Robot Learning

15-494 Cognitive Robotics
David S. Touretzky &
Ethan Tira-Thompson

Carnegie Mellon
Spring 2008

What Can Robots Learn?

- Parameter tuning, e.g., for a faster walk
- Perceptual learning: ALVINN driving the Navlab
- Map learning, e.g., SLAM algorithms
- Behavior learning; plans and macro-operators
  - Shakey the Robot (SRI)
  - Robo-Soar
- Training by operant conditioning (reinforcement)
  - Skinnerbots

AIBO Walk Optimization

- How fast can an AIBO walk? Figures from Kohl & Stone, ICRA 2004, for the ERS-210 model:

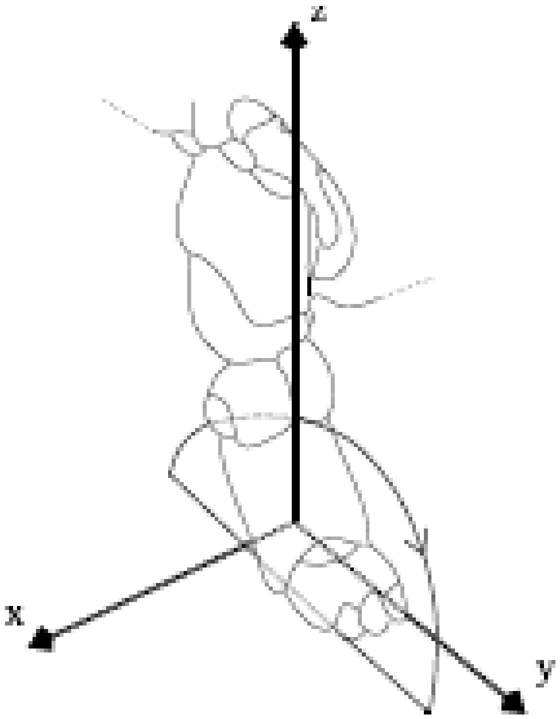
CMU (2002)	200 mm/s	Hand-tuned gaits
German Team	230 mm/s	
UT Austin Villa	245 mm/s	
UNSW	254 mm/s	

Hornsby (1999)	170 mm/s	Learned gaits
UNSW	270 mm/s	
UT Austin Villa	291 mm/s	

Walk Parameters

12 parameters to optimize:

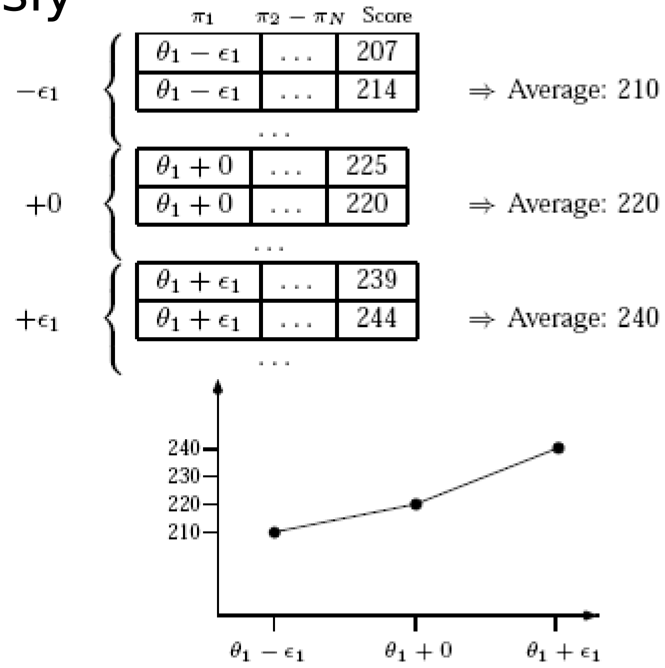
- Front locus (height, x pos, ypos)
- Rear locus (height, x pos, y pos)
- Locus length
- Locus skew multiplier
  (in the x-y plane, for turning)
- Height of front of body
- Height of rear of body
- Foot travel time
- Fraction of time foot is on ground



From Kohl & Stone (ICRA 2004)

Optimization Strategy

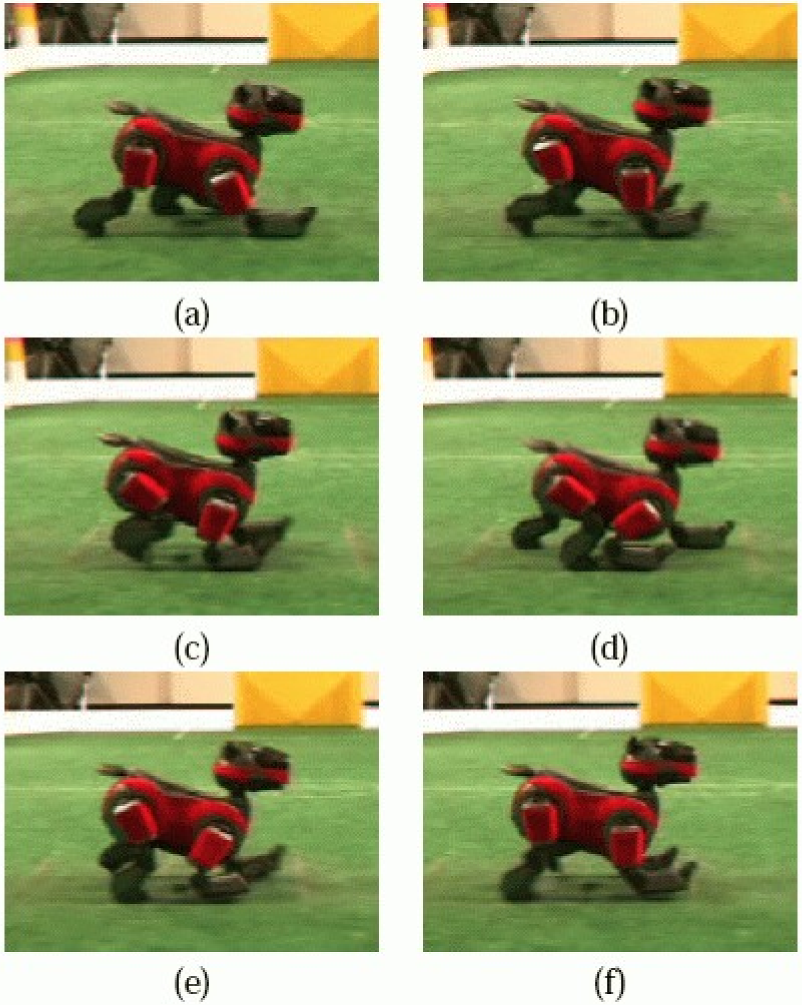
- "Policy gradient reinforcement learning":
  - Walk parameter assignment = "policy"
  - Estimate the gradient along each dimension by trying combinations of slight perturbations in all parameters
  - Measure walking speed on the actual robot
  - Optimize all 12 parameters simultaneously
  - Adjust parameters according to the estimated gradient.



	π_1	$\pi_2 - \pi_N$	Score	
$-\epsilon_1$	$\theta_1 - \epsilon_1$	...	207	
	$\theta_1 - \epsilon_1$	...	214	⇒ Average: 210
	...			
$+0$	$\theta_1 + 0$	...	225	
	$\theta_1 + 0$	...	220	⇒ Average: 220
	...			
$+\epsilon_1$	$\theta_1 + \epsilon_1$	...	239	
	$\theta_1 + \epsilon_1$	...	244	⇒ Average: 240
	...			

Kohl & Stone Results

- Used three robots
  - 23 iterations
  - 1000 total runs
  - elapsed time 3 hours
- Final speed 291 mm/s: faster than any other AIBO walk
- Videos: initial walk (clumsy 150 mm/s), final walk


(a) (b) (c) (d) (e) (f)

Kohl & Stone Results

- Initial result was for optimizing walking speed alone.
- But stability is also important:
  - Bouncy walk makes vision hard

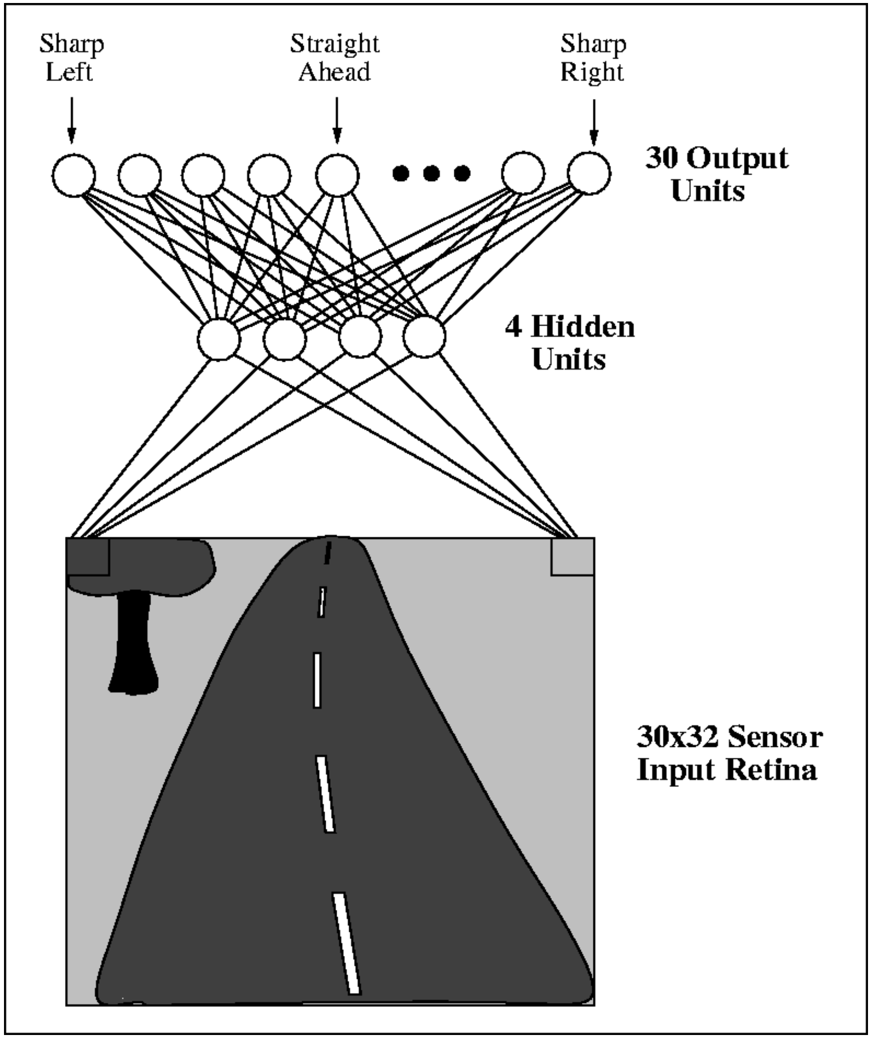
- Later experiments optimized for a combination of speed and stability.
- Also applied the technique to the ERS-7

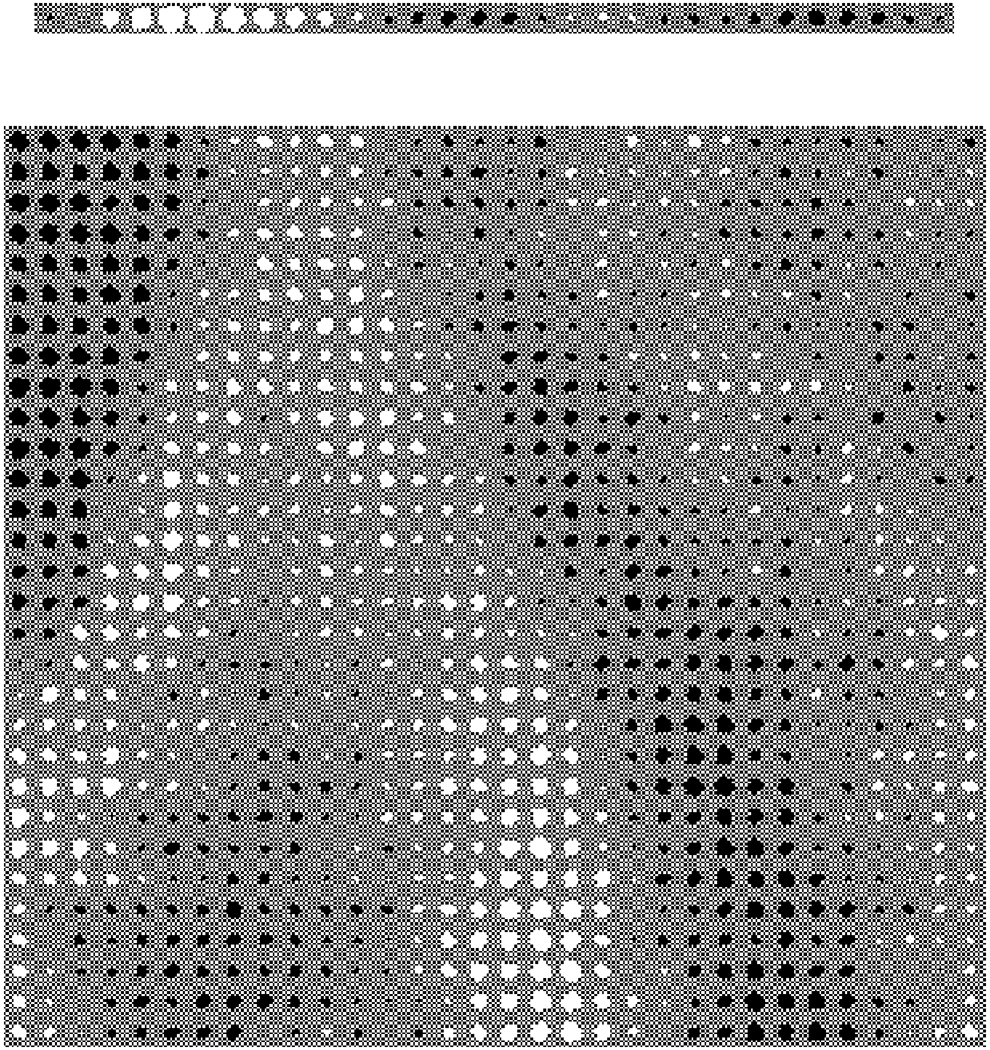
(videos)

Perceptual Learning

- ALVINN (Autonomous Land Vehicle in a Neural Network) learns to drive the Navlab

ALVINN


Sharp Left
Straight Ahead
Sharp Right
30 Output Units
4 Hidden Units
30x32 Sensor Input Retina



ALVINN Training

- Watched a human drive for a few minutes
- Used clever techniques to expand the training set
- Maintained a pool of 200 training images
- Trained on the fly

168

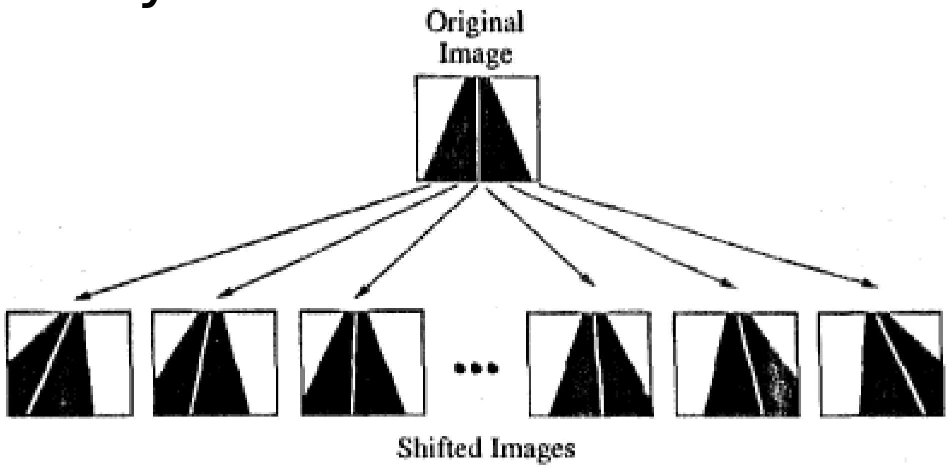
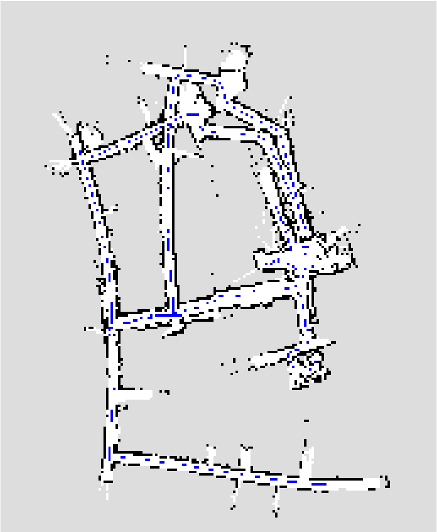
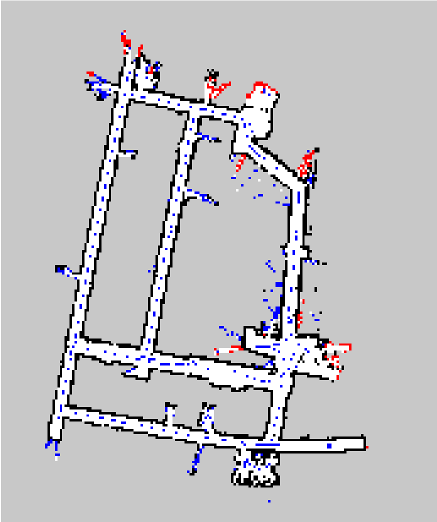


Figure 5.3.6
Shifted video images from a single original video image used to enrich the training set used to train ALVINN. (From D. A. Pomerleau, 1991, with permission of the MIT Press.)

Map Learning



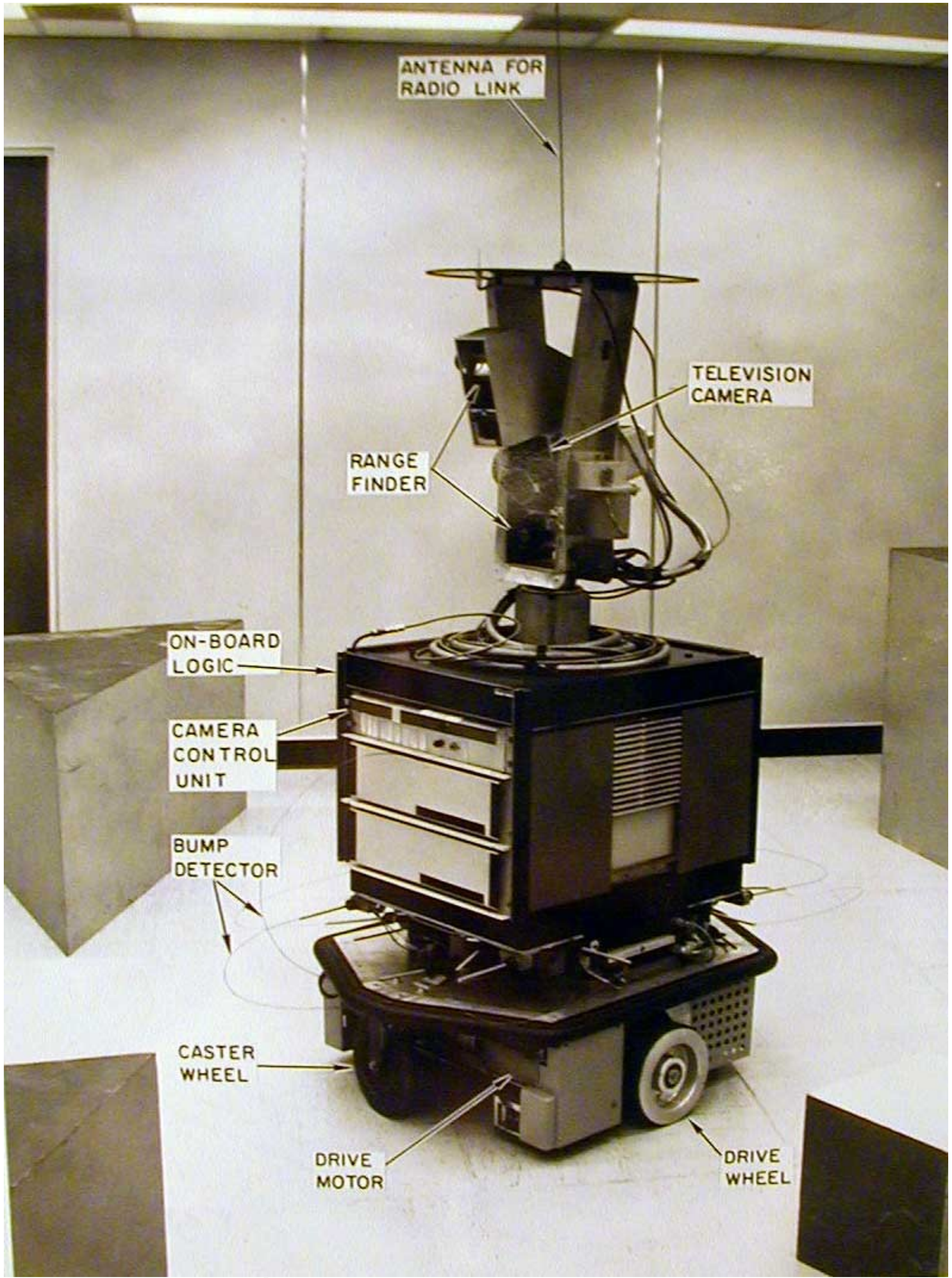
- Lots of work on learning maps
  - from sonar data
  - from laser rangefinder data
  - using visual landmarks
- Dieter Fox et al., particle filters for map learning



- SLAM: Simultaneous Localization and Mapping
  - many algorithms, all based on probabilistic approaches like particle filters

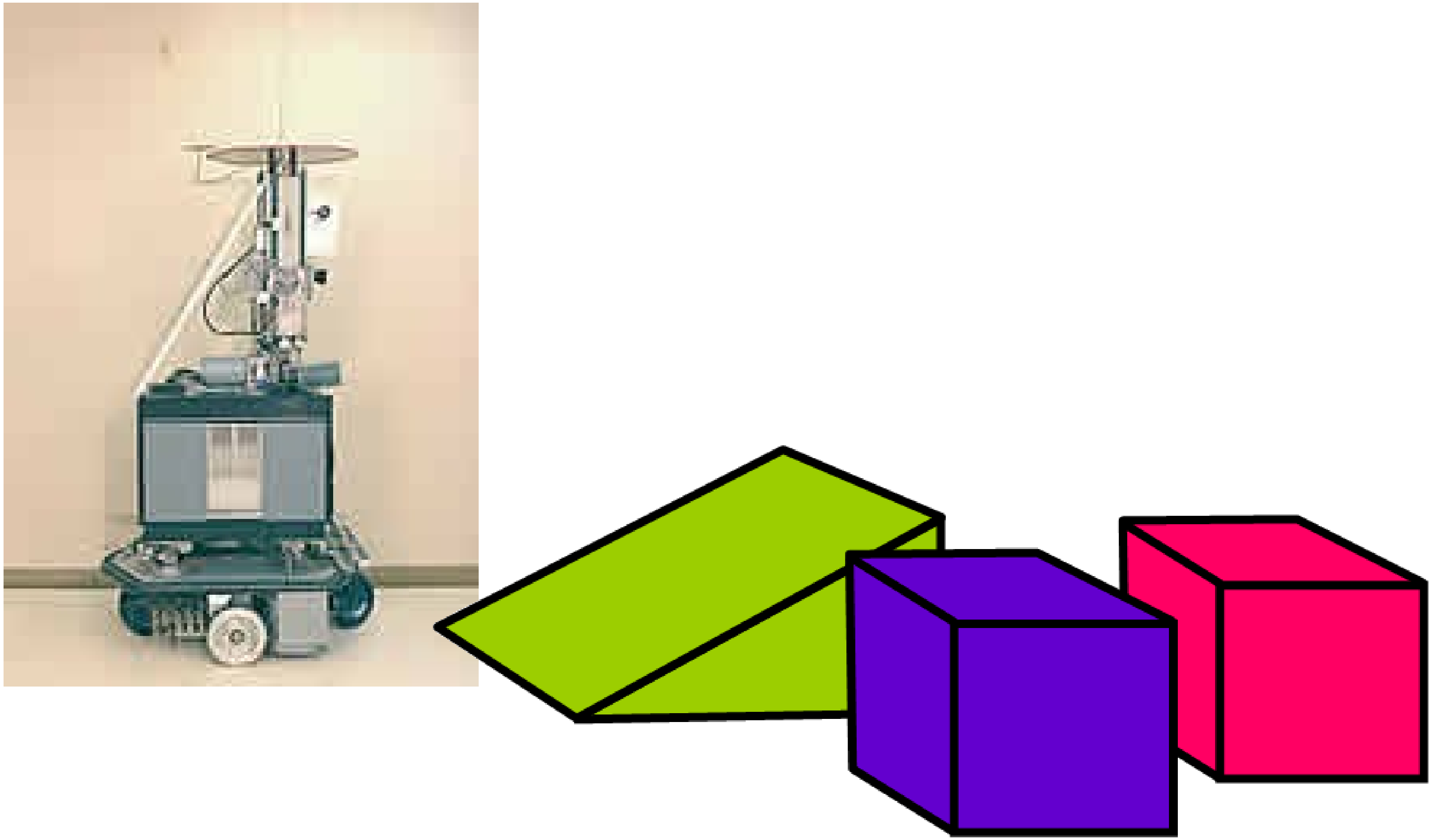
Shakey the Robot

- SRI International, 1968-1972
- Remote controlled by a PDP-10
- Programmed in Lisp and a theorem proving planner called STRIPS



Learning Blocks World Plans

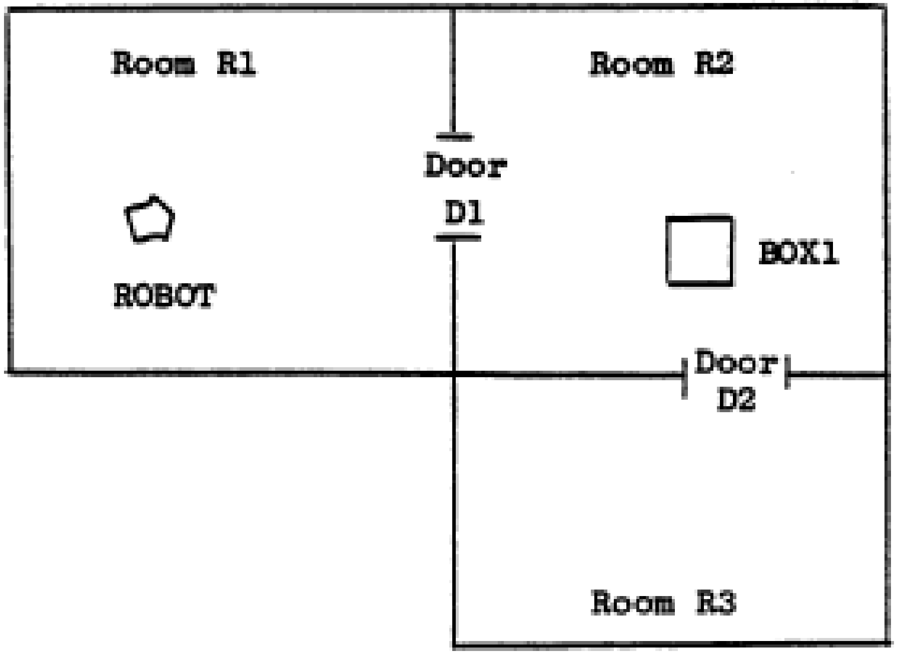
- Shakey learned plans for manipulating blocks by pushing them around.



Sample Problem

Initial model:

```
INROOM(robot,R1)
CONNECTS(D1,R1,R2)
CONNECTS(D2,R2,R3)
BOX(BOX1)
INROOM(BOX1,R2)
(∀x∀y) [CONNECTS(d,x,y) →
         CONNECTS(d,y,x)]
```



Goal:

```
(∃x)[BOX(x) & INROOM(x,R1)]
```

Available operators:

```
GOTHRU(d,r1,r2)
PUSHTHRU(b,d,r1,r2)
```

GOTHRU Preconditions; Add and Delete Lists

- Operator: GOTHRU(d,r1,r2)
- Precondition:
  INROOM(ROBOT,r1) & CONNECTS(d,r1,r2)
- Delete list: INROOM(ROBOT, $)
- Add list: INROOM(ROBOT, r2)

PUSHTHRU Preconditions; Add and Delete Lists

- Operator: PUSHTHRU(b,d,r1,r2)
- Precondition:
  INROOM(b, r1) & INROOM(ROBOT,r1)
  & CONNECTS(d,r1,r2)
- Delete list: INROOM(ROBOT, $), INROOM(b, $)
- Add list: INROOM(ROBOT, r2), INROOM(b, r2)

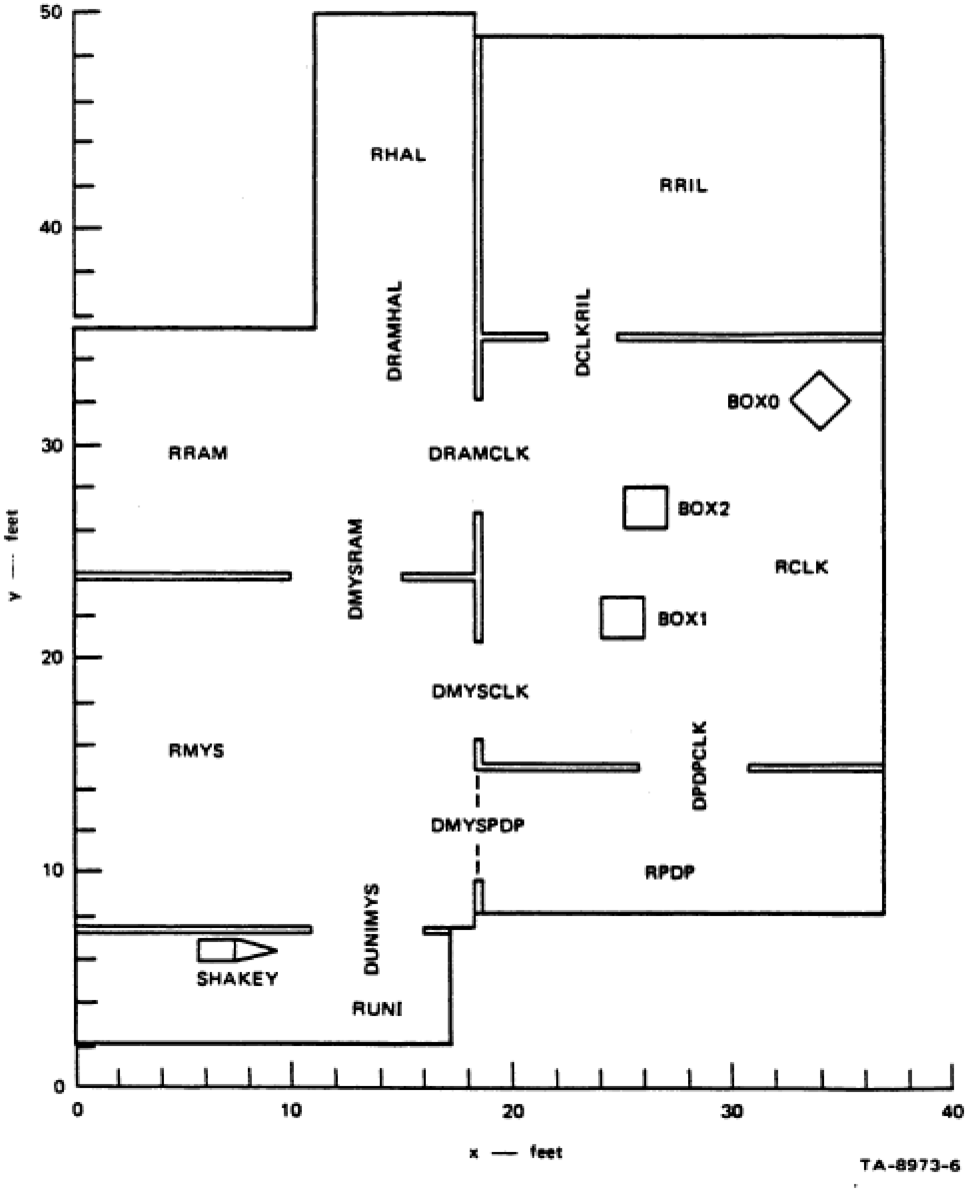
Solving the Problem

- Goal G0: (∃x) [`BOX(x) & INROOM(x,R1)`]
  - `Not satisfied in current model`
  - `But PUSHTHRU could make INROOM(BOX1,R1) true`
- `Precondition for PUSHTHRU gives subgoal G1:`
  `INROOM(BOX1,r1) & INROOM(ROBOT,r1)`
  `&  CONNECTS(d,r1,R1)`
  - `Not satisfied in current model`
  - `But if r1=R2 and d=D1, the operator could apply`
  - `Need INROOM(ROBOT,R2)`
- `Precondition for GOTHRU gives subgoal G2:`
  `INROOM(ROBOT,r1)  &  CONNECTS(d,r1,R2)`

Solving the Problem

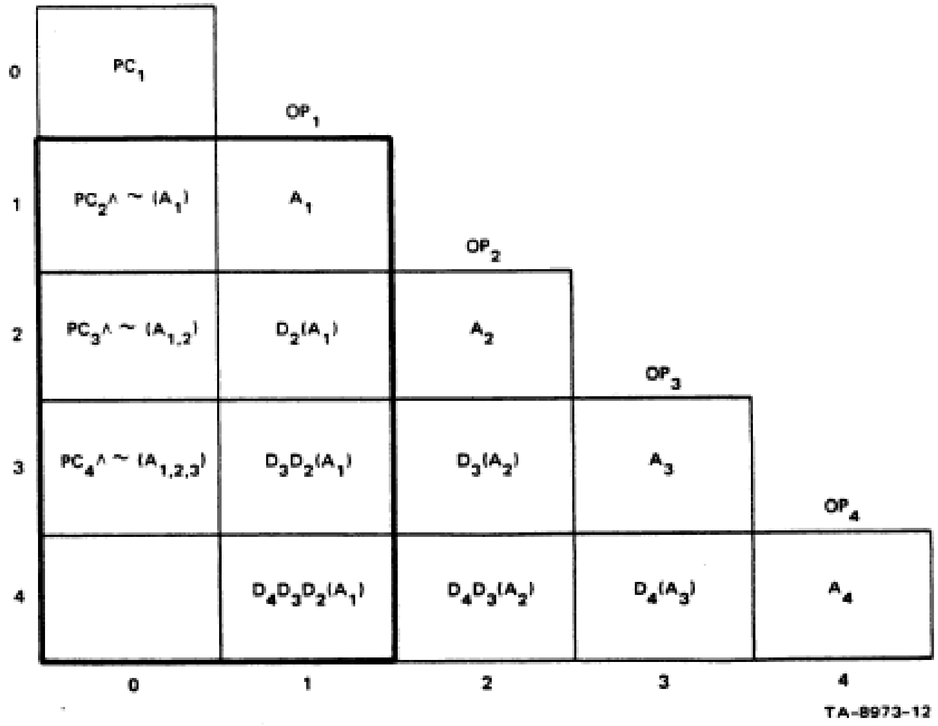
- Apply operator GOTHRU(D1,R1,R2)
- New model M1:
  ```
  INROOM(ROBOT,R2)
  CONNECTS(D1,R1,R2)
  CONNECTS(D2,R2,R3)
  BOX(BOX1)
  INROOM(BOX1,R2)
  ```
- `Apply operator PUSHTHRU(BOX1,D1,R2,R1)`
- `Goal G0 has now been achieved.`

Shakey's Full Environment



Macro Operators (MACROPs)

- Idea: extract general plans from solutions to specific problems.
- Reuse those plans in novel contexts by rebinding variables and/or deleting irrelevant steps.
- “Triangle table” defines additions/deletions for each operator in a plan.
- Reasoning algorithm determines which clauses are relevant to the new plan.

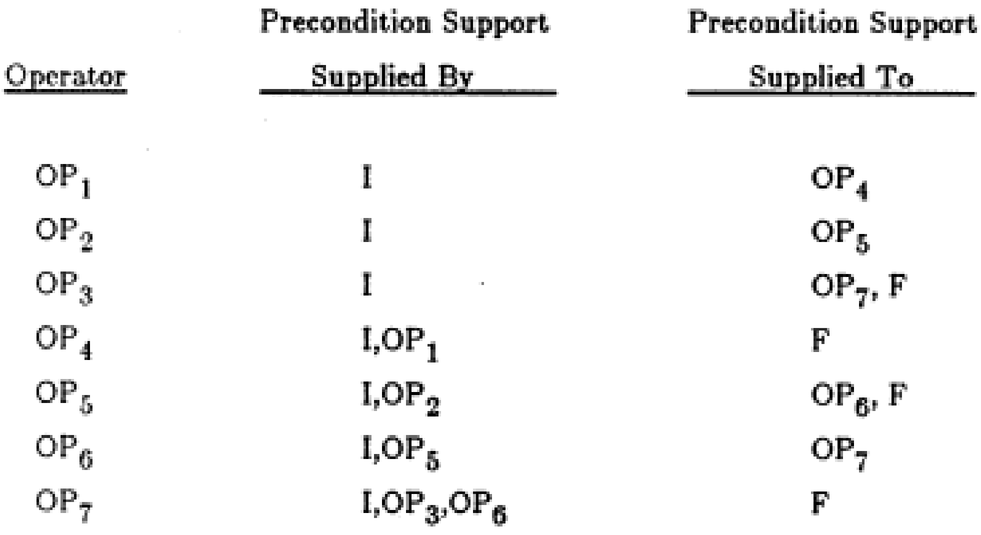


0	PC_1	OP_1			
1	$PC_2 \wedge \sim (A_1)$	A_1	OP_2		
2	$PC_3 \wedge \sim (A_{1,2})$	$D_2(A_1)$	A_2	OP_3	
3	$PC_4 \wedge \sim (A_{1,2,3})$	$D_3D_2(A_1)$	$D_3(A_2)$	A_3	OP_4
4		$D_4D_3D_2(A_1)$	$D_4D_3(A_2)$	$D_4(A_3)$	A_4
	0	1	2	3	4

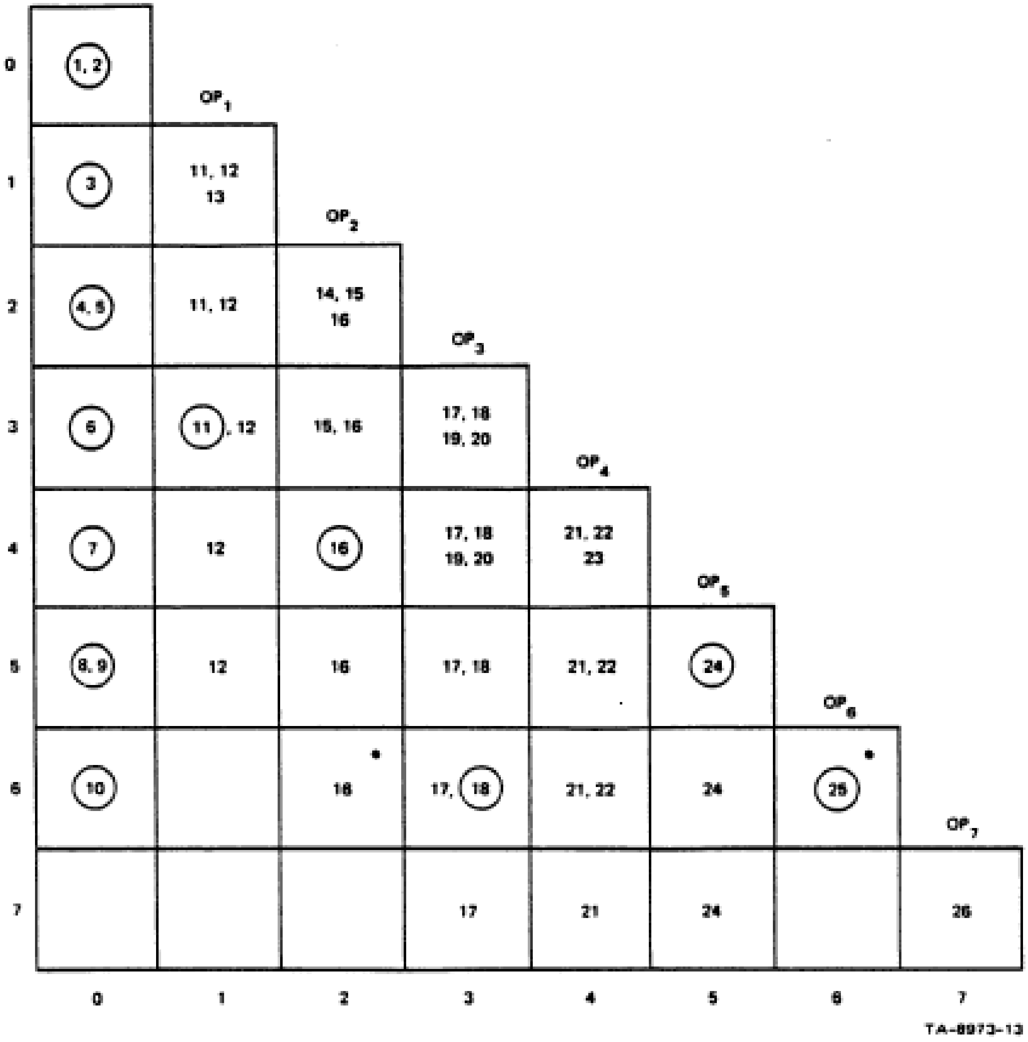
TA-8973-12

Marked Clauses

- Clauses are “marked” if needed to prove the precondition of the operator in that row.
- Unneeded clauses are deleted.



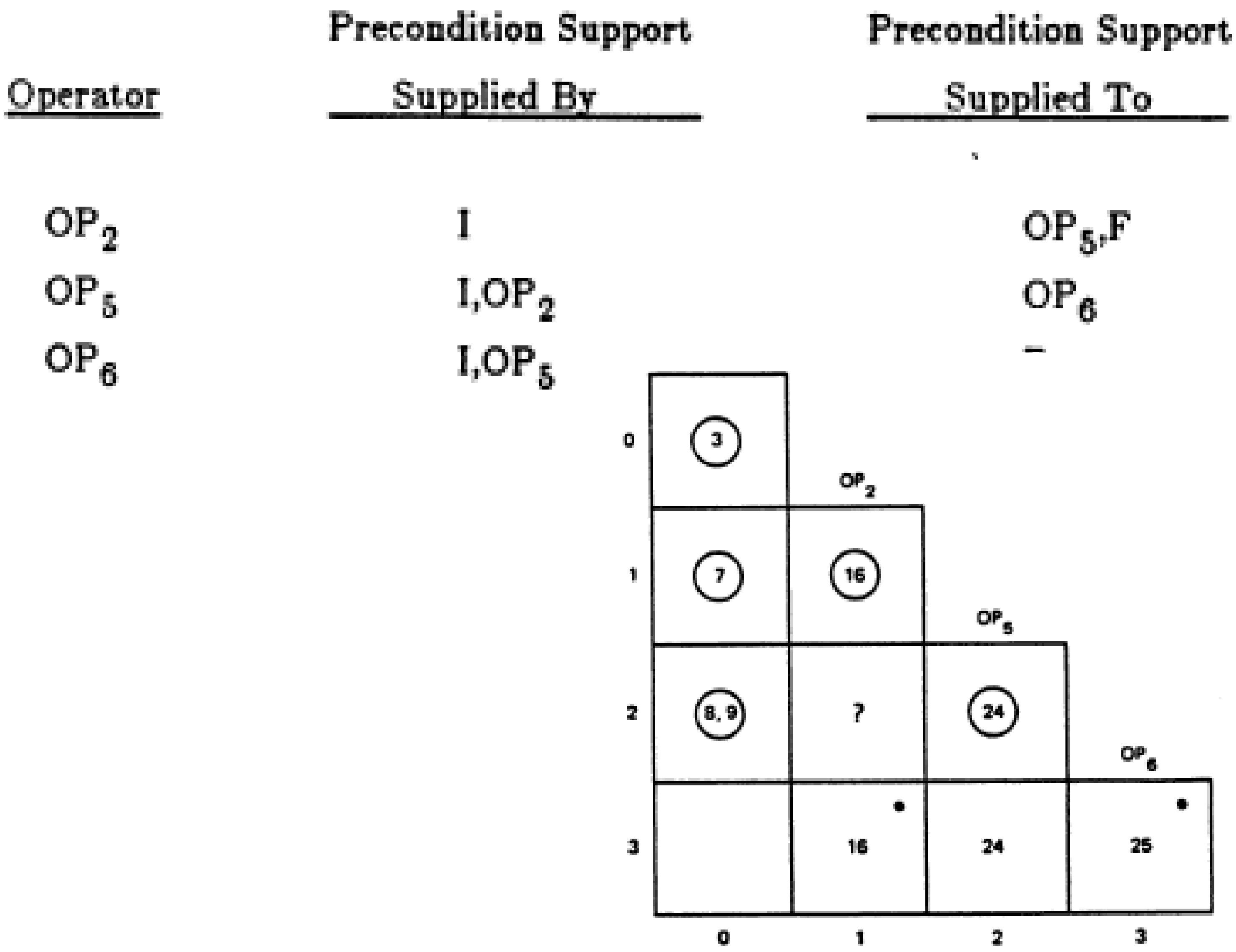
Operator	Precondition Support Supplied By	Precondition Support Supplied To
OP_1	I	OP_4
OP_2	I	OP_5
OP_3	I	OP_7, F
OP_4	I,OP_1	F
OP_5	I,OP_2	OP_6, F
OP_6	I,OP_5	OP_7
OP_7	I,OP_3,OP_6	F



	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

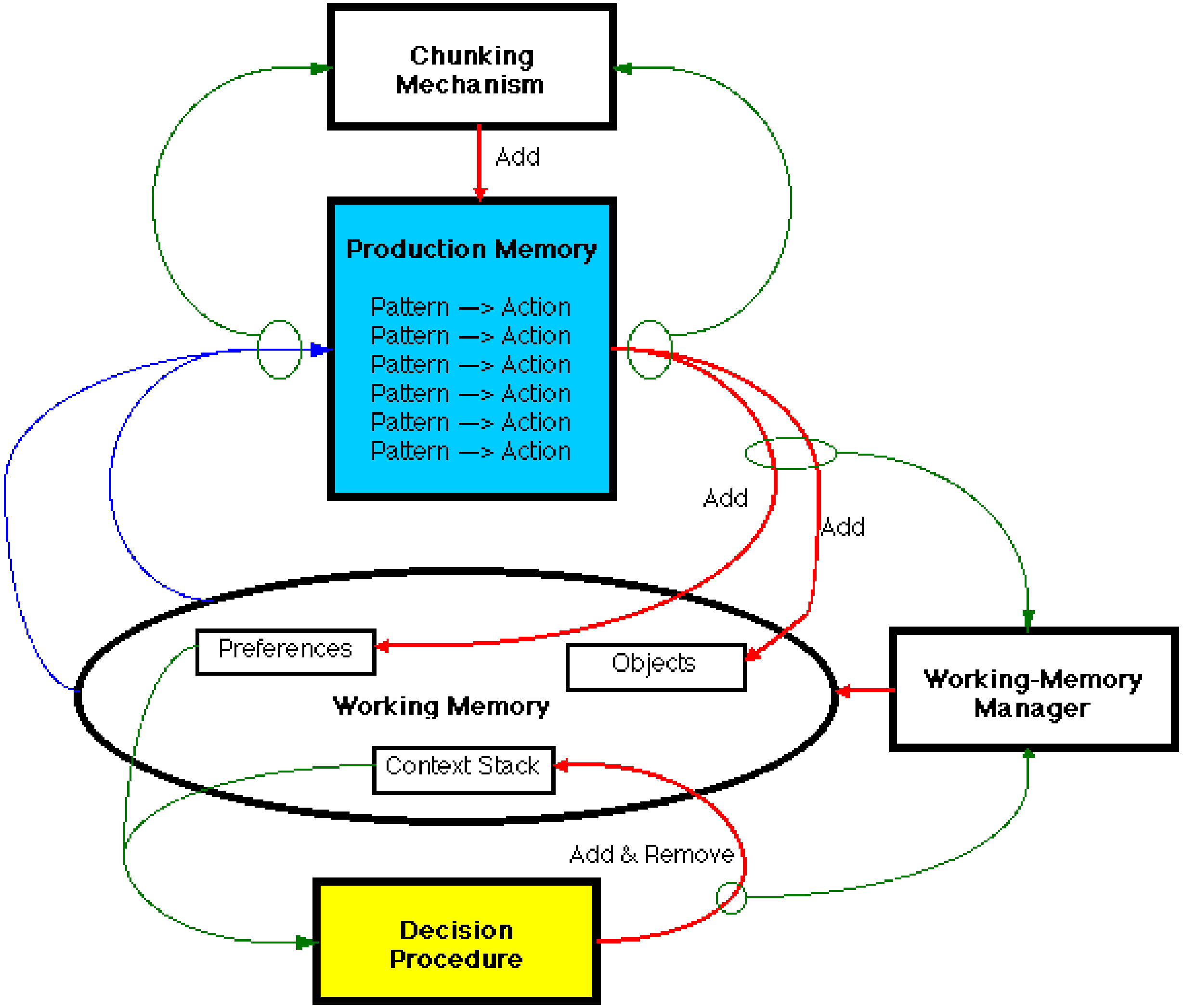
Resulting MACROP



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	–

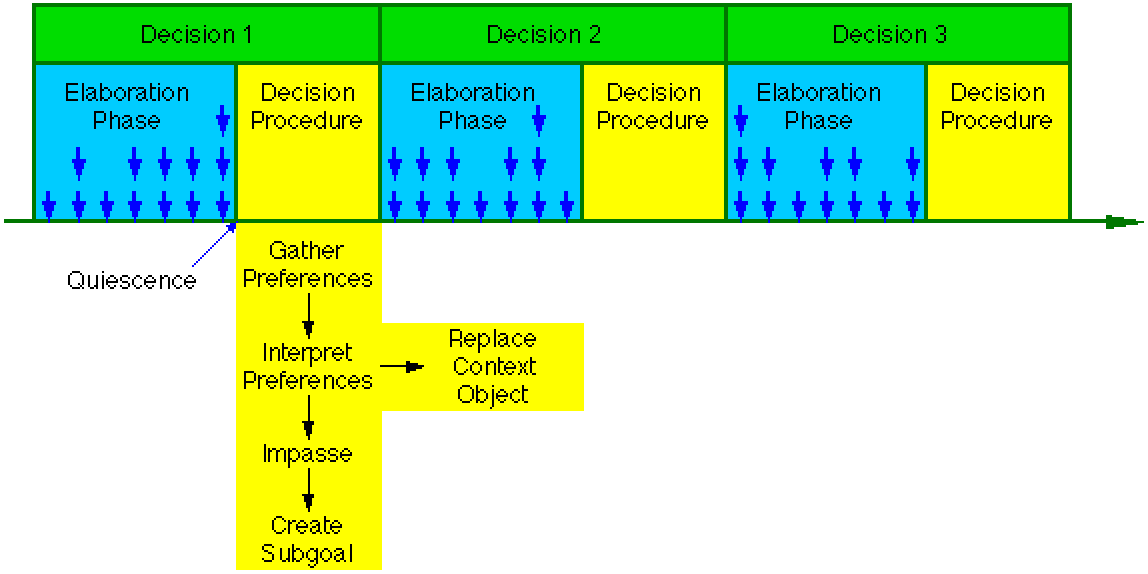
SOAR

- SOAR is a cognitive modeling architecture originally developed by Allen Newell and his students at CMU.



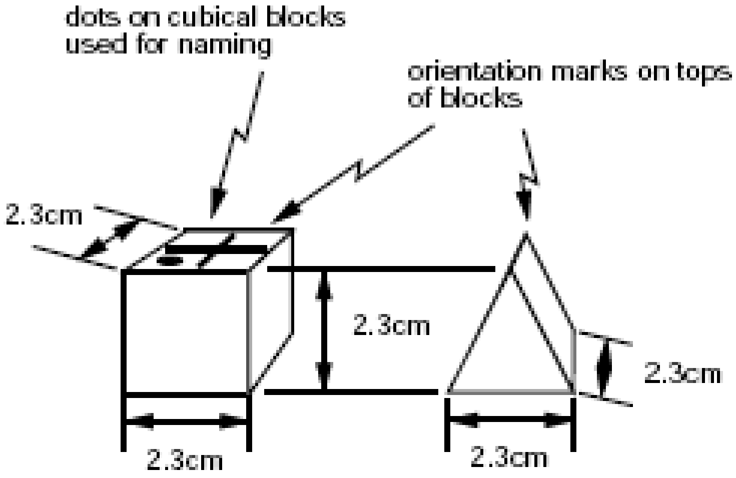
SOAR

- SOAR uses *production rules* to match items in working memory, update the memory, and initiate actions.
- An impasse (failure to match) leads to subgoal creation.
- Chunking is used to abstract and remember production sequences.



Robo-SOAR

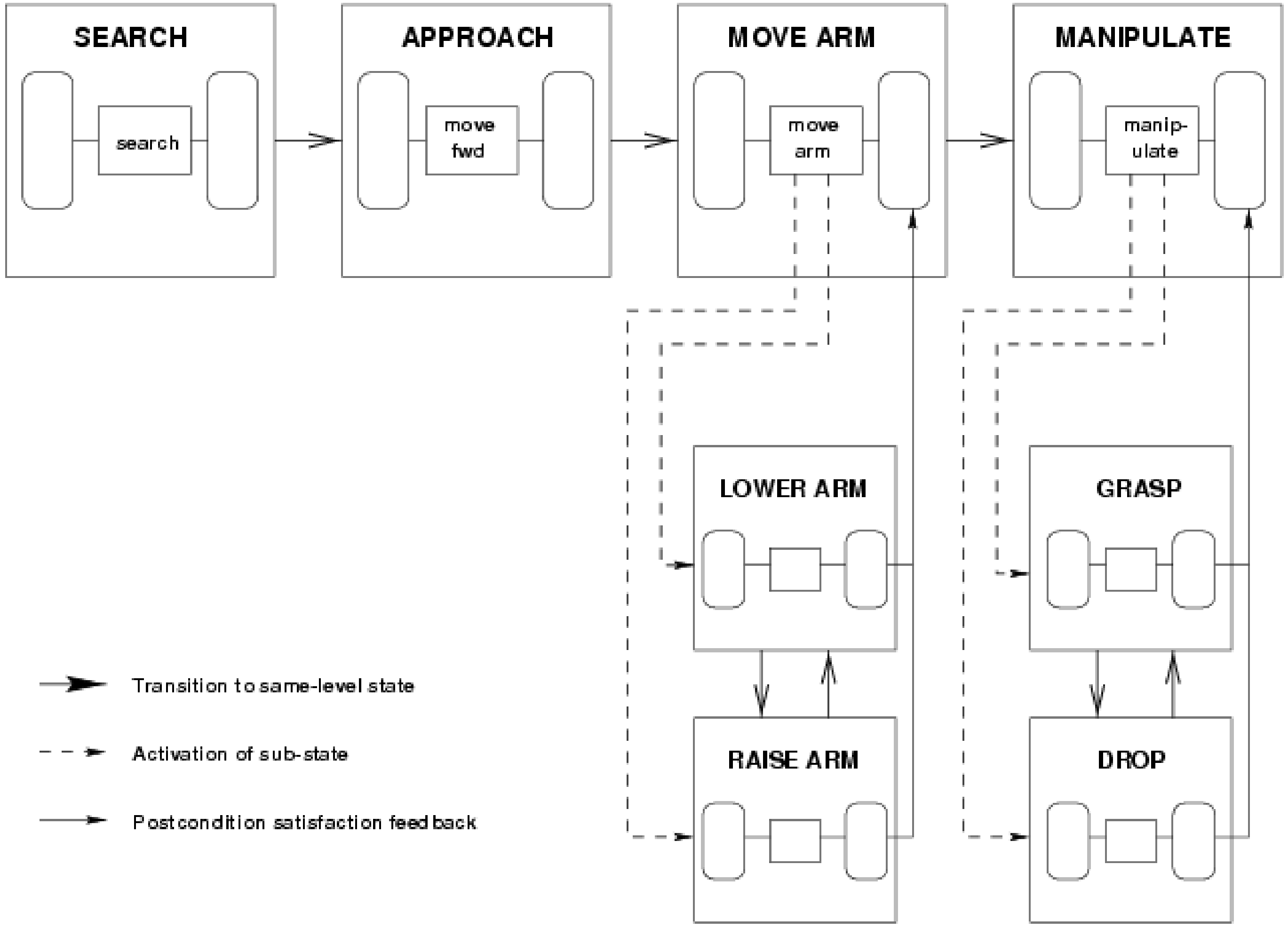
- Laird, Yager, and Hucka (1991)
- Extended SOAR to allow control of a Puma 560 arm
- Solved simple blocks world problems
- Two block types: cubes and pyramids
- Initial plan to pick up a pyramid fails when gripper not oriented correctly.



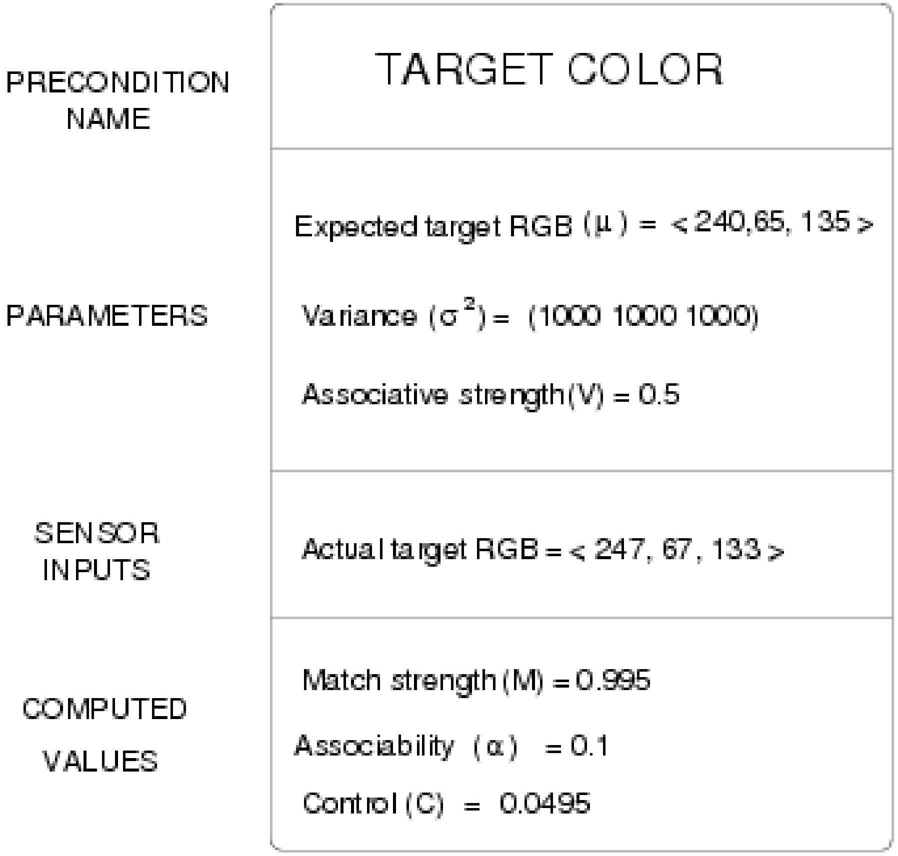
Skinnerbots (Touretzky & Saksida)

- Can we apply Skinnerian (operant) conditioning to robots?
  - Represent behaviors as schemas with modifiable preconditions
  - Use reward signal to train the robot
  - Try to infer preconditions from context + reward

Schema Representation



Example of a Precondition



PRECONDITION NAME	TARGET COLOR
PARAMETERS	Expected target RGB (μ) = < 240,65, 135 > Variance (σ^2) = (1000 1000 1000) Associative strength (V) = 0.5
SENSOR INPUTS	Actual target RGB = < 247, 67, 133 >
COMPUTED VALUES	Match strength (M) = 0.995 Associability (α) = 0.1 Control (C) = 0.0495

Machine Learning Algorithms Applied to the AIBO

- Temporal Difference (TD) learning for classical conditioning

- Two-armed bandit learning problem

Potential for Learning in Tekkotsu

- Currently there is map learning (MapBuilder)

- Cognitive robotics student project (2006): interface to ACT-R

- New ideas:
  - Provide a way to supply reward signals
  - Provide some persistence of memory across reboots via a facility for storing variables on the memory stick.