOCKED(DX,RX)
∧ (∃RY)JOINSROOMS(DX,RX,RY)
```

Delete List:

```
AT(ROBOT,$1,$2)
AT(BX,$1,$2)
UNBLOCKED(DX,RX)
NEXTTO(ROBOT,$1)
NEXTTO(BX,$1)
NEXTTO($1,BX)
```

Add List:

```
*BLOCKED(DX,RX,BX)
 NEXTTO(ROBOT,BX)
```

Blocks door DX with an object BX by pushing BX to a place in room RX directly in front of door DX.

UNBLOCK(DX,RX,BX)

Preconditions:

```
BLOCKED(DX,RX,BX) ∧ INROOM(ROBOT,RX) ∧ PUSHABLE(BX)
```

Delete List:

```
AT(ROBOT,$1,$2)
BLOCKED(DX,RX,BX)
AT(BX,$1,$2)
NEXTTO(ROBOT,$1)
NEXTTO(BX,$1)
NEXTTO($1,BX)
```

Add List:

```
*UNBLOCKED(DX,RX)
 NEXTTO(ROBOT,BX)
```

Unblocks door DX by pushing object BX away from its place in room RX directly in front of door DX.

GOTHRUDR(DX,RX,RY)

Preconditions:

```
NEXTTO(ROBOT,DX) ∧ INROOM(ROBOT,RX)
∧ JOINSROOMS(DX,RX,RY) ∧ UNBLOCKED(DX,RX)
∧ UNBLOCKED(DX,RY)
```

Delete List:

```
AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
INROOM(ROBOT,$1)
```

Table 2 (continued)

Add List:

```
*INROOM(ROBOT,RY)
 NEXTTO(ROBOT,DX)
```

Takes Shakey through door DX from room RX into room RY.

GOTO2(X)

Preconditions:

```
(∃RX)[INROOM(ROBOT,RX) ∧ INROOM(X,RX)]
∨ (∃RX,RY)[INROOM(ROBOT,RX)
∧ JOINSROOMS(X,RX,RY) ∧ UNBLOCKED(X,RX)]
```

Delete List:

```
AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
```

Add List:

```
*NEXTTO(ROBOT,X)
```

Takes Shakey from any point in a room to a location next to any object or doorway, X, in the same room. (Shakey will navigate around obstacles that might be in the way of a direct path.)

PUSH(OB,X,Y)

Preconditions:

```
(∃RX)[INROOM(ROBOT,RX) ∧
INROOM(OB,RX) ∧ LOCINROOM(X,Y,RX)]
∧ PUSHABLE(OB)
```

Delete List:

```
AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
AT(OB,$1,$2)
NEXTTO(OB,$1)
NEXTTO($1,OB)
```

Add List:

```
*AT(OB,X,Y)
 NEXTTO(ROBOT,OB)
```

Pushes object OB from one point in a room to a coordinate location (X,Y) in the same room. (Shakey must initially be in the same room as OB and (X,Y), but will push OB around obstacles that might be in the way of a direct path.)

NAVTO(X,Y)

Preconditions:

```
(∃RX)[INROOM(ROBOT,RX)
∧ LOCINROOM(X,Y,RX)]
```

Table 2 (concluded)

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)

Add List:

*AT(ROBOT,X,Y)

Takes Shakey from any point in a room to the coordinate location (X,Y) in the same room. (Shakey will navigate around obstacles that might be in the way of a direct path.)

POINT(DIRECTION)

Preconditions:

none

Delete List:

THETA(ROBOT,$1)

Add List:

*THETA(ROBOT,DIRECTION)

Turns Shakey so that its heading is DIRECTION.

PUSH3(OB,X)

Preconditions:

$\text{PUSHABLE(OB)} \wedge \exists(\text{RX})\{\text{INROOM(ROBOT,RX)} \wedge \text{INROOM(OB,RX)}$
$\wedge [\text{INROOM(X,RX)} \vee \exists(\text{RY})\text{JOINSROOMS(X, RX,RY)}]\}$

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
AT(OB,$1,$2)
NEXTTO(OB,$1)
NEXTTO($1,OB)

Add List:

*NEXTTO(OB,X)
NEXTTO(ROBOT,OB)

Pushes object OB from one point in a room to a location next to any object or doorway X in the same room. (Shakey will push OB around obstacles that might be in the way of a direct path.)

[*] Note: An asterisk(*) in front of an add-list clause indicates that this clause is one of the "primary effects" of the operator.

in this generalized plan are represented by parameters. The generalized plan is thus a subroutine whose arguments are the parameters. These arguments are bound to specific constants only when the plan is executed. The value of the generalized subroutine is that it can be stored away (or "learned") and then used again in other situations perhaps as part of a plan for a more complex task. The way in which STRIPS produces these generalized plans is discussed in Section VI.

The task in question elicits the following generalized plan from STRIPS:

```
GOTO2(PAR6),GOTHRUDR(PAR6,PAR7,PAR5)
GOTO2(PAR4),GOTHRUDR(PAR4,PAR5,PAR2),
BLOCK(PAR1,PAR2,PAR3)    .
```

This plan is stored away as the macro operator:

```
MACROP1(PAR3,PAR1,PAR2,PAR4,PAR5,PAR7,PAR6)    .
```

STRIPS creates a triangle table representation of MACROP1. This table compactly stores information vital to monitoring the execution of MACROP1 and information needed to use MACROP1 (or parts of it) as a component of a future plan. We show this triangle table representation of MACROP1 in Table 3* and refer the reader to Section VI for a discussion of triangle tables and their uses.

After the creation of the triangle table representation of MACROP1, STRIPS prepares a version of it that will solve the given task, namely, to "Use BOX2 to block door DPDCLK from room RCLK." This version is obtained from MACROP1 by replacing those parameters standing

*Note: For all triangle tables, an asterisk (*) before a clause indicates that this clause was used to prove the preconditions of the operator named at the right of the row in which the clause appears.

Table 3

TRIANGLE TABLE FOR MACROP1(PAR3,PAR1,PAR2,PAR4,PAR5,PAR7,PAR6)

*UNBLOCKED(PAR6,PAR7)
*JOINSROOMS(PAR6,PAR7,PAR5)
*INROOM(ROBOT,PAR7)
	GOTO2(PAR6)				

*UNCLOCKED(PAR6,PAR7)
*INROOM(ROBOT,PAR7)
*JOINSROOMS(PAR6,PAR7,PAR5)
*UNBLOCKED(PAR6,PAR5)
	*NEXTTO(ROBOT,PAR6)	GOTHRUDR(PAR6,PAR7,PAR5)			

*UNBLOCKED(PAR4,PAR5)
*JOINSROOMS(PAR4,PAR5,PAR2)
		*INROOM(ROBOT,PAR5)
		NEXTTO(ROBOT,PAR6)
			GOTO2(PAR4)		

*UNBLOCKED(PAR4,PAR5)
*JOINSROOMS(PAR4,PAR5,PAR2)
*UNBLOCKED(PAR4,PAR2)
		*INROOM(ROBOT,PAR5)	*NEXTTO(ROBOT,PAR4)	GOTHRUDR(PAR4,PAR5,PAR2)	

*~EQ(ROBOT,PAR3)
*UNBLOCKED(PAR1,PAR2)
*~EQ(ROBOT,PAR3)⇒INROOM(PAR3,PAR2)
*PUSHABLE(PAR3)
*JOINSROOMS(PAR1,PAR2,PAR8)
				*INROOM(ROBOT,PAR2)
				NEXTTO(ROBOT,PAR4)
					BLOCK(PAR1,PAR2,PAR3)

				INROOM(ROBOT,PAR2)	NEXTTO(ROBOT,PAR3)
					BLOCKED(PAR1,PAR2,PAR3)

for constants in the goal wff by those constants. That is, in this case, we replace PAR1 by DPDPCLK, PAR2 by RCLK, and PAR3 by BOX2 throughout the MACROP1 triangle table. This instantiated table is then given to PLANEX for execution.

PLANEX is a program that supervises the execution of those ILAs corresponding to the operators in the plan. For a discussion of the operation of PLANEX, see the last part of Section VI, and Ref. 4. PLANEX takes as input a partially instantiated MACROP in triangle table form. (This MACROP may have some parameters remaining after those occurring in the goal wff have been instantiated.) The PLANEX algorithm looks for a specific, fully instantiated subsequence of the operators in the MACROP that can be executed in the present situation to achieve the goal. The ILA corresponding to the first operator is then executed. In the case of the task we are considering the first ILA to be executed is GOTO2(DUNIMYS), which causes the robot to go to the door named DUNIMYS.

The PLANEX algorithm then determines that the next ILA to be executed should be GOTHRUDR(DUNIMYS,RUNI,RMYS). Execution of this ILA begins by calling the vision routine CLEARPATH, which takes a TV picture through the doorway to determine whether the path in RMYS is clear (since the contents of RMYS are unknown). The path is in fact clear, so Shakey proceeds through the doorway.

Next PLANEX calls for the execution of GOTO2(DMYSCLK). Since the contents of RMYS are unknown to Shakey, GOTO calls CLEARPATH again. To illustrate how Shakey can deal with unforeseen difficulties, we now place a box directly in Shakey's path in front of the door DMYSCLK. As Figure 1 and Table 1 show, Shakey does not know of the existence of this box. CLEARPATH determines that the path is blocked and notes the approximate location of the blocking object. Since Shakey expects that it might encounter unknown objects in room RMYS, GOTO next calls a vision routine called OBLOC. This routine calculates the size and exact location of the object, gives it a name, BOX3, and adds this information to the model. (It also assumes, perhaps optimistically, that the new box is pushable.) OBLOC also notes that BOX3 is blocking door DMYSCLK, so it adds the wff BLOCKED(DMYSCLK,RMYS,BOX3) to the model. Since the conditions for continuing the execution of GOTO(DMYSCLK) are no longer satisfied, control returns to PLANEX. Our interest in this experiment is to show how Shakey can gracefully recover from such an unexpected failure of its plan.

PLANEX, as usual, attempts to find a fully instantiated version of the parameterized MACROP1 that can be executed in the present situation to achieve the goal. In this case, PLANEX finds another instantiation of MACROP1 that works. The operators in this instantiation are:

GOTO2(DMYSPDP),GOTHRUDR(DMYSPDP,RMYS,RPDP),
GOTO2(DPDPCLK),
GOTHRUDR(DPDPCLK,RPDP,RCLK)
BLOCK(DPDPCLK,RCLK,BOX2).

Here we see one of the advantages of constructing parameterized plans. To perform the original task, we first constructed a parameterized plan having an instance that solves the problem. Later in the task execution we find that after an unexpected difficulty, another instance of the same parameterized plan can be used to achieve the goal. We expect that this method of error recovery will be quite valuable in robot problems. (If PLANEX could find no applicable instance of MACROP1 that would achieve the goal, then STRIPS would be asked to produce another plan and MACROP.)

After finding this new instance of MACROP1, PLANEX calls for the execution of the first operator GOTO2(DMYSPDP). Shakey thus moves to door DMYSPDP. PLANEX next calls for going through the door, and the process continues until finally Shakey enters room RCLK. Then PLANEX calls for the execution of BLOCK(DPDPCLK,RCLK,BOX2). Running this ILA calls for going to BOX2 and pushing it around BOX1 and then to door DPDPCLK (A "two-leg" push). The local planning needed to accomplish this push operation is done entirely within the PUSH ILA called by BLOCK. With this operation complete, Shakey has accomplished the first task, in spite of the unforeseen difficulty. We also note that MACROP1 has been filed away and can be used as an operator in future problem solving.

b. Task 2

The state of things in Shakey's world is now as shown in Figure 2. We now test Shakey's ability to learn by giving it a task that can be solved by using <u>part</u> of MACROP1. The statement of the task given to the system, in English, is "UNBLOCK DOOR DYMSCLK FROM ROOM RMYS." That is, we want Shakey to move away the object (BOX3) that it discovered to be blocking DMYSCLK.

Again, the English statement is converted into a predicate calculus wff:

UNBLOCKED(DMYSCLK,RMYS).

STRIPS now attempts to find a sequence of operators that will make the wff true, but now it has MACROP1 available in its operator repertoire (in addition to the operators corresponding to ILAs). STRIPS first decides that it should try to apply the operator UNBLOCK(DMYSCLK,RMYS, BOX3). To do so, Shakey must be in room RMYS, so STRIPS looks for operators that will achieve INROOM(ROBOT,RMYS).

STRIPS determines that an instance of the GOTHRUDR operator will work, but so also will subsequences of MACROP1. One subsequence consists of the first two operators in MACROP1 and the other consists of the first four. (For a discussion of how STRIPS makes selections of

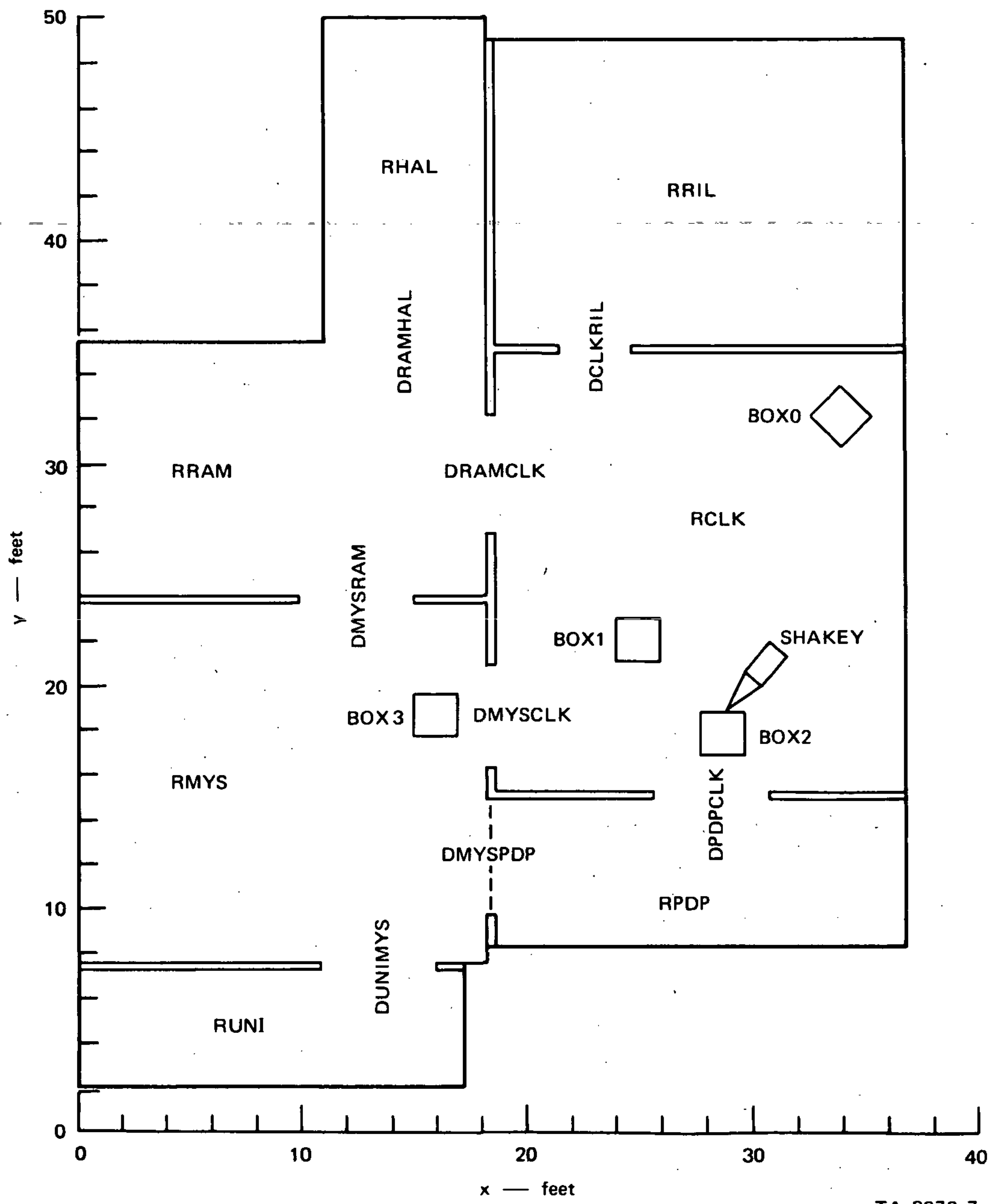


FIGURE 2 MAP OF SHAKEY'S WORLD AFTER COMPLETION OF THE FIRST TASK

MACROP subsequences, see Section VI.) Since an instance of a sequence of the first four operators in MACROP1 is both applicable in Shakey's present situation and achieves the condition INROOM(ROBOT,RMYS), STRIPS is quickly able to settle on this instance and produce a plan for Task 2. Let us denote by MACROP1′ the subsequence of MACROP1 selected by STRIPS. MACROP1′ still contains free parameters that are left to be bound at execution time. Its definition in terms of the operators comprising it is:

<u>MACROP1′ (PAR2,PAR4,PAR5,PAR7,PAR6)</u>

```
GOTO2(PAR6)
GOTHRUDR(PAR6,PAR7,PAR5)
GOTO2(PAR4)
GOTHRUDR(PAR4,PAR5,PAR2)    .
```

The complete generalized plan for the second task is:

```
MACROP1′ (PAR2,PAR4,PAR5,PAR7,PAR6)
UNBLOCK(PAR1,PAR2,PAR3)    .
```

This generalized plan is given the name MACROP2 and is saved for possible later use. The triangle table representation of MACROP2 is shown in Table 4.

After creating the general version of MACROP2, STRIPS prepares a version of it for PLANEX by instantiating it with those constants appearing in the task description. Namely, DMYSCLK is substituted for PAR1 and RMYS for PAR2. It then gives this partially instantiated version to PLANEX to be executed. PLANEX finds that the following instantiation of the plan will achieve the goal:

```
MACROP1′ (RMYS,DMYSRAM,RRAM,RCLK,DRAMCLK)
UNBLOCK(DMYSCLK,RMYS,BOX3)    .
```

Table 4

TRIANGLE TABLE FOR MACROP2(PAR3,PAR1,PAR6,PAR7,PAR5,PAR4,PAR2)

*INROOM(ROBOT,PAR7)
*UNBLOCKED(PAR6,PAR7)
*JOINSROOMS(PAR6,PAR7,PAR5)
*UNBLOCKED(PAR6,PAR5)
*JOINSROOMS(PAR4,PAR5,PAR2)
*UNBLOCKED(PAR4,PAR5)
*UNBLOCKED(PAR4,PAR2)
	MACROP1′(PAR2,PAR4,PAR5,PAR7,PAR6)	

*BLOCKED(PAR1,PAR2,PAR3)	*INROOM(ROBOT,PAR2)
	NEXTTO(ROBOT,PAR4)
		UNBLOCK(PAR1,PAR2,PAR3)

	INROOM(ROBOT,PAR2)	NEXTTO(ROBOT,PAR3)
		UNBLOCKED(PAR1,PAR2)

Next, PLANEX calls for execution of MACROP1′. This execution is accomplished by PLANEX itself. The ability to handle "nested" triangle tables is one of the features of our system. PLANEX discovers that the first ILA to be executed in MACROP1′ is GOTO(DRAMCLK). In a similar manner, PLANEX ultimately executes the entire string of ILAs in MACROP1′ and then the UNBLOCK ILA to accomplish the second task.

When these experiments are actually conducted, it is probable that the system may decide to exercise another one of our error-recovery capabilities. Recall that the model contains information about the probable error in Shakey's location stored in the predicate DAT. Model-maintenance programs automatically increase the estimate of error after every robot motion. During execution of ILAs such as GOTO2, this probable error is checked to see whether it is still less than some specific tolerable error. Whenever the error estimate exceeds the tolerance, a visual program called LANDMARK is called. LANDMARK takes a picture of some nearby feature (such as a doorjamb), calculates from this picture the robot's actual location, and enters this updated location into the model. It also resets the DAT predicate to the error estimate appropriate after having just taken a picture.

Several features of the system are illustrated in these experiments. Most important of these are the ability to learn generalized plans and the ability to recover from various types of failures. The system of ILAs is designed to be robust in the sense that each ILA does what it can locally to correct any errors. When the appropriate recovery procedures are beyond a specific ILA's knowledge and abilities, there are several higher levels where recovery can occur, namely, at higher level ILAs, in PLANEX, or in STRIPS.

III LOW-LEVEL ACTIONS

A. Introduction

The low-level actions, or "LLAs", define the interface between major robot software packages and the bottom, hardware-oriented level of the system. The intermediate-level actions (ILAs), to be described in Section IV, control the operation of these LLAs. The LLAs, in turn, communicate with the PDP-15 computer and the robot vehicle according to the protocol described in Appendix G, "Robot Communication between the PDP-15 and the PDP-10," of the Semiannual Report.[2] Initial specifications for these programs appeared as Appendix B, "Bottom level PDP-10 Software for the SRI Robot," of the 1970 Final Report.[1]

In this section we shall describe the upper face of the LLAs, i.e., the face presented to higher-level programs.

Since the robot moves very slowly, we have taken great pains to permit the user to view the robot as behaving asynchronously to as great an extent as appropriate. Thus the user must take cognizance of this asynchrony by confirming the completion or "settling" of any robot activity before doing anything that assumes that activity to have been successful. This low-level software package provides the necessary interlocking in the following manner. Communications between the user and the robot are separated into two unidirectional channels: orders from the user to the robot are handled by calls on LLAs (i.e., the functions in this package); the current state of the robot's world is reflected in the robot's world model. Now, the functions by which the user can access these particular entries in the robot's world model have special provisions to ensure that an activity has settled before granting access to any part of the model which that activity might affect. For example, one might move the robot to a given location by first turning it to face the target spot and then rolling it straight forward by the required distance. One could conceivably confirm the initial turn (by interrogating the proper part of the model) before rolling ahead. The model-access function will then delay until the turn has settled before reporting the bearing of the robot. On the other hand, the user will not be delayed for completion of the roll until he interrogates the position of the robot. Thus we have synchronization (between the user and the robot) whenever we need it but not otherwise.

This sort of synchronization is effected in another circumstance having to do with interlocks between activities. In particular, each activity has associated with it certain conflicting activities. (For example, one cannot take a TV picture while the robot's head is panning.) A set of initiation functions automatically take cognizance of all possible conflicts: each ensures that all potentially conflicting activities are settled before initiating its own activity. For the purpose of programming actual use of the robot, however, one should note that settling of an activity does not necessarily mean its successful completion. For example, a roll can terminate by the robot unexpectedly bumping into some obstacle--this will "settle" the roll, but the robot cannot be assumed to have attained its destination.

B. Measurement and Control

Before proceeding further, we shall define the precise robot capabilities that the LLAs control. Shakey can move about the floor by turning his body and by rolling straight forward or backward, and he can pan and tilt his head. He can take pictures and rangefinder readings, and he can adjust the focus and iris states of the TV camera's lens. Finally, he can set some global parameters both for taking TV pictures and for rolling or turning. These ten activities will be more fully explained below. First we shall describe the measurement conventions in Shakey's environment.

The robot rolls about in a right-handed Cartesian coordinate system. Distances are measured in feet. The world is partioned into rooms by doors and walls. The walls lie parallel to Cartesian axes, and any door is completely contained in a wall. This coordinate system is constant in the sense that all robot motion is relative to it and that other things generally move with respect to it only when the robot pushes them.

Angles are measured in degrees, and we will call the principal value of an angle that value between -180° and +180°. The bearing of the robot is a horizontal angle referred to the positive direction of the global y-axis; thus the robot is parallel to the x-axis in the negative sense when its bearing is 90°. The pan angle of the robot's head is a horizontal angle referred to the robot's bearing, and the tilt angle of the robot's head is a vertical angle measured from the horizontal plane. Thus when the robot has its pan angle at zero and the tilt angle at -45°, the rangefinder and TV camera are pointed at the floor right before its very wheels.

We turn now to optical values. The iris of the TV camera is set in exposure value units (EVs), which have a logarithmic relation to f-numbers: increasing the EV number by one doubles the amount of light arriving at the inner regions of the TV camera. Focus values and rangefinder readings are distances in feet from the intersection of the axes about which the robot's head tilts and pans. That point in turn is about 4 feet 1-1/2 inches above the floor and 9 inches forward of the axis about which the robot turns, when the robot is standing (or sitting or whatever it does) on a level flat floor.

Having covered the numeric quantities in the robot's world, we have but a few other items to discuss. Perhaps the simplest of these to describe is a TV picture: it resides on a disk file in FORTRAN binary format. Now TV pictures are digitized in square arrays of picture elements; the size of the array is constant, but one can select two coarsenesses: 120 or 240 picture elements on a side. One can, however, alter the configuration of the array for the sake of special stereo optics. These two options are combined into one number called the tvmode, as follows:

"tvmode: 0 means 120 $\times$ 120 nonstereo

"tvmode" 1 means 120 $\times$ 120 stereo

"tvmode" 2 means 240 $\times$ 240 nonstereo

"tvmode" 3 means 240 $\times$ 240 stereo.

To explain the last two quantities of this section, we must first explain the two main tactile sensors of the robot and how they interact with the roll and turn activities. The tactile sensors are seven catwhiskers and a pushbar; each catwhisker can signal engagement with an obstacle, and the pushbar can signal each of two levels of pressure: mere engagement and hard contact. All nine of these conditions are reflected in a quantity called the whiskerword; to a first approximation each of these conditions has its own bit in the whiskerword, whose format is shown in the following table:

Bit No.	Octal Code	Meaning of "1"
21	040000	Pushbar is engaged and ready to push.
23	010000	Left front whisker is engaged.
25	002000	Front horizontal whisker is engaged.
26	001000	Right front whisker is engaged.
28	000200	Right rear whisker is engaged.
29	000100	Encountered immovable object and backed off.
30	000040	Rear whisker is engaged.
33	000004	Left rear whisker is engaged.
35	000001	Front vertical whisker is engaged.

The robot has a couple of motor reflexes pertinent to this disucssion: it will stop moving whenever the pushbar becomes disengaged, and it will not move while a catwhisker is engaged. However, these two reflexes can be overriden selectively; the corresponding orders are sent to the PDP-15 by means of the override activity and the override code word, which has the following significance:

Code Word	Pushbar	Catwhisker
0	Enabled	Enabled
1	Enabled	Overriden
2	Overriden	Enabled
3	Overriden	Overriden

C. The LLA Portion of Shakey's Model

We will now enumerate and define the 17 predicates by which the robot's lowest-level state is represented in the axiomatic world model. They are:

Atom in axiomatic model	Affected by
(AT ROBOT xfeet yfeet)	ROLL
(DAT ROBOT dxfeet dyfeet)	ROLL
(THETA ROBOT degreesleftofy)	TURN
(DTHETA ROBOT dthetadegrees)	TURN
(WHISKERS ROBOT whiskerword)	ROLL, TURN
(OVRID ROBOT overrides)	OVRID
(TILT ROBOT degreesup)	TILT
(DTILT ROBOT ddegreesup)	TILT
(PAN ROBOT degreesleft)	PAN
(DPAN ROBOT ddegreesleft)	PAN
(IRIS ROBOT evs)	IRIS
(DIRIS ROBOT devs)	IRIS
(FOCUS ROBOT feet)	FOCUS
(DFOCUS ROBOT dfeet)	FOCUS
(RANGE ROBOT feet)	RANGE
(TVMODE ROBOT tvmode)	TVMODE
(PICTURESTAKEN ROBOT ±picturestaken)	SHOOT

The two predicates AT and THETA give the position and bearing of the robot itself in the global coordinate system; the statistical uncertainties are given by the predicates DAT and DTHETA, which are separated from AT and THETA to facilitate planning. The state of the whiskerword is updated whenever a ROLL or TURN settles, and the OVRID predicate reflects the state of the overrides in the robot.

The TILT and PAN predicates refer to the direction the robot's head is pointed. DTILT and DPAN give corresponding error estimates. All three angles (tilt angle, pan angle, and heading THETA) are stored as their principal values. RANGE gives the value resulting from the most recent rangefinder reading. the PICTURESTAKEN predicate, which we will describe more fully in our discussion of the SHOOT activities, gives the approximate number of pictures taken to date. The meanings of the rest of the predicates should be clear from the previous discussion.

D. The LLAs

The predicates are the means by which the robot tells the user about its state; the LLAs provide the means by which the user tells the robot to alter its state. One should understand that this clean division is largely just formal; in practice an interrogation of a predicate is intercepted by a function that ensures settling of any relevant robot activities before proceeding to the actual access. Also, the initiation of an action does not guarantee its completion; actions may terminate for a variety of reasons, such as engagement of limit switches or malfunctions in the telemetry link. The state of the system after an action may be determined by investigating the model.

The following functions initiate fundamental low-level activities (whenever numeric parameters are used, negative numbers are permissible and mean motion in the direction opposite to that indicated):

(TILT degreesup) tilts the robot's head upward by "degreesup" degrees. The motion can be prematurely terminated by a limit switch.

(PAN degreesleft) pans the robot's head by "degreesleft" degrees to the left or far enough to activate a limit switch.

(FOCUS feetout) the TV camera is initially focused on a plane removed by some focal distance from the center of the head's gimbals; this function increases that distance by "feetout" feet. Of course the range of focal distances is limited by limit switches.

(IRIS evs) opens the robot's iris (on the TV camera) by "evs" EVs. Thus if "evs" has the value 1, this form will double the amount of light getting into the TV camera. There are limits for this activity too.

(OVRID overrides) sets the overrides as specified by the "overrides" code word.

(TVMODE tvmode) sets the TV mode as specified by the "tvmode" code word.

(RANGE) reads the robot's rangefinder; this automatically includes turning on the rangefinder and waiting for it to warm up.

(SHOOT) puts a TV picture onto the disk file "TV.DAT". The picture is taken according to the current TV mode. Assuming correct operation of hardware and software, a subsequent examination of the PICTURESTAKEN atom (in the world model) will yield a positive integer giving the number of current pictures in a series (1,2.3,...) begun when the robot system was loaded or initialized. In the event of an unrecovered system malfunction (e.g., transmission error), the value stored with PICTURESTAKEN will be the negative of the serial number of the last successfully taken picture.

<u>(ROLL feet)</u> tells the robot to roll forward by "feet" feet. This activity has three normal ways of prematurely terminating: the robot can come into contact with an obstacle, engaging a catwhisker; it can lose contact with an object it is pushing, disengaging the pushbar; or it can encounter an immovable object, causing the pushbar to come on hard. The first two conditions cause the robot to stop by reflex actions that can be overridden; the last causes the robot to attempt to free itself using more complex evasive actions in a reflex that cannot be overridden. When the robot encounters an immovable object, it will not only stop, but it will back away from it by some distance, currently a constant 6 inches. (Of course, the information in the model will be correctly maintained.) The whiskerword in the model is updated at the end of a ROLL or TURN; it contains the description of the current state if the catwhiskers and pushbar are returned from the robot, but it has another bit for immovable objects--this bit showing the history of an event rather than showing a current state. This bit is set only when the whiskerword is updated the first time after hard contact.

<u>(TURN degreesleft)</u> tells the robot to turn to the left by "degreesleft" degrees. Otherwise the above description of the ROLL activity applies excepting only the way immovable objects are evaded. In this case, the robot turns back; currently it turns back to its initial heading.

The functions discussed so far that initiate motions have been incremental in form if not in essence. However, even this level of robot software has a memory of the various aspects of the robot's position in the axiomatic model so dutifully maintained by the settling functions. Capitalizing on this circumstance, we have also provided some functions to initiate motions to a given goal (rather than by a given amount).

Although these functions are formally and conceptually outside the lowest LISP level of robot software, they have sufficiently simple internal structure that it is convenient to describe them here rather than in the next (ILA) section. With one exception we expect their meanings to be self-evident. These additional initiation functions are:

```
(TILTO degreesup)
(PANTO degreesleft)
(FOCUSTO feet)
(IRISTO evs)
(ROLLTO xfeet yfeet)
(TURNTO degreesleftofy).
```

The exception is ROLLTO: it must first turn the robot to point toward its goal, so it must do (and does) more than simple calling the corresponding incremental function with the difference between the desired and current position.

E. Summary

Table 5 is a summary of Shakey's low-level activities. Figure 3 sketches how these activities fit into the overall system control structure.

Table 5

LOW-LEVEL ACTIVITIES OF ROBOT

Initiation Functions					
Primary	Absolute	Conflicts (+self)	Terminating Conditions	Needs from Model	Puts into Model
(TILT degreesup)	(TILTO degreesup)	RANGE,SHOOT	upper limit (35°) lower limit (-45°)	TILT,DTILT	TILT,DTILT
(PAN degreesleft)	(PANTO degreesleft)	RANGE,SHOOT	left limit (116°) right limit (-107°)	PAN,DPAN	PAN,DPAN
(FOCUS feetout)	(FOCUSTO feetout)	SHOOT	near limit far limit	FOCUS,DFOCUS	FOCUS,DFOCUS
(IRIS evs)	(IRISTO evs)	SHOOT	open limit closed limit	IRIS,DIRIS	IRIS,DFOCUS
(OVRID overrides)	--	TURN,ROLL	--	--	OVERRIDE
(TVMODE tvmode)	--	SHOOT	--	--	TVMODE
(RANGE)	--	TURN,ROLL,TILT,PAN	--	--	RANGE
(SHOOT)	--	TVMODE,ROLL,TURN, TILT,PAN,IRIS,FOCUS	--	TVMODE,PICTURESTAKEN	PICTURESTAKEN
(ROLL feet)	(ROLLTO xfeet yfeet)*	TURN,RANGE,OVRID, SHOOT,ROLL	bump-ignored bump-stopped drop object-stopped immovable object-backed off	AT,DAT,THETA,DTHETA	AT,DAT,WHISKERS
(TURN degreesleft)	(TURNTO degreesleft)			THETA,DTHETA	THETA,DTHETA, WHISKERS

*ROLLTO evokes the TURN activity.

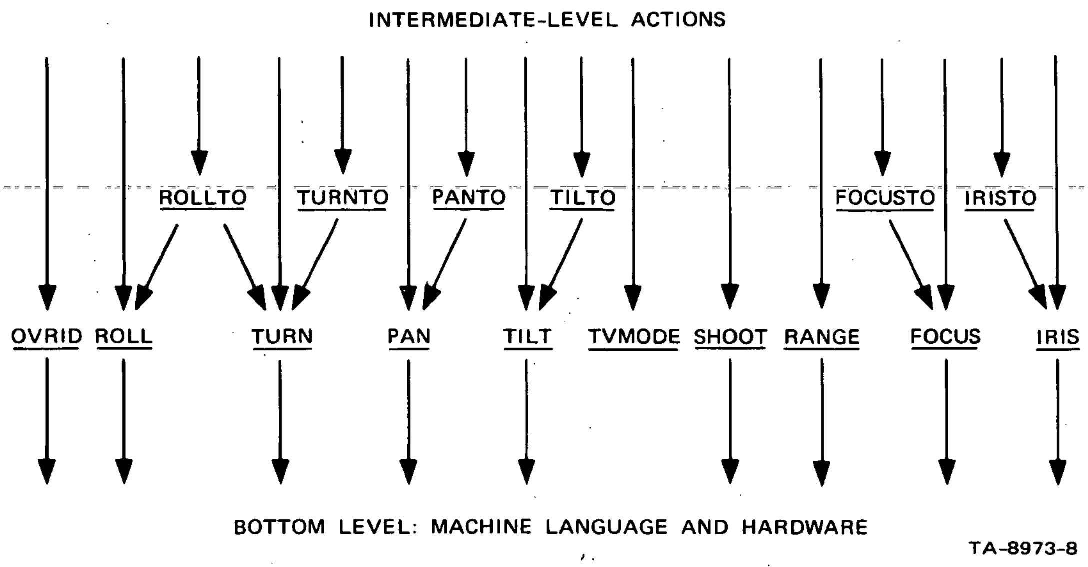


FIGURE 3 CONTROL STRUCTURE OF LOW-LEVEL ACTIVITIES

IV INTERMEDIATE-LEVEL ACTIONS

A. Introduction

The Intermediate-Level Actions (ILAs) are the action routines associated with the STRIPS operators described in Table 2. Here we distinguish "action routines" from "operators" on the following basis: operators are used by STRIPS for planning, and the corresponding action routines are invoked by PLANEX (or any other suitable executive) to actually move the robot. The ILAs are written in a language we call Markov because of its resemblance to Markov algorithms.[6] There is a large body of auxiliary LISP functions that accompanies the ILAs, but we will confine the present discussion to a brief description of the Markov language and a brief exposition of the current ILAs and the intraroom navigation algorithm.

B. The Markov Language

The central part of the Markov language is the Markov table, specifying actions to be performed and the criteria for determining their sequence. The format of a Markov table is an ordered collection of rows of identical format. Each row starts with a label, which is followed by a predicate, a sequence of actions to be performed, and finally the label of some other line in the table. This last item (which we have been calling the "go-to") can optionally specify that execution of the table could cease, causing the calling routine's execution to resume in the conventional subroutine fashion. The characteristic execution pattern is a sequential scan through the table's rows, testing the predicates one by one until a row is found whose predicate is true. Then the scan terminates and the actions (if any) in that row are performed, and the go-to is followed; it will either indicate completion of the execution of the table, or it will name a line in the table at which the scan is to recommence. When the Markov table is first entered, the scan begins with the first line in the table. Execution may be terminated in three ways: it can be completed explicitly, by reaching a special go-to; the sequential scan can get to the bottom of the table without having found a line with a true predicate; and finally, an action can be fruitless, which will cause a loop suppressor to terminate execution of the table. In all three cases, there is only one form of return from a Markov table, and the calling routine (or Table) is expected to test for the desired results. (This seemed much simpler than trying to make the individual action routines guess what its caller had in mind.)

The actions called for in an ILA may be LLAs, other ILAs, or arbitrary programs (usually coded in LISP). Since the Markov interpreter is itself a LISP program, any ILA can call itself recursively.

The "go-to" part of a Markov table line is interpreted after completion of the action part. In its simplest case, the "go-to" consists of the label of a line at which to continue the search for a true predicate. If several lines have the given label, one of the lines is arbitrarily chosen; if no lines have it or if it is NIL, execution is terminated. (NIL is our conventional explicit return.) The other case involves "loop suppression" and will be discussed below.

A Markov table is generally a sequence of actions that would transform an initial state into a final "goal" state via a linear sequence of intermediate states. Whether an action is applicable to a particular state can usually be tested by a relatively simple predicate--the one heading the table line with the action. Since actions in the real world frequently fail to achieve their desired results, the Markov interpreter determines which action to execute by testing the state predicates one by one, starting from the goal predicate (on the top line) and working backward (i.e., down the table) until a true predicate is found. Markov operators typically follow the execution of any component action by starting again with the goal predicate. In its simplest form, each line of a Markov table would contain one of the state predicates and the operator to be applied to that state; its "go-to" would specify the first line, which contained the goal predicate and an explicit return. Falling off the end of a Markov table thus corresponds either to a drastic failure of one of the component actions or to an inappropriate application of the Markov operator. Of course, persistent failure of a component action to achieve its desired effect, i.e., to produce a state satisfying a predicate higher in the table, would cause indefinite looping in such a Markov table. To circumvent this possibility without requiring specific consideration in each Markov table, we introduced "loop suppression" into the Markov interpreter. Whenever the predicate of a line is found to be true, a counter is incremented and checked against a limit before the line's action is executed; if the counter becomes greater than the limit, then interpretation of the table is terminated without execution of the action. Thus if the limit for a line is three (this is the current default value) then the action(s) on that line will be executed a maximum of three times; if the line's predicate is found true a fourth time, the table will return to the operator that invoked it. Of course, one can specify a limit for a table line rather than accepting the default value. There is an alternative form for the "go-to" just for this purpose: rather than being just a label, it can be a two-element list. In this case, the first element is the label, and the second element is the loop-suppression limit for that line; it is evaluated only once, at the time of the first loop-suppression check for that line.

Table 6 illustrates the Markov language by presenting the actual code for the lowest-level ILA that pushes an object. Here, line 10 does some initialization; the action [i.e., the (SETQ XYTARG...)] is always performed because its predicate T is always true. Then line 20's predicate checks whether the pushing operation is finished by means of its (NEARENOUGH OB XYTARG TOL) predicate; if this is the case, then no actions (i.e., NIL) are performed, and control jumps to the label CLEANUP for some post-processing before exit. Line 25's predicate similarly determines whether the object's position is known closely enough to continue the pushing operation. (This may not be the case either initially or as the result of the object dropping off the pushbar during a push.) Line 30 causes the table to exit (via CLEANUP) if the object is past its target. Line 40's predicate is true if the robot has just pushed the object into a wall, and finally, line 50's predicate is true if the robot has proper contact with the object. Line 10 and the lines starting with the label CLEANUP are representative of a more usual programming language, with the normal execution being sequential. Lines 20 through 50, however, have the characteristic execution pattern of the ILAs: a loop testing for the main goal and various subgoals and error conditions and recycling after any action is performed. This particular ILA is designed to be especially simple because it is intended to be embedded in several more layers of ILA before STRIPS becomes concerned with their robustness. Even STRIPS-visible ILAs are called by PLANEX from its execution tables, so it is perfectly acceptable for this lowest-level pushing operator to fail as readily as it does.

C. <u>The Actions</u>

The following are brief descriptions of the present ILAs. The control relations among the ILAs and between them and the rest of the system are shown in Figure 4.

Table 6

MARKOV TABLE FOR THE LOWEST-LEVEL PUSHING ILA

```
(DEFPROP PUSH1 PUSH1 (*: MARKOVTABLE NIL))

(DEFPROP PUSH1
        ((10. T ((SETQ XYTARG (XYTARG (OBPOS OB) (MLVFIND (QUOTE (THETA ROBOT $))) DIST))) 20.)
         (20. (NEARENOUGH OB XYTARG TOL) NIL CLEANUP)
         (25. (NOT (NEARENOUGH OB (OBPOS OB) TOL)) NIL C1)
         (30. (GREATERP (ABS (ANGLEDIF (BEARINGTO XYTARG (OBPOS OB)) (MLVFIND (QUOTE (THETA ROBOT $))))) 90.)
              NIL
              CLEANUP)
         (40. (MEMQ (QUOTE HC) (WHISKERS))
              ((SETQ DOSETPOS NIL) (SETPUSHOBPOS OB (PLUS RADFRONT 0.5)))
              CLEANUP)
         (50. (MEMQ (QUOTE FH) (WHISKERS))
              ((OVRID 1.) (ROLL
                            (DIFFERENCE (DISTANCE XYTARG (OBPOS (QUOTE ROBOT)))
                                        (PLUS RADFRONT (MLVFIND (LIST (QUOTE RADIUS) OB (QUOTE $))))))
                           (OVRID 0.)
                           (SETQ DOSETPOS NIL)
                           (SETPUSHOBPOS OB RADFRONT)
                           (MLDELETE (LIST (QUOTE NEXTTO) OB (QUOTE $)))
                           (MLDELETE (LIST (QUOTE NEXTTO) (QUOTE $) OB)))
             20.)
        (CLEANUP DOSETPOS ((SETPUSHOBPOS OB (PLUS RADFRONT 0.5))) C1)
        (C1 (FCWON) ((ROLLBACK) (ROLL -1.)) C2)
        (C2 T ((MLDELETE (QUOTE (NEXTTO ROBOT $)))) R))
       (*: MARKOVTABLE TABLE))

(DEFPROP PUSH1 (DIST OB TOL) (*: MARKOVTABLE PARAMETERS))

(DEFPROP PUSH1 ((TOL 1.) XYTARG (RADFRONT 1.5) (DOSETPOS T)) (*: MARKOVTABLE LOCALS))
```

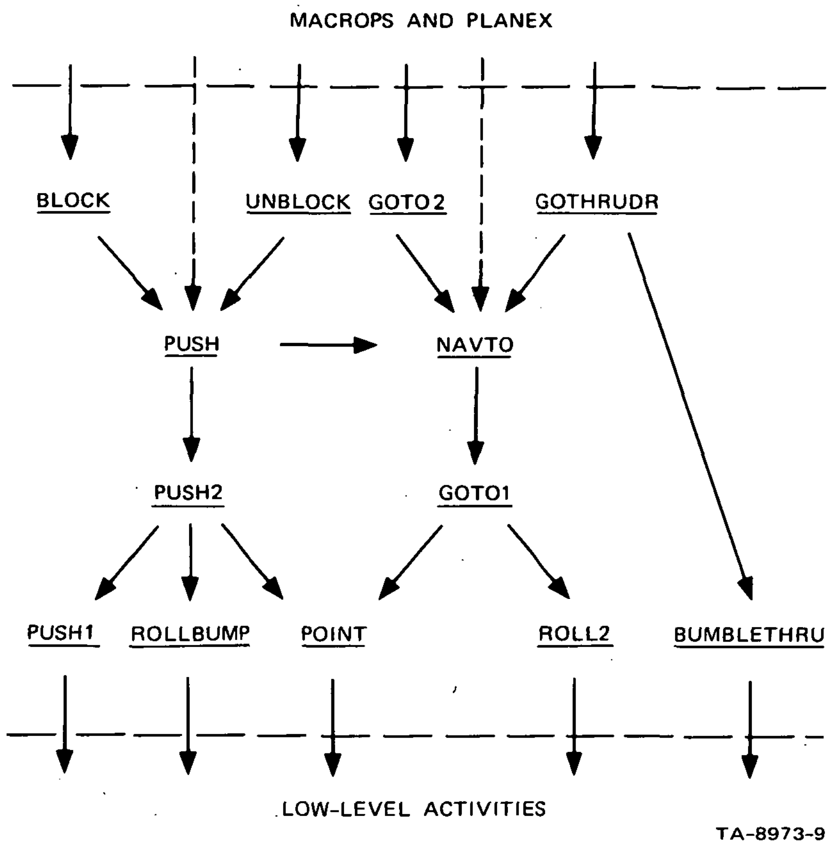


FIGURE 4 CONTROL STRUCTURE OF THE INTERMEDIATE LEVEL

ILAs that affect the state of the world have responsibility for making corresponding changes to Shakey's axiom model of the current world. Such changes are mentioned below wherever relevant; "$" will be used to denote unspecified or changing values in the model.

GOTHRUDR(DOOR FROMRM TORM) moves the robot from room FROMRM to room TORM via door DOOR. It assumes only that the robot is in FROMRM; it uses NAVTO to get to the door and BUMBLETHRU to go through it.

BLOCK(DX RX BX) pushes box BX within room RX to a position blocking door DX. This routine directly replaces the axiom (UNBLOCKED DX RX) by (BLOCKED DX RX BX) in the model.

UNBLOCK(DX RX BX) pushes box BX within room RX to a position in which it does not block door DX; it directly replaces the axiom (BLOCKED DX RX BX) by UNBLOCKED DX RX). This routine prefers to push the box to the far side of the door (as viewed from the initial position of the robot), but it will also consider the other push.

GOTO2(X) moves the robot into the vicinity of X if X is a door; it directly updates the (NEXTTO ROBOT $) axiom. A contemplated extension of GOTO2 is to permit X to be an object.

PUSH1(DIST OB TOL) is the lowest-level push; as such, it maintains OB's position and deletes the (NEXTTO OB $) and (NEXTTO $ OB) axioms from the model. It pushes OB forward by DIST feet (within TOL feet); it assumes that the front horizontal catwhisker is on when it is entered, and it exits under any of the following conditions:

(1) It is unnecessary to push OB forward, i.e.:

 (a) OB is within TOL of the implied goal point; or

 (b) OB is past the goal point in the current heading.

(2) The pushbar comes on hard.

(3) The front horizontal catwhisker is off.

In any of these cases, the robot backs up 2 feet in an attempt to free its catwhiskers for normal navigation. The last argument TOL is optional and is defaulted to 1 foot if not supplied.

ROLL2(DIST TOL) is the lowest-level free-floor roll; as such it deletes the (NEXTTO ROBOT $) axiom from the model. It moves the robot forward by DIST feet (within TOL feet); if it engages a front catwhisker it asserts the (JUSTBUMPED ROBOT T) axiom and backs away in an attempt to free the catwhisker. TOL is an optional parameter defaulted to 1 foot if not supplied; DIST may be negative.

BUMBLETHRU(FROMRM DOOR TORM) moves the robot from room FROMRM to room TORM through door DOOR. It assumes that the robot is initially in FROMRM and in front of DOOR. It heads for the corresponding position in TORM and uses the catwhiskers (if necessary) to help it negotiate the door. It updates the (INROOM ROBOT $) and (NEXTTO ROBOT $) axioms in the model, and it is the most basic door-negotiating routine in the system. It uses the vision routine CLEARPATH before entering an unknown room.

PUSH(OBJECT GOAL TOL) is the highest-level ILA for pushing a box. Its three arguments are the name of an object, the goal coordinates to be pushed to, and the allowable tolerance. The tolerance argument may be omitted, in which case its value defaults to 2.0 feet.

The only precondition for PUSH is that Shakey and the OBJECT are in the same room. The routine calls FINDPATH (described below) to plan a path to GOAL from the current object location. PUSH will fail if any of the following conditions are true:

(1) OBJECT is not in a pushable location.

(2) No path of width W [W = MAX(WIDTH(OBJECT),WIDTH(ROBOT))] can be found from the current position of OBJECT to GOAL.

(3) No path can be found from the current position of the robot to the "pushplace" of OBJECT, i.e., Shakey cannot get behind OBJECT.

PUSH2(OBJECT GOAL TOL) is a straight-line push, invoked by PUSH to move OBJECT along successive legs of the planned path. PUSH2 attends to updating the positions of ROBOT and OBJECT. If the uncertainties in position exceed TOL, PICLOC updates the position of ROBOT or OBLOC the position of OBJECT (PICLOC and OBLOC are described in Section V).

A PUSH2 is accomplished in three basic stages:

(1) The robot navigates to the "pushplace" of OBJECT.

(2) The robot rolls forward and makes contact with the object with a front catwhisker, by using ROLLBUMP.

(3) PUSH1 is called, which turns on the overrides and causes the robot to roll forward the required distance.

NAVTO(GOAL TOL) will maneuver the robot to within TOL feet of the point GOAL. Like the PUSH ILA, it uses FINDPATH to plan the journey to GOAL. NAVTO will fail if no path is found; if a path exists, it uses POINT AND GOTO1 for each leg of the journey.

POINT(THETA TOL) attempts to turn the robot to within TOL degrees of bearing THETA. If necessary, the vision routine PICTHETA (Section V) will be used to determine the bearing of the robot. A catwhisker engaged during the turn will cause the robot to turn back to its original bearing and then attempt to locate the object with PICBUMPED (Section V).

GOTO1(GOAL TOL) moves the robot forward in a straight line to within TOL feet of GOAL. It will use ROLL2 to actually move the robot, or it will use vision under the following conditions:

(1) If the robot's location is uncertain (>TOL), it will update its position using PICLOC.

(2) If moving in an unknown room, it will use CLEARPATH.

(3) If the result of CLEARPATH is BLOCKED, it will use PICDETECTOB (Section V) to enter information about the obstacle in the model.

(4) If the robot unexpectedly engages a catwhisker while rolling, PICBUMPED will locate the object and update the model.

ROLLBUMP(DIST TOL OBJECT) moves the robot forward DIST feet to engage a front catwhisker on the object OBJECT. It updates the (NEXTTO ROBOT $) predicate(s) in the model. If an object is not encountered within TOL feet of DIST, ROLLBUMP fails.

D. The Pathfinding Algorithm

FINDPATH(ROB G JOURN) is the routine to plan an intraroom path from ROB to G. The arguments ROB and G are each a list of X, Y coordinate pairs. JOURN is the type of journey to be undertaken, either ROLL or PUSH. If JOURN is ROLL, the function returns a path along which the robot can navigate from ROB to G. If JOURN is PUSH, the returned value is a path by which the robot can move a box at ROB to point G. In this case global variables PUSHOBNAME (name of the box) and OBRAD (radius of the box) are set, so that in computing a pushing path the box radius and the ability of the robot to get behind the box are taken into account.

The returned value from FINDPATH is a list of subgoal points to be arrived at in order: $(X_1Y_1)(X_2Y_2)\ldots(X_{n-1}Y_{n-1})G)$. If a direct-line path exists from ROB to G, the value of FINDPATH is just (G); if no path exists, the value is NIL.

The pathfinding algorithm is a breadth-first search of the tree of predecessors to G. At each node of the tree, FINDPATH tests for a direct-line path between ROB and the current node, say PN. If it exists, the path from PN to G is returned. Otherwise, the tree is grown one level deeper from PN by computing predecessors to that point. If no predecessors exist, the path from PN to G is removed from the tree, thus reducing the search space.

The predecessors to node PN are defined as the intersections of the tangent lines from ON and ROB around the obstructing object in the straightline path connecting them. Thus, each point has at most two predecessors. Figure 5 illustrates one possible configuration that would generate the tree in Figure 6.

Before a computed predecessor is added to the tree, it is tested to determine whether it is within the room or within the region of another obstacle. If either condition is true (as for PØ in Figure 5), a shorter path (P5 P4) is computed using the tangents that generated PØ. If either of these points is unacceptable under the criterion just described, the entire search in that direction is abandoned, and the next node (in this case P3) is considered. A predecessor that is acceptable under this criterion is added to the tree <u>if</u> a straightline path exists between it and its parent node. Otherwise, predecessors are sought recursively to find a path from the parent node to its computed predecessor.

The searching in FINDPATH terminates, then, when either a path has been found or when the search tree is reduced to NIL. Thus, the path that is chosen (assuming at least one exists) is the first one found,

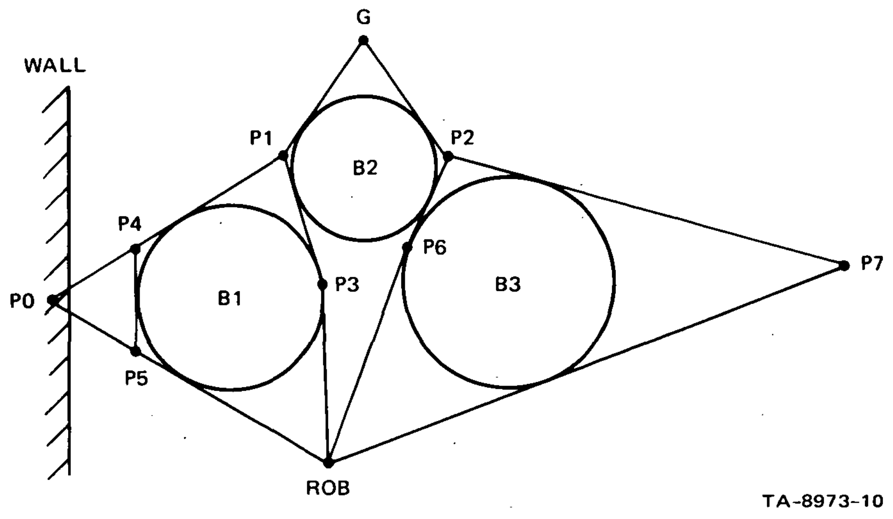


FIGURE 5 AN OBSTACLE CONFIGURATION FOR FINDPATH

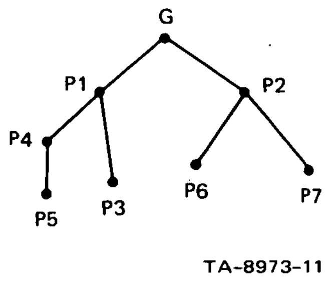


FIGURE 6 SEARCH TREE FOR CONFIGURATION OF FIGURE 5

that is, the one with the smallest number of legs in the journey. This criterion was chosen over a minimum-distance criterion to reduce the amount of subsequent thinking and execution time for the robot.

V VISION ROUTINES

A. Introduction

The current robot executive program never calls for a general visual scene analysis. Instead, under appropriate circumstances various of the intermediate-level actions (ILAs) call specific vision routines to answer certain specific questions. These specialized vision programs perform three basic tasks: locating and orienting the robot, detecting the presence of objects, and locating objects.

A summary of the six vision routines currently used by the ILAs is given in Section V-C. Some of these, such as PICLOC and CLEARPATH, have been described in detail in previous reports (Refs. 1 and 2, respectively). Most of the other routines make use of LOBLOC, which uses vision to locate accurately an object whose position is only roughly known.

The following section describes the operation of this routine in some detail.

B. Object Location

Given the approximate floor location of an object, OBLOCF takes a television picture of the object, analyzes the picture to find the exact coordinates, and enters this information in the robot's world model. This specialized task can be done more rapidly and with less chance for error by a special program than by performing a complete scene analysis and then extracting the desired answer from the resulting description. However, certain preconditions must be satisfied for LOBLOC to function properly. These are as follows:

(1) The approximate location must be sufficiently accurate and the object must be sufficiently small and unoccluded that at least two, and preferably three, lower corners of the object are in view.

(2) The object and the robot must be in the same room.

(3) The location of the robot with respect to the walls must be known to within approximately one foot.

The first action that LOBLOC performs is to pan and tilt the television camera so that the nominal floor position images in the center of the picture. The resulting picture is taken at 60-line resolution to speed subsequent region analysis operations. However, before region analysis is begun, the program accesses the model to compute the image of the wall-floor boundary. Everything in the picture above this boundary is erased, thereby eliminating baseboards, door jambs, and other possible sources of confusion.

The resulting picture is then subjected to region analysis. That is, it is partitioned into elementary regions, and these regions are merged using the phagocyte and weakness heuristics.[7] The following regions are automatically deleted from the resulting region list:

(1) The region above the wall-floor boundary.

(2) All regions smaller than some threshold θ. (Currently $\theta = 4$ cells.)

The next major step is to identify the floor region. This is done by scoring each region. The features or properties that enter into this score are the area A, the ratio R of perimeter-squared to area, the average brightness B, and the lowest coordinate Z of the external contour. Letting A_{max} be the largest area, R_{max} the largest ratio, B_{max} the highest brightness, and Z_{min} the smallest coordinate, we compute the scoring function by

$$D^2 = \left(1 - \frac{A}{A_{max}}\right)^2 + \left(1 - \frac{R}{R_{max}}\right)^2 + \left(1 - \frac{B}{B_{max}}\right)^2 + \left(\frac{Z - Z_{min}}{60}\right)^2 .$$

The region for which D^2 is minimum is declared to be the floor.

The next major step is to inspect the n neighbors of the floor to find the ones that are most likely to be the faces of the object in question. Special tests are made to treat the simple cases where n happens to be 0, 1, or 2. In general, for each region neighboring the floor we compute its area A and a quantity X which is a simple measure of the horizontal displacement of the region from the center of the picture. These features are combined in a scoring function:

$$D^2 = \left(1 - \frac{A}{A_{max}}\right)^2 + \left(\frac{X - X_{min}}{60}\right)^2 ,$$

and the region for which D^2 is minimum is declared to be one face of the object. The same criterion is used to select the other visible face from the neighbors of both the floor and the first face.

The major problem remaining is to identify the vertices where the corners of the object meet the floor. This is done by processing the common boundary between the face regions and the floor region. After simple straight-line connections are made between endpoints of any gaps, this common boundary consists of a chain of points along the lower edge of the object. The lowest point on this chain is taken to be the central vertex, and the corners on either side are found by the method of iterative end-point fits.[8] Once these three image points are determined, the support hypothesis is used to locate the corresponding points on the floor. The resulting coordinates can then be entered in the model under the name of a new object if the status of the room is unknown, or under the name of the nearest object if the status is known.

C. ILA Vision Routines

The following is a summary of the intermediate-level routines related to Shakey's visual system:

CLEARPATH (X Y) decides whether the path from (AT ROBOT $*) to (X Y) is clear. In analyzing pictures, it inspects only the image of the path to be traversed, and it uses the range finder to detect large, close objects. The value returned is either CLEAR, UNKNOWN, or (BLOCKED XO YO), where (XO YO) roughly locates an obstacle.

OBLOC (OB) uses the model information about the location of object OB and the routine OBLOCF to update (AT OB $*) and (DAT OB $*).

PICBUMPED (X Y) is called when a bump occurrs at (X Y). If Shakey is in a room of known status, PICBUMPED calls PICLOC; otherwise it calls PICDETECTOB (X Y).

PICDETECTOB (X Y) uses LOBLOC to locate the object near (X Y). If Shakey is in a room of known status, and if OB is the nearest object, (AT OB $*) and (DAT OB$*) are updated; otherwise a new object is entered in the model.

PICLOC uses the landmark routine[1] to update (AT ROBOT $*), (DAT ROBOT $*), (THETA ROBOT $), and (DTHETA ROBOT $).

PICTHETA updates (THETA ROBOT $) and (DTHETA ROBOT $). Intended to be used before a long, straight-line journey, PICTHETA currently calls PICLOC.

VI PLANNING, GENERALIZATION, AND EXECUTION

A. Introduction

The basic problem-solving system used by Shakey is STRIPS, a system that makes use of a combination of heuristic search and formal deductive techniques, and that has been well described in previous reports.[3] However, STRIPS in its original form is limited to constructing a plan for solving a specific problem. In this section we describe new:

(1) Procedures for constructing "generalized" plans that are applicable to a large family of problems (in addition to the specific problem that motivated the planning process).

(2) Methods for storing, selecting, and monitoring the use of generalized plans while a task is actually being carried out.

The recently developed methods for storing and using generalized plans allow us:

(1) To store a generalized plan as a sequence of, say, n parameterized operators.

(2) To use as a single operator in a subsequent planning process many of the legal subsequences among the $2^n - 1$ subsequences of the original sequence of n operators.

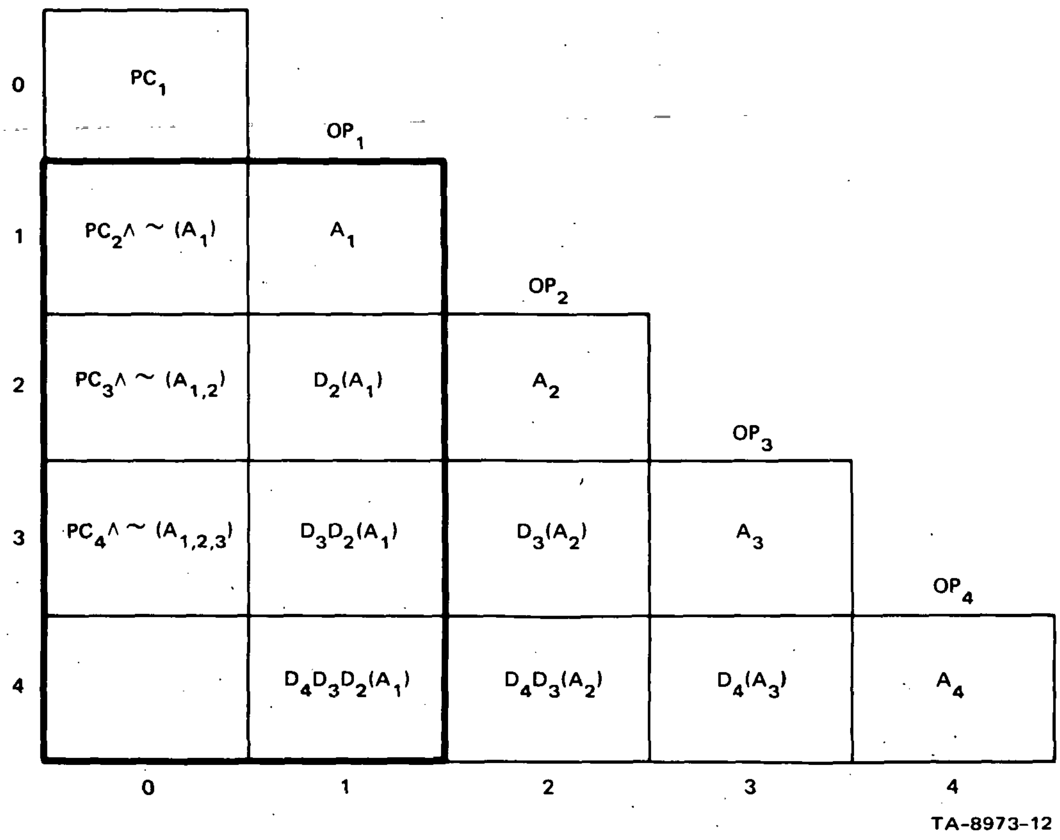
(3) To identify for monitoring purposes exactly those effects of a selected subsequence that are necessary for the success of the new plan.

As a rough illustration of the use of these capabilities, suppose that we already have a generalized plan for closing a door and turning off a light. We are now given the task of just turning off some particular light. The methods to be described will extract from the original plan the appropriate subsequence of operators needed to turn off the light. Suppose now that the subsequence of operators, or _subplan_, for turning off the light also has the effect of leaving the robot pointing in a specified direction. If this effect is a legitimate _side-effect_--that is, if the successful execution of the plan does not require the robot to be pointing in a specified direction--then the methods described will identify this fact and the final robot orientation will not be monitored during plan execution. Hence, the plan execution mechanism will not reject as "unsuccessful" an execution that has failed only in a detail irrelevant to the task at hand.

The processes for storing a generalized plan begin with the creation by STRIPS of a generalized plan, or _macro operator_--that is, a sequence of n operators whose arguments are parameters. During the creation of this plan, STRIPS performed proofs demonstrating that each operator was in fact applicable at the time it was used. We assume throughout this section the availability of both the STRIPS plan and certain information about the structure of the proofs performed by STRIPS to generate the plan. We also assume the availability of _descriptions_ of each operator used in the plan. An operator description consists of three things: a _precondition_ _formula_, which must be provable from a model if the operator is to be applied to that model; an _add-list_, specifying clauses added to the model; and a _delete_ _function_ (represented as a list of literals), which maps a set of clauses into a subset of itself that remains true after the operator has been applied.

B. Storage of a Generalized Plan

We store a generalized plan in the form of a triangular table as shown in Figure 7. The columns of the table, with the exception of



0	PC_1	OP_1			
1	$PC_2 \wedge \sim (A_1)$	A_1	OP_2		
2	$PC_3 \wedge \sim (A_{1,2})$	$D_2(A_1)$	A_2	OP_3	
3	$PC_4 \wedge \sim (A_{1,2,3})$	$D_3D_2(A_1)$	$D_3(A_2)$	A_3	OP_4
4		$D_4D_3D_2(A_1)$	$D_4D_3(A_2)$	$D_4(A_3)$	A_4
	0	1	2	3	4

TA-8973-12

FIGURE 7 TYPICAL MACROP

column 0, are labeled with the names of the operators of the plan, in this example $OP_1,\ldots,OP_4$. For each column i, i = 1,...,4, we place in the top cell the add-list A_i of operator OP_i. Going down the i^{th} column, we place in consecutive cells the portion of A_i that survives the application of subsequent operators. This is indicated by the delete function D_i, i = 2, 3, 4, that maps an add-list into the subset of itself remaining true after the application of OP_i. (The delete function D_1 of OP_1 is applied to the model in which MACROP is applied, and not to any of the add-lists.) Thus, cell (2,1) contains $D_2(A_1)$, which is the portion of A_1 still true after OP_2 is applied. Cell (3,1) contains $D_3(D_2(A_1)) = D_3D_2(A_1)$, which is the subset of A_1 that survives the application of both OP_2 and OP_3.

We can now interpret the content of the i th row of the table, excluding the first column. Since each cell in the i th row (excluding the first) contains statements added by one of the first i operators and not deleted by any of the first i operators, we see that the union of the cells in the i th row (excluding the first cell) specifies the add-list obtained by applying in sequence $OP_1,\ldots,OP_i$. We denote by $A_{1,\ldots,i}$ the add-list achieved by the first i operators applied in sequence. The union of the cells in the bottom row of a triangle table specifies the add-list of the complete macro operator.

Let us now consider the first column of the triangle table, which we have so far ignored. Loosely, the statements in row i of column zero are involved with the precondition formula PC_{i+1} of OP_{i+1}. To be more specific, cell (i,0) contains clauses needed to prove PC_{i+1} but not contained in $A_{1,\ldots,i}$. We will call the set of clauses (axioms) used to prove a formula the <u>support</u> of that formula. The clauses in cell (i,0) are therefore the portion of the support of PC_{i+1} that was true in the initial state. (In Figure 7, we have used the notion $PC_i \wedge \sim A_{1,\ldots,i}$ to indicate the contents of cell (i,0).) The remaining part of the support of PC_i is supplied by applying in sequence $OP_1,\ldots,OP_i$. The i th row of the table, then, contains the complete support of the precondition of OP_{i+1}. It is convenient to flag the clauses in row i that are the support of PC_{i+1}, and hereafter speak of marked clauses; by construction, obviously, all clauses in column zero are marked.

C. Planning with Generalized Plans

1. General Approach

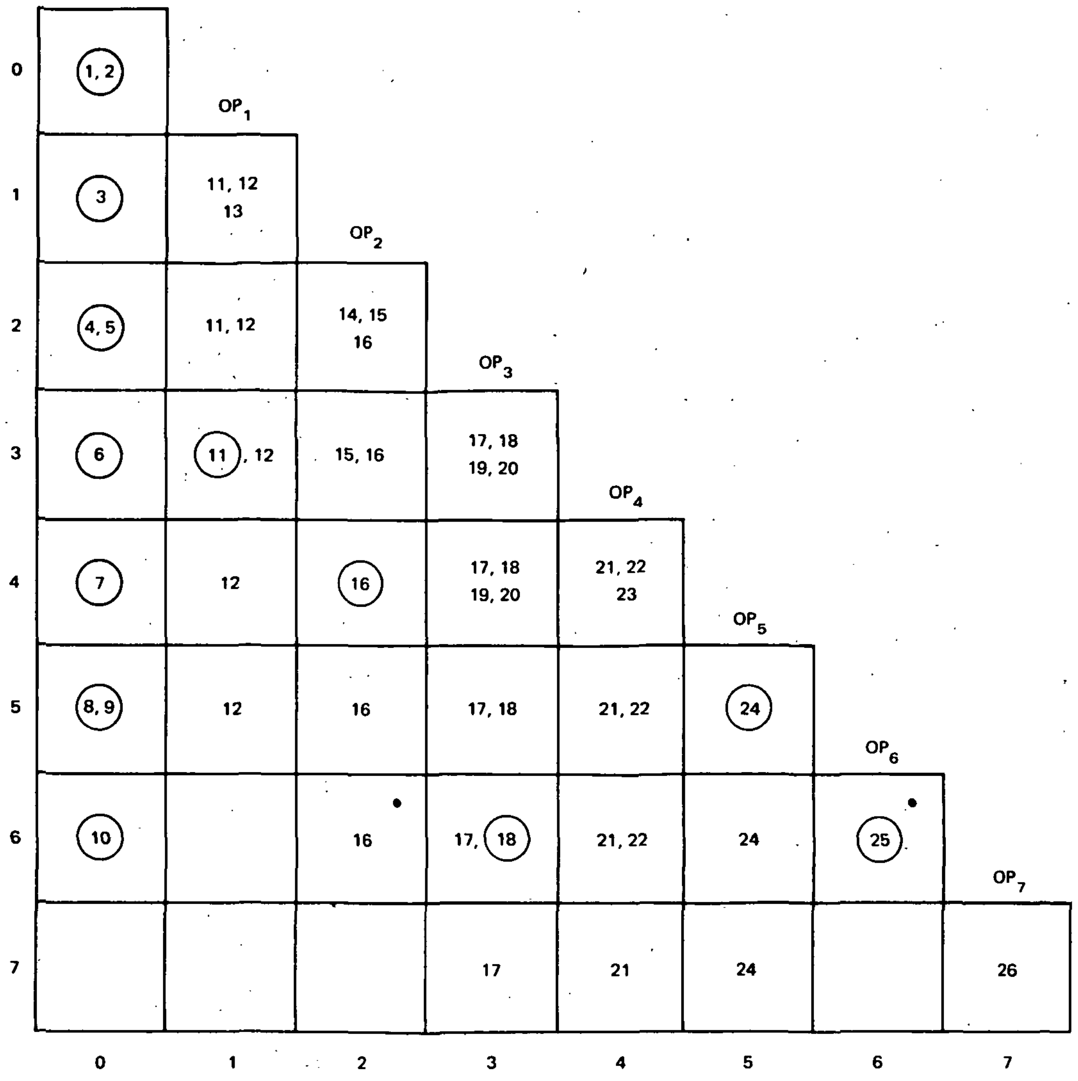
In the preceding section, we prescribed the construction of triangle tables for storing generalized plans. Now let us consider how a generalized plan will be used by STRIPS during a subsequent planning process.

The first thing to emphasize is that the i^{th} row of a triangle table (excluding its first cell) represents the add-list $A_{1,\ldots,i}$; an n-row table presents STRIPS with n alternative add-lists, any one of which can be used to reduce a difference encountered by STRIPS during its normal planning process. STRIPS selects a particular add-list in the usual fashion by testing the relevance of that add-list with respect to the difference currently being considered. Suppose for a moment that STRIPS selects the i^{th} add-list $A_{1,\ldots,i}$, $i < n$. Since this add-list is achieved by applying in sequence $OP_1,\ldots,OP_i$, we will obviously not be interested in the application of $OP_{i+1},\ldots,OP_n$, and will therefore not be interested in establishing any of the preconditions $PC_{i+1},\ldots,PC_n$. Now in general, some steps of a plan are needed only to establish preconditions for subsequent steps. If we lose interest in the <u>tail</u> of a plan--that is, in the last (n - i) operators--then we may be able to achieve some economies by omitting those operators among the first i whose sole purpose is to establish preconditions for the tail. Conceptually, then, we can think of a single triangle table as representing a family of generalized operators. Upon the selection by STRIPS of a relevant add-list, we must extract from this family an economical parameterized operator achieving the add-list. STRIPS must then be provided with a complete description--precondition wff, add-list, and delete function--of the extracted operator so that it can be used during the planning process.

In the following paragraphs, we will explain by means of an example an algorithm for accomplishing this task of operator extraction.

2. The Operator Extraction Algorithm

Consider the illustrative triangle table shown in Figure 8. Each of the numbers within cells represents a single clause. The circled clauses are "marked" in the sense described earlier; that is, they are used to prove the precondition of the operator whose name appears on the



	0	1	2	3	4	5	6	7
0	①,②	OP_1						
1	③	11, 12 13	OP_2					
2	④,⑤	11, 12	14, 15 16	OP_3				
3	⑥	⑪, 12	15, 16	17, 18 19, 20	OP_4			
4	⑦	12	⑯	17, 18 19, 20	21, 22 23	OP_5		
5	⑧,⑨	12	16	17, 18	21, 22	㉔	OP_6	
6	⑩		16	17, ⑱	21, 22	24	㉕	OP_7
7				17	21	24		26

TA-8973-13

FIGURE 8 MACROP WITH MARKED CLAUSES

same row. A summary of the structure of this plan is shown below, where "I" refers to the initial state and "F" to the final state:

Operator	Precondition Support Supplied BY	Precondition Support Supplied To
OP_1	I	OP_4
OP_2	I	OP_5
OP_3	I	OP_7, F
OP_4	I,OP_1	F
OP_5	I,OP_2	OP_6, F
OP_6	I,OP_5	OP_7
OP_7	I,OP_3,OP_6	F

Suppose now that STRIPS selects $A_{1,\ldots,6}$ as the desired add-list and, in particular, selects clause 16 and clause 25 as the particular members of the add-list that are relevant to reducing the difference of immediate interest. These clauses have been marked on the table with a dot. The operator extraction algorithm proceeds by examing the table to determine what effects of individual operators are not needed to produce clauses 16 and 25. First, OP_7 is obviously not needed; we can therefore remove all circle marks from row 6, since those marks indicate the support of PC_7. We now inspect the columns, beginning with column 6 and going from right to left, to find the first column with no marks of either kind. Column 4 is the first such column. The absence of marked clauses in column 4 means that the clauses added by OP_4 are not needed to reduce the difference and are not required to prove the precondition of any subsequent operator; hence we delete OP_4 from the plan and unmark all clauses in row 3. Continuing our right-to-left scan of the columns, we note the column 3 contains no marked clauses. (Recall that we have already unmarked clause 18.) We therefore delete OP_3 from the plan and unmark all clauses in row 2. Continuing the scan, we note that column 1 contains no marked entries (we have already unmarked clause 11), and therefore delete OP_1 and the marked entries in row 0.

The result of the table-editing process just described is shown in Figure 9. (The question mark in cell (2,1) will be explained momentarily.) A summary of the structure of this plan is shown below:

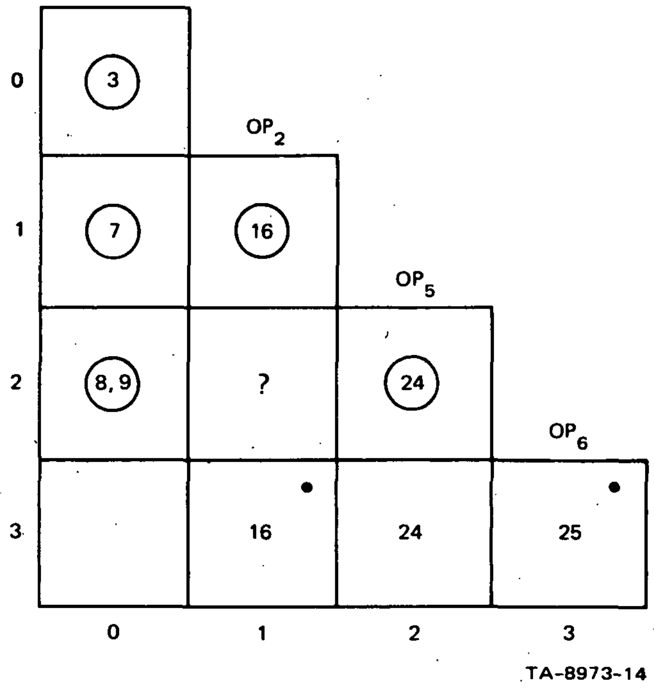


FIGURE 9 MACROP AFTER EDITING

Operator	Precondition Support Supplied By	Precondition Support Supplied To
OP_2	I	OP_5, F
OP_5	I, OP_2	OP_6
OP_6	I, OP_5	F

We have thus reduced the seven-step generalized plan we started with to a compact three-step plan that specifically produces an add-list containing the relevant clauses.

Now that an operator achieving a desired add-list has been extracted, we must provide STRIPS with its description. The precondition wff is obvious; it consists of the conjunction of all clauses in column 0. The computation of the add-list and delete function of the new operator is a little more complicated. First, notice in Figure 8 that clauses 14, 15, and 16 are added by OP_2. Clause 14 is evidently deleted by OP_3 since it does not appear in cell (3,2) The extracted plan, however, does not include OP_3, and we cannot tell whether clause 14 would survive the application of OP_5 or OP_6 in the extracted plan--hence the question mark in Figure 9. Furthermore, cell (3,1) may contain more clauses than shown. This example illustrates the necessity of computing a new add-list and delete function for the extracted operator.

The computation of a new add-list and delete function for a macro operator is based on the add-lists and delete functions of the component operators. Suppose the macro operator of Figure 9 is applied to some state S_i (in which we assume that clauses 3, 7, 8, and 9 are true). Since STRIPS does deletions before additions, we can write the resulting state S_f as:

$$S_f = D_6(D_5(D_2(S_i) + A_2) + A_5) + A_6 \quad ,$$

where we have used "+" to mean set union. Now it is not difficult to show that delete functions distribute over set union, that is, to show for any sets A and B and any delete function D that

$$D(A + B) = D(A) + D(B) \quad .$$

Hence, we can write the final state S_f as:

$$S_f = D_6 D_5 D_2 (S_i) + D_6 D_5 (A_2) + D_6 (A_5) + A_6 \quad .$$

Since this has the form $S_f = D(S_i) + A$, we see that the delete function of the macro operator is the composed function

$$D_6 D_5 D_2 \quad ,$$

and that its add-list is

$$D_6 D_5 (A_2) + D_6 (A_5) + A_6 \quad .$$

It is interesting to note that this add-list is precisely the last row of the triangle table constructed as described in the previous section, for the plan OP_2, OP_5, OP_6. In general, we can say that the add-list of a macro operator is given by the last row of its triangle table representation, and that its delete function is given by the composition of the component delete functions.

3. Refinements

In the previous paragraphs, we outlined an algorithm for extracting from a generalized plan a subsequence of operators that add particular clauses to a model. We would now like to describe two refinements: one needed to avoid certain inconsistencies that could otherwise occur, and one for achieving further economies when more than one level of triangle tables are involved.

a. Add-list refinement

Consider a simple generalized plan consisting of two consecutive PUSH operators, each of which pushes a (parameterized) object to a (parameterized) place. The triangle table for this plan might be as shown in Figure 10, where for simplicity we have assumed that the PUSH

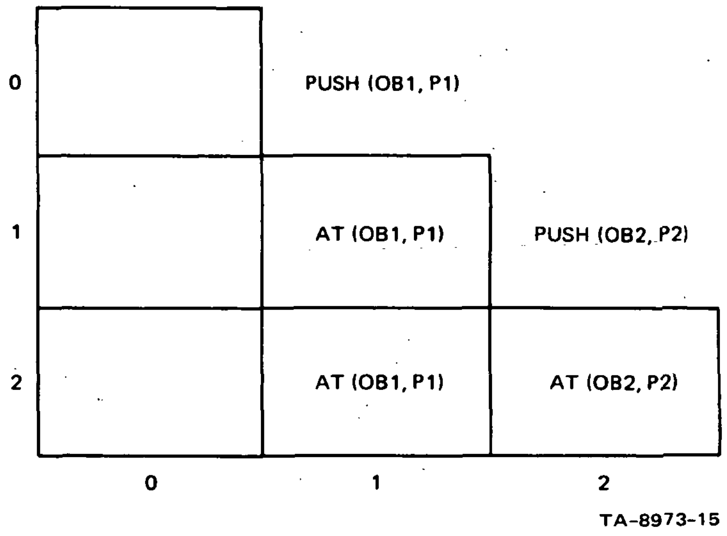


FIGURE 10 GENERALIZED PLAN FOR TWO-PUSH MACROP

operator has no precondition and hence column 0 is empty. Because the clause AT(OB1,P1) appears in cell (2,1), we know that this clause was not deleted by the second PUSH operator. Suppose now that STRIPS selects row 2 as an add-list. By instantiating OB1 and OB2 to the same object name, and instantiating P1 and P2 to distinct locations, we evidently have a plan for achieving a state in which the same object is simultaneously at two different places! The source of this embarrassment lies in the delete mechanism used by STRIPS, which we now examine in some detail.

The delete function of an arbitrary STRIPS operator is specified by a <u>delete-list</u> consisting of a set of literals. If the operator is applied to a state S, then STRIPS deletes from S every clause containing a literal unifying (without regard to sign) with any member of the delete-list. If a potential unification involves parameters, as it often does, then the unification can be made only if it does not contradict any existing bindings of the parameters to constants. To continue our example, suppose the second push operator is applied to the parameterized state S:

AT(OB1, P1)
AT(OB2, P3).

The delete-list of the second push operator, we assume, contains the single literal AT(OB2, \$), where "\$" unifies with anything. If there were no existing bindings of parameters to constants, then both clauses in S would be deleted. From Figure 10, to the contrary, we see that AT(OB1, P1) was not deleted; hence, it must have been the case that OB1 and OB2 represented distinct objects in the unparameterized problem for which the plan was originally created. If in a subsequent attempt to use this plan we set OB1 = OB2, then we are violating the constraint responsible for the occurrence of AT(OB1, P1) in the final state. Accordingly, we replace the entry in cell (2,1) of Figure 10 by the new entry:

$$(\mathrm{OB1} \neq \mathrm{OB2}) \supset \mathrm{AT}(\mathrm{OB1},\mathrm{P1}) \quad .$$

By this means we indicate that row 2, and cell (2,1) in particular, produces the literal AT(OB1, P1) only under the condition that OB1 and OB2 are not instantiated to the same constant.

The previous example illustrates how a literal can be allowed to survive the application of a delete function only under some condition on the bindings of its arguments. We introduced this notion in the context of maintaining the validity of a triangle table, but it is more broadly applicable within the general framework of STRIPS. Although it is an enlargement on our main theme of storing and using generalized plans, let us briefly consider how the notion of conditional survival of a literal can be exploited.

During the planning process, STRIPS frequently permits a delete function to delete true clauses from a state description. To overcome this tendency toward excessive deletions, we make use of the notion of conditional survival as defined by the following algorithm.

Let L(P1) be a literal in a parameterized state description, and suppose that the deletion of the clause containing this literal depends on binding parameter P1 to another parameter P2. Then:

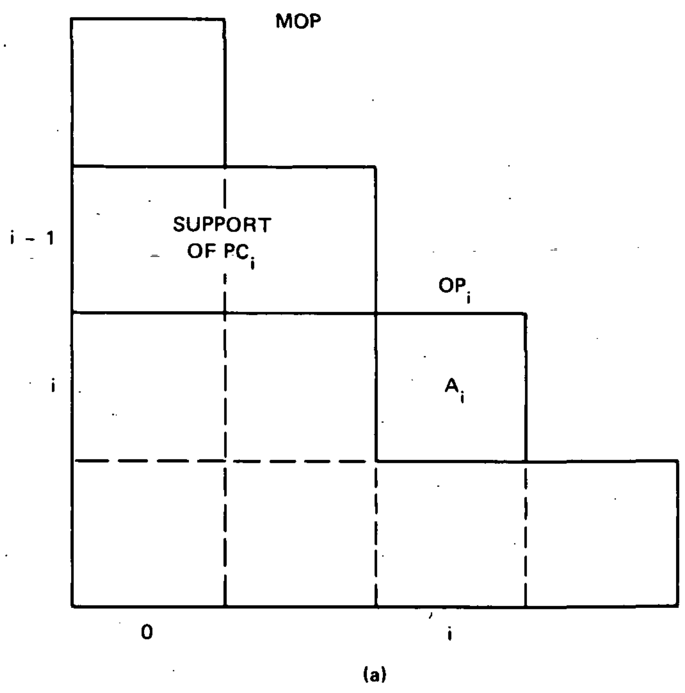
> If P1 or P2 has no constant binding then replace L(P1) by $P1 \neq P2 \supset L(P1)$. (In "standard" STRIPS this clause would simply be deleted.)
>
> If P1 and P2 both represent the same constant in the original problem, then delete the clause containing L(P1). (This is what STRIPS does as a standard operation.) In the appropriate cell of the triangle table, place $P1 \neq P2 \supset L(P1)$. (This generalizes the triangle table beyond the planning states used by STRIPS.) If P1 and P2 represent distinct constants in the original problem, then replace L(P1) by $P1 \neq P2 \supset L(P1)$. (This is the case illustrated by our previous example.)

We should note that the inclusion in a table of such clauses as, say, $P1 \neq P2 \supset L(P1)$ leads to certain complications. Suppose, in a subsequent problem, that STRIPS uses such a clause in the proof of some precondition. Often, the proof will produce the unit clause $P1 = P2$. In this case, we consider the proof completed by assuming $P1 \neq P2$ (providing the assumption contradicts no existing bindings). However, we must record this assumption by placing $P1 \neq P2$ in column 0 of the table being constructed; it is, after all, now a hypothesis of the theorem. Moreover, all subsequent proofs in the new plan must not violate this hypothesis. As a bookkeeping procedure, we can conjoin the assumption (viz., $P1 \neq P2$) to each new precondition that STRIPS attempts to prove; this has the effect of preventing violations of our assumption.

b. Relaxing Preconditions in Nested Tables

Consider the situation shown in Figures 11(a) and (b), where we have shown a macro operator MOP whose i^{th} operator is itself the *macro* operator OP_i. As always, cell (i, i) of MOP contains the complete add-list of OP_i, while the marked entries of Row (i - 1) constitute the support of the proof of the preconditions of OP_i. During the planning process, suppose STRIPS selects from one of the rows of MOP certain clauses it would like to add to the current state of the world. Suppose further that some, but not all, of the clauses in cell (i.i) of Figure 11(a) are marked. We can therefore mark in Figure 11(b) those clauses in A_i that are needed, and exercise the operator extraction algorithm on table OP_i. As we saw earlier, this will at times result in the deletion of some of the clauses from PC_i. Suppose, then, that some of the clauses of PC_i are in fact deleted by the operator extraction algorithm. This raises the possibility of deleting some of the clauses in the support of PC_i since they now need to support only a weaker theorem. If the support of PC_i can be weakened--that is, if some of the clauses in row (i - 1) can be unmarked--than in general we may be able to delete more steps from MOP and/or obtain weaker, more easily established, preconditions for MOP.

In order for this scheme of precondition relaxation to be feasible, we need an economical solution to the following abstractly stated problem: Given that a set of clauses $C_1,\ldots,C_k$ implies a theorem $T_1 \cap \ldots \cap T_m$, which C_i's can be deleted from the premises if a selected subset of the T_i's are deleted from the theorem? Fortunately, it is



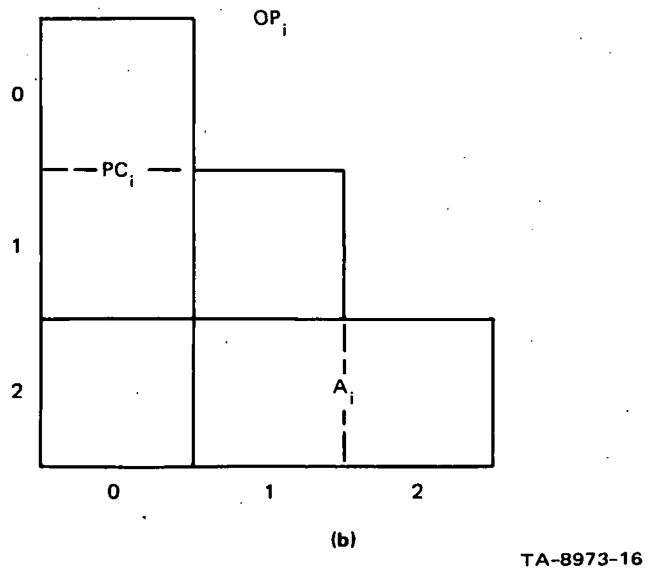


FIGURE 11 MOP: A NESTED MACROP

possible to solve this problem by appropriately labeling literals during the refutation proof of the theorem. We will not elaborate here on the details of this bookkeeping procedure. In terms of the example of Figures 11(a) and (b) the important point is that the bookkeeping need be done only once, namely, when PC_i is shown to be a consequence of its support. Thereafter, it is a straightforward matter to compute, without recourse to theorem proving, the appropriate relaxation of the support of PC_i given a relaxation of PC_i itself.

The ability to relax preconditions leads to an obvious refinement of the operator extraction algorithm described earlier. Previously we unmarked clauses only when a component operator was deleted from a macro operator, in which case the entire support of the precondition of that operator was unmarked. Now we can also unmark a subset of the support of a component operator still retained in the macro operator. Finally we remark that although Figure 11 shows only two levels of tables, the procedure for relaxing preconditions can be implemented recursively; hence; nested tables to arbitrary depth can be readily processed.

D. Monitoring the Execution of Plans

In this section we outline an algorithm for using triangle tables to monitor the real-world execution of generalized plans. An important feature of the algorithm is that it monitors only those effects of operators, and only those aspects of the world, relevant to the problem solution. Additionally, the algorithm embodies a modest replanning capacity in the form of an ability to reinstantiate parameters of operators.

The plan execution algorithm rests on the observation that a triangle table contains complete information about the internal structure of the plan it represents. More specifically, a triangle table specifies exactly what each operator accomplishes in terms of providing support for the preconditions of subsequent operators or the goal statement. Equivalently, a triangle table also specifies the conditions that must obtain in order for a component operator to be applicable.* The plan execution algorithm to be described uses this information in a straightforward manner.

Important information about the internal structure of a plan is embodied in the <u>kernels</u> of a triangle table. The i^{th} kernel of a triangle table for an n-step plan is the largest rectangular subarray containing cells (n,0) and cell (i-1,i-1). In Figure 7, by way of an example, we have outlined the second kernel of MACROP. The importance of the i^{th} kernel stems from the fact that it contains the support of the preconditions for the tail of the plan--that is, for the operator sequence $OP_i, \ldots, OP_n$. This should be clear, since row J of the i^{th} kernel contains that portion of the support of PC_{j+1} that must already be true when OP_i is executed. To continue with the example of Figure 7, cells (2,0) and (2,1) contain those axioms in PC_3 that are presumably true before OP_2 is executed. If any of these axioms are false, then even perfect execution of OP_2 will not result in a state in which OP_3 is applicable. Roughly speaking, then, a reasonable plan execution algorithm should find the highest indexed kernel with all true entries and execute the corresponding component operator.

Such an algorithm would reflect the heuristic that it is best to execute the "legal" operator least removed from the goal.

*Strictly speaking, a triangle table specifies the support for the particular proof of a precondition found by STRIPS. In general, there are many possible supports for a given precondition, but we would not expect a plan execution algorithm to be cognizant of them.

An important refinement of the rough execution algorithm outlined above can be obtained by noting that the ith kernel contains in general not only those clauses supporting proofs of preconditions but many additional clauses as well. These additional clauses are irrelevant to the problem at hand, and we would certainly want our execution algorithm to ignore them. The identification of relevant clauses is easily accomplished using the operator extraction algorithm previously described. The final row of the table representing a plan to be executed contains the support of the goal formula, and the supporting clauses are marked as before. The operator extraction algorithm then produces a new operator for achieving those clauses. (We dispense with the computation of precondition formula, add-list, and delete function.) Typically, but not necessarily, all the component operators will be retained. More importantly, only some of the table entries will be marked, and these are the only portions of the kernels that need be monitored.

The task of finding an efficient algorithm for finding the "highest true kernel"--that is, the highest indexed kernel with all marked clauses true--is of some interest in itself. Our algorithm for finding the highest true kernel involves a cell-by-cell scan of the triangle table. Each cell examined is evaluated as either True (i.e., all the marked clauses are true in the current model) or False. The interest of the algorithm stems from the order in which cells are examined. Let us call a kernel "potentially true" at some stage in the scan if all evaluated cells of the kernel are true. The scan algorithm can then be succinctly stated as:

> Among all unevaluated cells in the highest-indexed potentially true kernel, evaluate the left-most. Break "left-most ties" arbitrarily.

The reader can verify that, roughly speaking, this table-scanning rule results in a left-to-right, bottom-to-top scan of the table. However, the table is never scanned to the right of any cell already evaluated as false. An equivalent statement of the algorithm is "Among all unevaluated cells, evaluate the cell common to the largest number of potentially true kernels. Break ties arbitrarily." We conjecture that this scanning algorithm is optimal in the sense that it evaluates, on the average, fewer cells than any other scan guaranteed always to find the highest true kernel. A proof of this conjecture has not been found.

The plan execution algorithm described above is embodied in a computer program named PLANEX.[3] When PLANEX is called to execute a table, it executes the component operator associated with the highest true kernel. Typically, but not necessarily, this will be OP_1. When OP_1 completes its action, PLANEX rescans the table to find the highest currently true kernel. Typically, but not necessarily, this will be Kernel 2, in which case OP_2 is executed, and so forth, until the goal kernel is reached. We emphasize, however, that after each operator execution PLANEX may either call an earlier operator (perhaps to rectify an error) or skip to a later operator (perhaps a stroke of luck rendered some operators unecessary). Furthermore, many arguments of predicates in the table are parameters; PLANEX is free to instantiate these parameters in order to find a true instance of the predicate. Thus, PLANEX is really searching for the highest-indexed kernel an instance of which is satisfied by the current state of the world. This ability of PLANEX to instantiate--and reinstantiate--arguments provides a modest replanning capacity. If the turn of world events produces a situation in which a solution has the same form as a tail of the original plan, PLANEX will find it. If no tail of the plan is applicable, then no kernel will be true, and PLANEX returns control to STRIPS to replan.

VII PUBLICATIONS AND PRESENTATIONS

A. Publications

The following technical notes and papers were written, presented, or published during the past year by the staff of the Artificial Intelligence Group of Stanford Research Institute with the support of this project:

1. D. Luckham and N. J. Nilsson, "Extracting Information from Resolution Proof Trees," Artificial Intelligence, Vol. 2, No. 1, pp. 27-54 (North-Holland Publishing Company, Amsterdam, 1971).

2. L. S. Coles, "An Experiment in Robot Tool Using," a paper presented at the IEEE Systems Science and Cybernetics Conference, Pittsburgh, Pennsylvania, October 14-16, 1970. (Abstract published in the Proceedings.)

3. J. F. Rulifson, R. J. Waldinger, and J. A. Derksen, "QA4 Working Paper," Technical Note 42, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (October 1970).

4. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," paper presented at the Second International Joint Conference on Artificial Intelligence, London, England, September 1-3, 1971. (To be published in Artificial Intelligence.)

5. C. R. Brice and J. A. Derksen, "A Heuristically Guided Equality Rule in a Resolution Theorem Prover," paper presented at the Second International Joint Conference on Artificial Intelligence, London, England, September 1-3, 1971.

6. R. O. Duda, "Some Current Techniques for Scene Analysis," Technical Note 46, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (October 1970).

7. R. E. Kling, "A Paradigm for Reasoning by Analogy," paper presented at the Second International Joint Conference on Artificial Intelligence, London, England, September 1-3, 1971.

8. J. F. Rulifson, R. J. Waldinger, and J. A. Derksen, "A Language for Writing Problem-Solving Programs," paper presented at the IFIP Congress '71, Ljubljana, Yugoslavia, August 23-28, 1971. (To be published in the Proceedings.)

9. R. E. Kling, "Reasoning by Analogy as an Aid to Heuristic Theorem Proving," paper presented at IFIP Congress '71, Ljubljana, Yugoslavia, August 23-28, 1971. (To be published in the Proceedings.)

10. L. S. Coles, "The Application of Theorem Proving to Information Retrieval," Technical Note 51, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (January 1971).

11. R. E. Fikes, "Failure Tests and Goals in Plans," Technical Note 53, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (March 1971).

12. K. Masuda, "ISUPPOSEW--A Computer Program that Finds Regions in the Plan Model of a Visual Scene," Technical Note 54, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (March 1971).

13. R. E. Fikes, "Monitored Execution of Robot Plans Produced by STRIPS," paper presented at IFIP Congress '71, Ljubljana, Yugoslavia, August 23-28, 1971. (To be published in the Proceedings.)

14. J. H. Munson, "Robot Planning, Execution, and Monitoring in an Uncertain Environment," paper presented at the Second International Joint Conference on Artificial Intelligence, London, England, September 1-3, 1971.

B. Presentations

The following presentations were made by staff members of the AI Group during the period of this contract:

1. L. S. Coles, "Experiments in Robot Tool Using," IEEE Systems Science and Cybernetics Conference, Pittsburgh, Pennsylvania, October 15-16, 1970.

2. L. S. Coles, "Experiments in Robot Tool Using," University of Waterloo, Toronto, Canada, October 20, 1970.

3. L. S. Coles, Robotics seminar, San Fernando Valley State College, School of Engineering, Northridge, California, January 12, 1971.

4. L. S. Coles, "The State of the Art of Intelligent Robots," Third Annual Houston Conference on Computers and Systems Sciences, April 25, 1971.

5. P. E. Hart, "An Introduction to Automatic Scene Analysis," Jet Propopulsion Laboratory, Pasadena, California, August 23, 1971.

6. N. J. Nilsson, "Robots and Artificial Intelligence," Brown University, Providence, Rhode Island, April 15, 1971.

7. N. J. Nilsson, "Robots and Artificial Intelligence, University of Connecticut, Storrs, Connecticut, April 16, 1971.

8. N. J. Nilsson, "Robot Research at SRI," University of California at Los Angeles, May 4, 1971.

9. B. Raphael, "Robots and Artificial Intelligence," Jet Propulsion Laboratory, Pasadena, California, June 24, 1971.

10. B. Raphael, "Artificial Intelligence and Robots," Institute for Defense Analysis, Arlington, Virginia, June 18, 1971.

11. B. Raphael, "State of the Art in Robotics," NASA Headquarters, Washington, D.C., February 24, 1971.

12. C. A. Rosen, "Developing a Computer-Controlled Mobile Vehicle with Autonomous Characteristics," Design Engineering Conference, New York, New York, April 20, 1971.

13. C. A. Rosen, "An Experimental Mobile Automaton," American Nuclear Society 18th Conference on Remote Systems Technology, Washington, D.C., November 18, 1971.

REFERENCES

1. B. Raphael, "Research and Applications--Artificial Intelligence," Final Report, SRI Project 8259, Stanford Research Institute, Menlo Park, California (November 1970).

2. B. Raphael et al., "Research and Applications--Artificial Intelligence," Semiannual Progress Report, SRI Project 8973, Stanford Research Institute, Menlo Park, California (April 1971).

3. R. E. Fikes, "Monitored Execution of Robot Plans Produced by STRIPS," Technical Note 55, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (April 1971). Also, to appear in Proc. IFIP Congress '71, Ljubljana, Yugoslavia, August 23-28, 1971.

4. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Technical Note 43R, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (May 1971). Also, to appear in Artificial Intelligence (North-Holland Publishing Company, Amsterdam, The Netherlands).

5. L. S. Coles, "Talking with a Robot in English," Proc. International Joint Conference on Artificial Intelligence, Washington, D.C., May 7-9, 1969 (The MITRE Corporation, Bedford, Mass., 1969).

6. B. A. Galler and A. J. Perlis, A View of Programming Languages, "Markov Algorithms," pp. 3-11 (Addison-Wesley Publishing Company, Inc., 1970).

7. C. R. Brice and C. L. Fennema, "Scene Analysis Using Regions," Artificial Intelligence, Vol. 1, pp. 205-226 (1970).

8. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, pp. XII-21, 23, Stanford Research Institute, Menlo Park, California (December 1970).