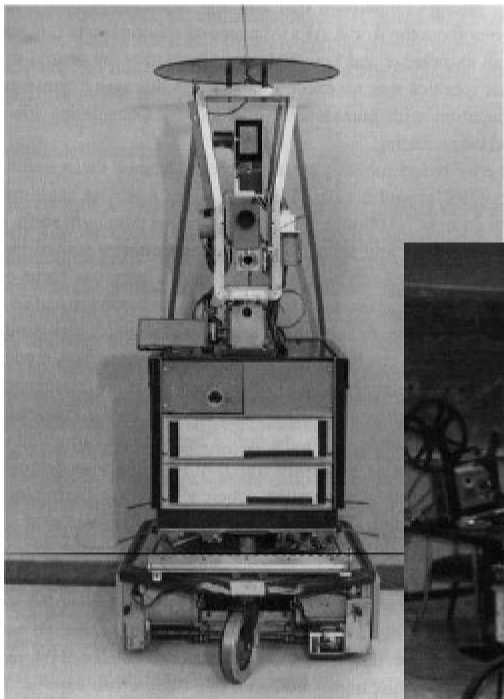
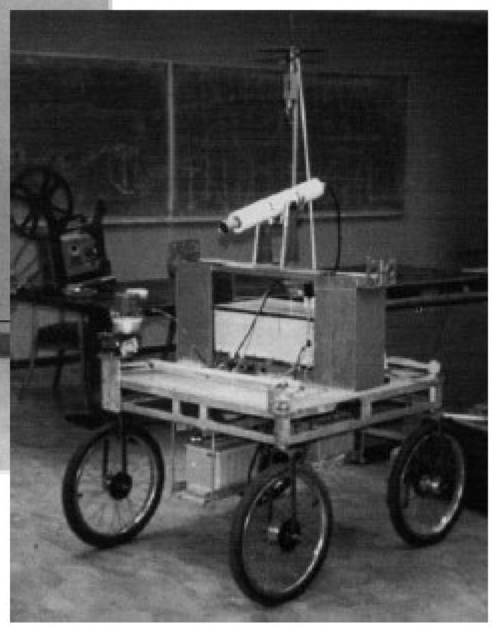
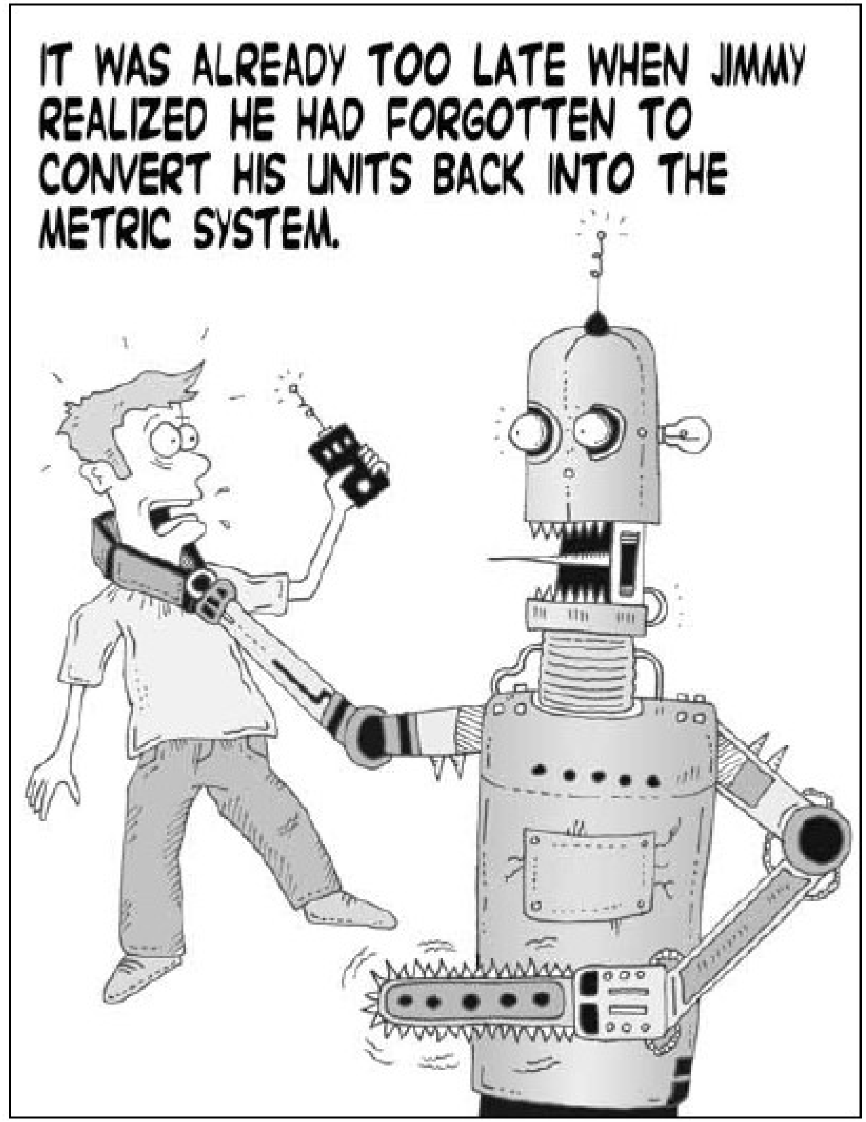
Hierarchical Paradigm



Shakey



Stanford Cart



Objectives

- To understand organizational differences between 3+ robot control paradigms
- To understand distinction between problem solver and planner
- To understand methods used by problem solvers
- To understand STRIPS approach to problem solving
- To understand example hierarchical architectures such as NHC and RCS

Robotic Paradigms

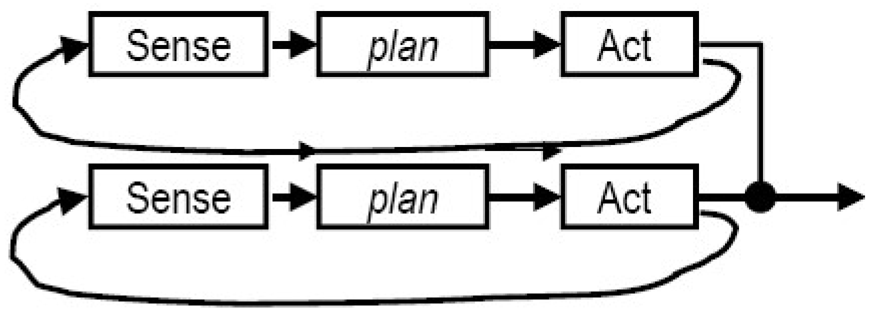
- A paradigm is a philosophy or set of assumptions and/or techniques which characterize an approach to a class of problems
- Applying the right paradigm makes problem solving easier
- There are generally three paradigms for organizing intelligence in robots: hierarchical, reactive, and hybrid deliberative/reactive

Three+ Primary Control Paradigms

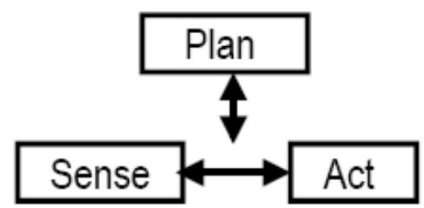
- Hierarchical

- Reactive

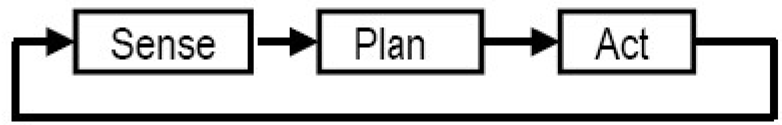
  – Behavior-based



- Hybrid deliberative/reactive



Hierarchical Paradigm

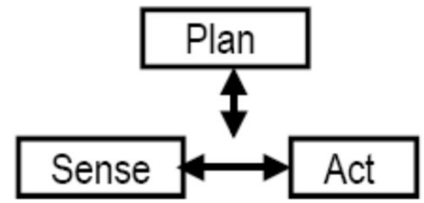


- 1967-1990
- Heavy on planning
- Senses the world, plans the next action, and acts
- Sensing data is gathered into one global world model
- World model is hard to maintain and brittle due to frame problem and closed world assumption

Reactive Paradigm

- 1988-1992
- Sensed information is directly coupled to an action, each is called a behavior
- Fast execution
- Motivated by:
  – Biology and cognitive psychology that examine living examples of intelligence
  – Low cost and increasing computing power on robot
- Multiple instances of SENSE-ACT couplings and combined results
- Throwing away planning is too extreme
- Behavior-based approach provides some levels of planning

Hybrid Paradigm



- 1990 - present
- Reactive paradigm serves as basis
- Robot first plans how to decompose a task into subtasks and then what behaviors to use for each subtask
- Then behaviors execute as per reactive paradigm
- Sensing is also hybrid, which connects to behavior directly and to the planner for constructing a task-oriented global world model

PLANNING
IN HIERARCHICAL PARADIGM

Planning vs. Problem Solving

- Planning agent is very similar to problem solving agent
  - Constructs plans to achieve goals, then executes them

- Planning agent is different from problem solving agent in:
  - Representation of goals, states, actions
  - Use of explicit, logical representations
  - Way it searches for solutions

Problem Solver Characteristics

- Search-based problem solver:
  - Representation of actions: programs that generate successor state descriptions
  - Representation of states: complete state description; a data structure holding permutations of all possible states
  - Representation of goals: goal test and heuristic function to decide desirability
  - Representation of plans: solution is a sequence of actions; considers only unbroken sequences of actions

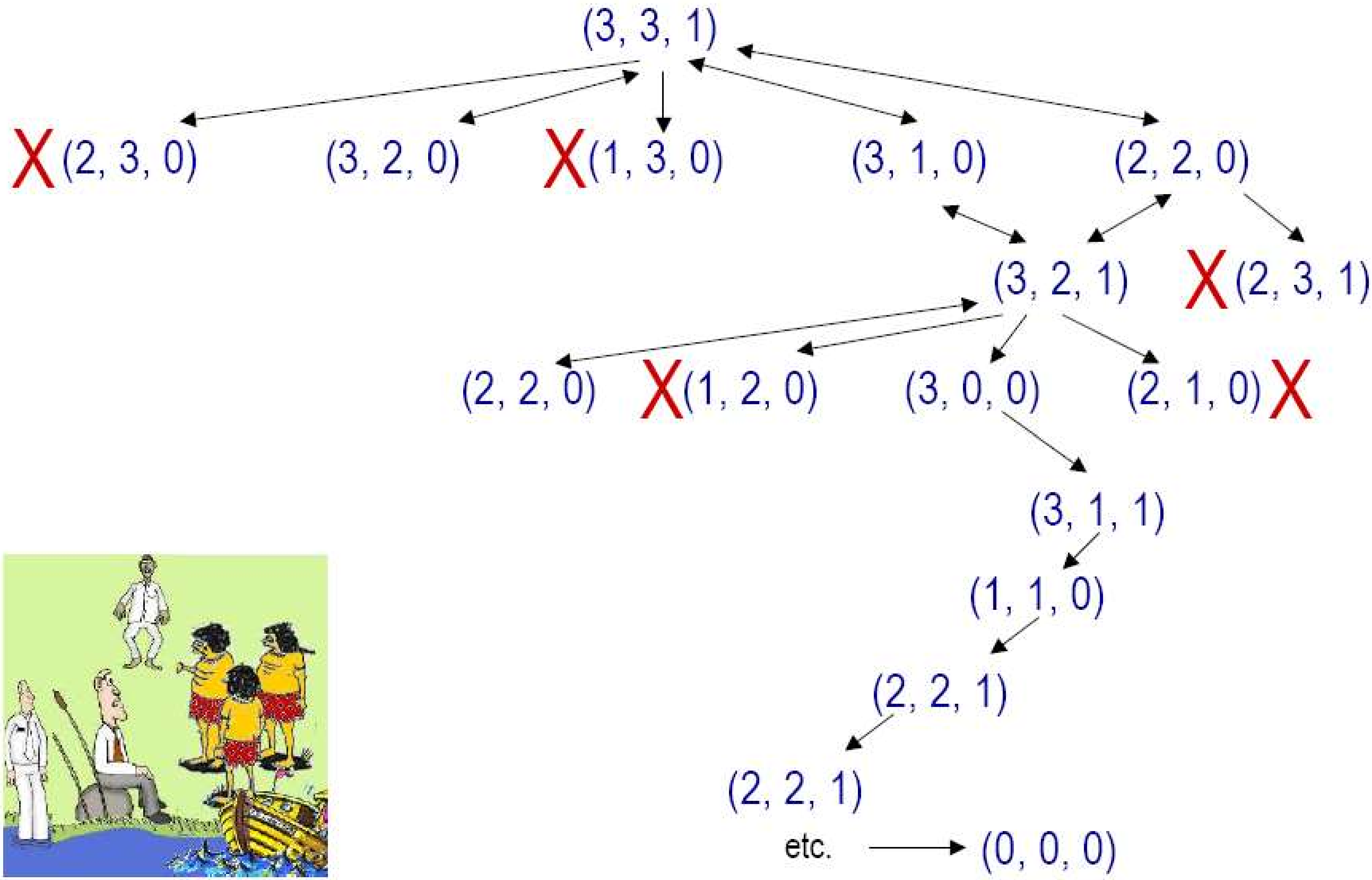
Famous Problem Solver Task: “Missionaries and Cannibals”

- “Missionaries and cannibals” problem:
  - Famous in AI
  - http://www.learn4good.com/games/puzzle/boat.htm

Formalizing Missionaries and Cannibals Problem

- Define states, actions (operators), goal test, path cost
  - States: <m, c, b> representing the # of missionaries and the # of cannibals, and the position of the boat
  - Initial state:
    - <3, 3, 1>
  - Actions:
    - take 1 missionary, 1 cannibal, 2 missionaries, 2 cannibals, or 1 missionary and 1 cannibal across the river
  - Goal test:
    - <0, 0, 0>
  - Path cost:
    - number of crossing

Solving Missionaries and Cannibals Problem

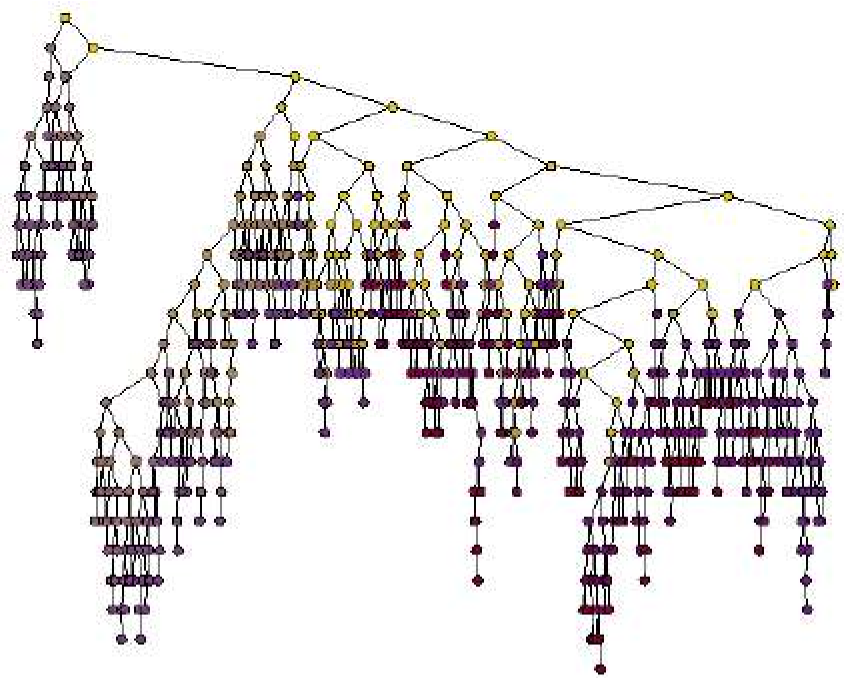


Search Strategies for Problem Solvers

- Key Criteria:
  - Completeness: guaranteed to find a solution when one exists
  - Time complexity: how long does it take to find solution?
  - Space complexity: how much memory is needed to find solution?
  - Optimality: does strategy find the highest-quality solution?

Various Search Strategies for Problem Solver

- Uninformed search (no info. about path cost from current state to goal):
  - Breadth-first
  - Uniform cost
  - Depth-first
  - Depth-limited
  - Iterative deepening
  - Bidirectional
  - Etc.

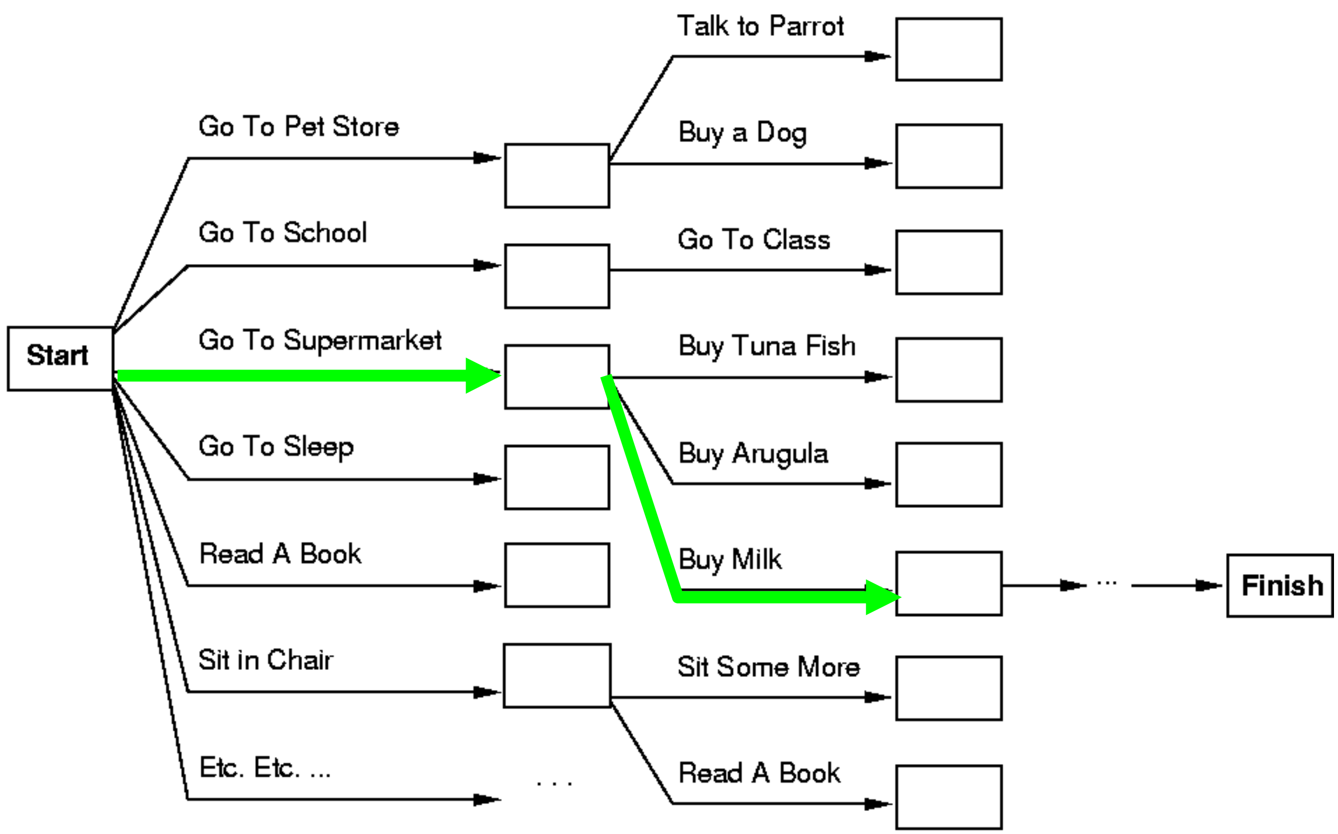


(a messy search tree)

- Informed search
  - Greedy
  - A*
  - Hill-climbing/gradient descent
  - Simulated annealing
  - Etc.

Search vs. Planning

- Consider the task: get milk, bananas, and a cordless drill
  - Standard search algorithms seem to fail miserably
  - Why? Huge branching factor & heuristics



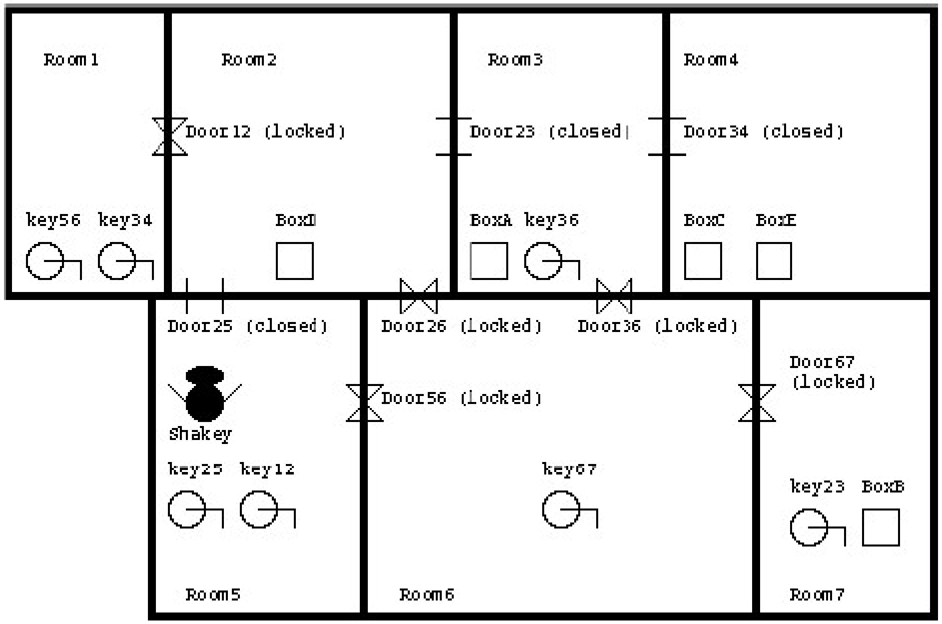
Search vs. Planning

- Planning systems do the following:
  - divide-and-conquer
  - relax requirement for sequential construction of solutions

	Search	**Planning**
States	data structures	logical sentences
Actions	code	preconditions/outcomes
Goal	code	logical sentences
Plan	sequence from s_0	constraints on actions

Planning-Based Approach to Robot Control

- Job of planner: generate a goal to achieve, and then construct a plan to achieve it from the current state
- Must define representations of:
  - Actions: generate successor state descriptions by defining preconditions and effects
  - States: data structure describing current situation
  - Goals: what is to be achieved
  - Plans: solution is a sequence of actions



First Order Predicate Calculus

- First order predicate calculus: formal language useful for making inferences and deductions

- Elementary components:
  - Predicate symbols (e.g., WRITES(), OWNS(), LIVES())
  - Variable symbols (e.g., *x*, *y*)
  - Function symbols (e.g., father(x))
  - Constant symbols (e.g., HOUSE-1)
  - Connectives: and, or, negation, implies
  - Quantification:
    - Universal
    - Existential

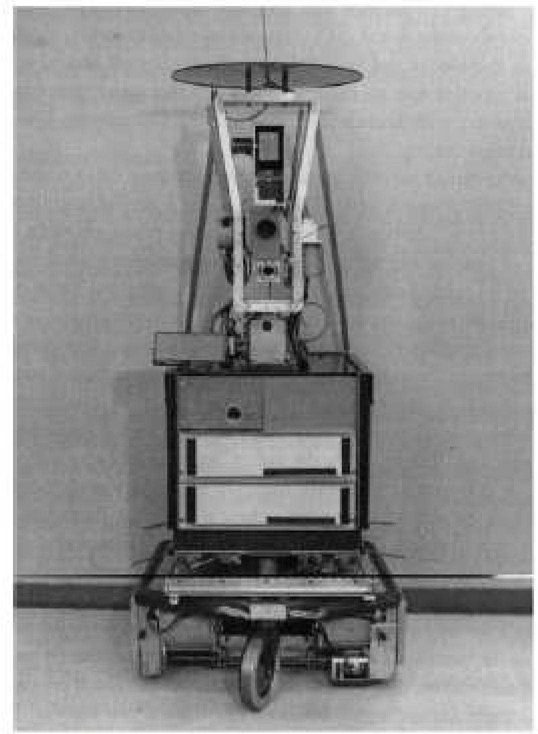
STRIPS-Based Approach to Robot Control

- Use first-order logic and theorem proving to plan strategies from start to goal

- STRIPS language:
  - "Classical" approach that most planners use
  - Lends itself to efficient planning algorithms

- Cost: $100, 000

- See a video of Shakey here: https://www.youtube.com/watch?v=GmU7SimFkpU&t=46s



Shakey (SRI), 1960's

STRIPS

- STRIPS (STanford Research Institute Problem Solver)
  - a restrictive way to express states, actions and goals, but leads to more efficiency

- States: conjunctions of ground, function-free, and positive literals
  - E.g. At(Home) ^ Have(Banana)

- Goals: conjunctions of literals, may contain variables (existential), hence goal may represent more than one state
  - E.g. At(Home) ^ ¬ Have(Bananas)
  - E.g. At(x) ^ Sells(x, Bananas)

- Actions: preconditions that must hold before execution and the effects after execution

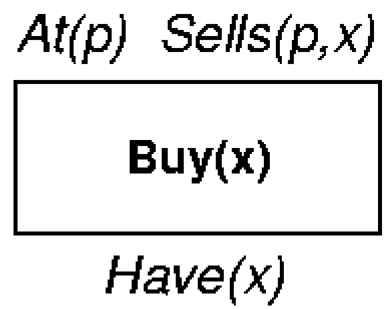
STRIPS Action Schema

- An action schema includes:
  - Action name & parameter list (variables): name for what an agent does
  - Precondition: a conjunction of function-free positive literals. Any variables in it must also appear in parameter list
  - Effect: a conjunction of function-free literals (positive or negative)
    - ADD List
    - DELETE List
- Example:

  Action: Buy (*x*)

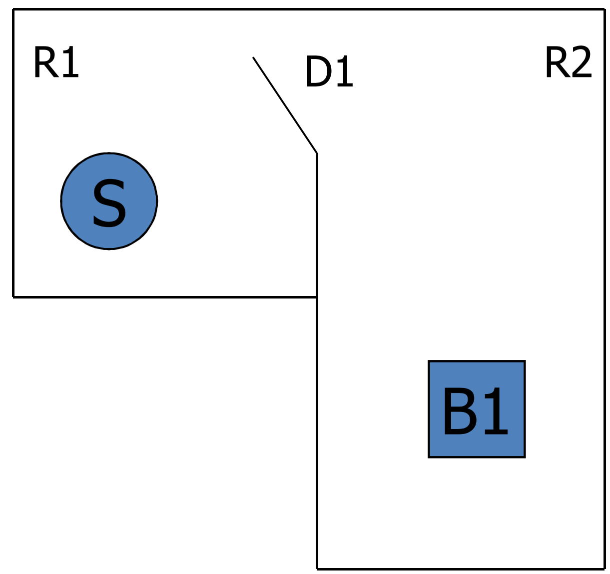
  Precondition: At (p), Sells (p, *x*)

  Effect: Have(*x*)



An Example of STRIPS World Model

- A world model is a set of facts
- The robot's knowledge can be represented by the following predicates:
  - INROOM(x, r), where x is a movable object, r is a room
  - NEXTTO(x, t)
  - STATUS(d, s), where d is a door, s means OPEN or CLOSED
  - CONNECTS(d, rx, ry)
- The world model in this figure can be represented with the above predicates, with the initial state and goal state



The Actions in STRIPS World

- Types of actions Shakey can make
  - Move from place to place:
    - GOTODOOR(S, dx):
      - PRECOND: INROOM(S, rk) ^ CONNECT(dx, rk, rm)
      - Effect:
        - » add-list: NEXTTO(S, dx)
        - » delete-list: null
    - GOTHRUDOOR(S, dx):
      - PRECOND: CONNECT(dx, rk, rm) ^ NEXTTO(S, dx) ^ STATUS(dx, OPEN) ^ INROOM(S, rk)
      - Effect:
        - » add-list: INROOM(S, rm)
        - » delete-list: INROOM(S, rk)

How to Develop a Plan in STRIPS?

- Use “means-ends analysis”
  - If robot is not at goal, then measure the distance (or difference) between robot’s current state and its goal state
  - Find an action that the robot can take that will reduce this distance (difference), and select it if open conditions are satisfied
  - Make the first false precondition of this action the new “subgoal”, remembering old goal by pushing on stack
  - Recursively reduce difference by repeating above
  - When all open conditions for an action are satisfied, add the action to the plan stack and update world model
  - When done, pop the actions off the stack to create the plan from start to goal

Go Back to the Previous Example

For details, please read the textbook 2.2.2.

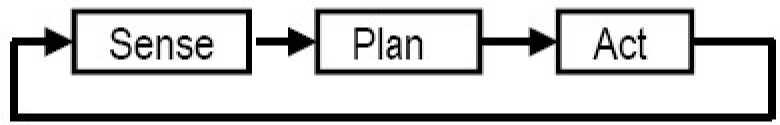
Summary of Planning Problem

- Planning agents use look-ahead to find actions to contribute to goal achievement
- Planning agents differ from problem solvers in their use of more flexible representation of states, actions, goals, and plans
- The STRIPS language describes actions in terms of preconditions and effects

Hierarchical Robotic Control: Representative Architectures

Recall: Hierarchical Paradigm

- Hierarchical paradigm



- The hierarchical paradigm is sequential and orderly
  - First the robot senses the world and constructs a global world model
  - Then “eyes” closed, the robot plans all the directives needed to reach the goal
  - Finally, the robot acts to carry out the first directive
  - After the robot has carried out the whole sequence, it repeats the cycle again

World Model

- All sensor observations are fused into one global data structure – the world model
  - a priori representation of the environment the robot is operating in (e.g., map)
  - sensing information (e.g., "I am at a doorway")
  - any additional cognitive knowledge
- Recall our STRIPS planner
  - it works recursively by backward chaining
  - it generates a list of operators (actions) to apply
  - it plans first and then executes

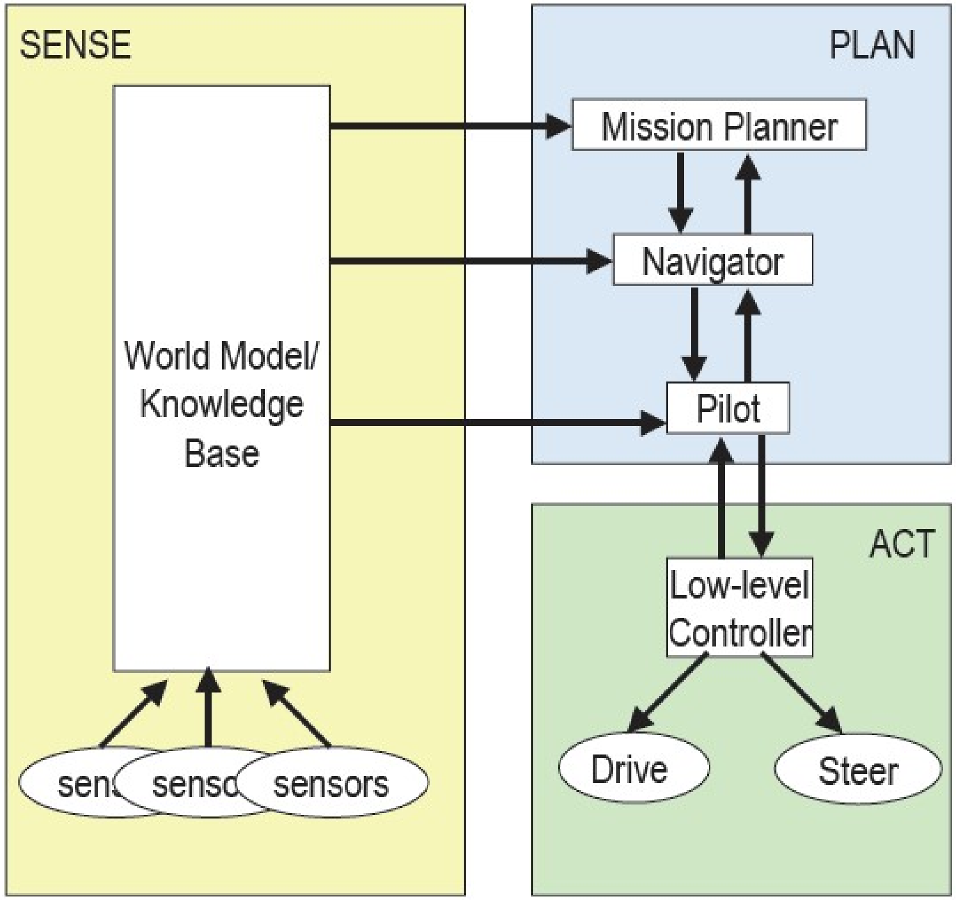
Architectures

- "Architecture": a method of implementing a control paradigm, of embodying certain principles in a concrete way
- Evaluation of existing architecture is based on the following criteria:
  - Support for modularity
    - does it show good software engineering principles?
  - Niche targetability
    - how well does it work for intended applications?
  - Ease of portability to other domains
    - how well would it work for other applications or other robots?
  - Robustness
    - is the system vulnerable?

Case Studies

- Two best-known hierarchical architectures:
  - Nested Hierarchical Controller (NHC)
    - developed by Meystel
  - NIST Realtime Control System (RCS)
    - adapted to a teleoperation version called NASREM
    - developed by Albus

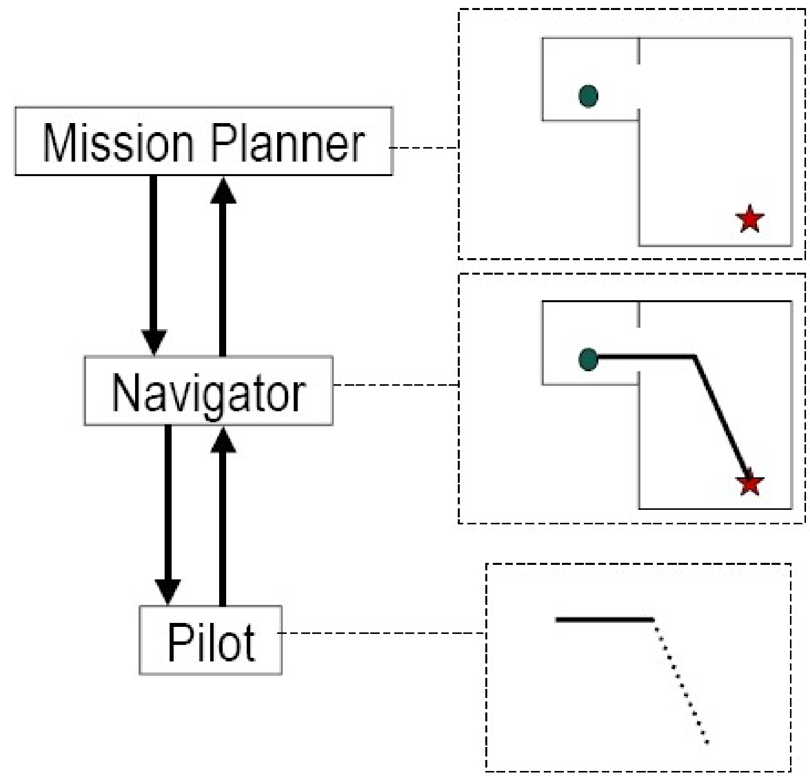
Nested Hierarchical Controller



Major contribution of NHC: decomposition of planning into three subsystems

Planning is Hierarchical

- Uses map to locate self and goal
- Generates path from current position to goal
- Generates actions robot must execute to follow path segment



Details of NHC

- Mission planner:
  - either receives a mission from a human or generates a mission for itself
  - responsible for translating the mission into terms that other functions can understand (e.g., access a map of the building and locates where the robot is and where the goal is)

- Navigator:
  - takes information from mission planner (e.g., current position, goal position) and generates a path in the form of a set of waypoints, or straight lines for the robot to follow

- Pilot:
  - takes the first straight line or path segment and determines what actions the robot has to do to follow the path segment (e.g., speed, turning rate)

Details of NHC, Continue

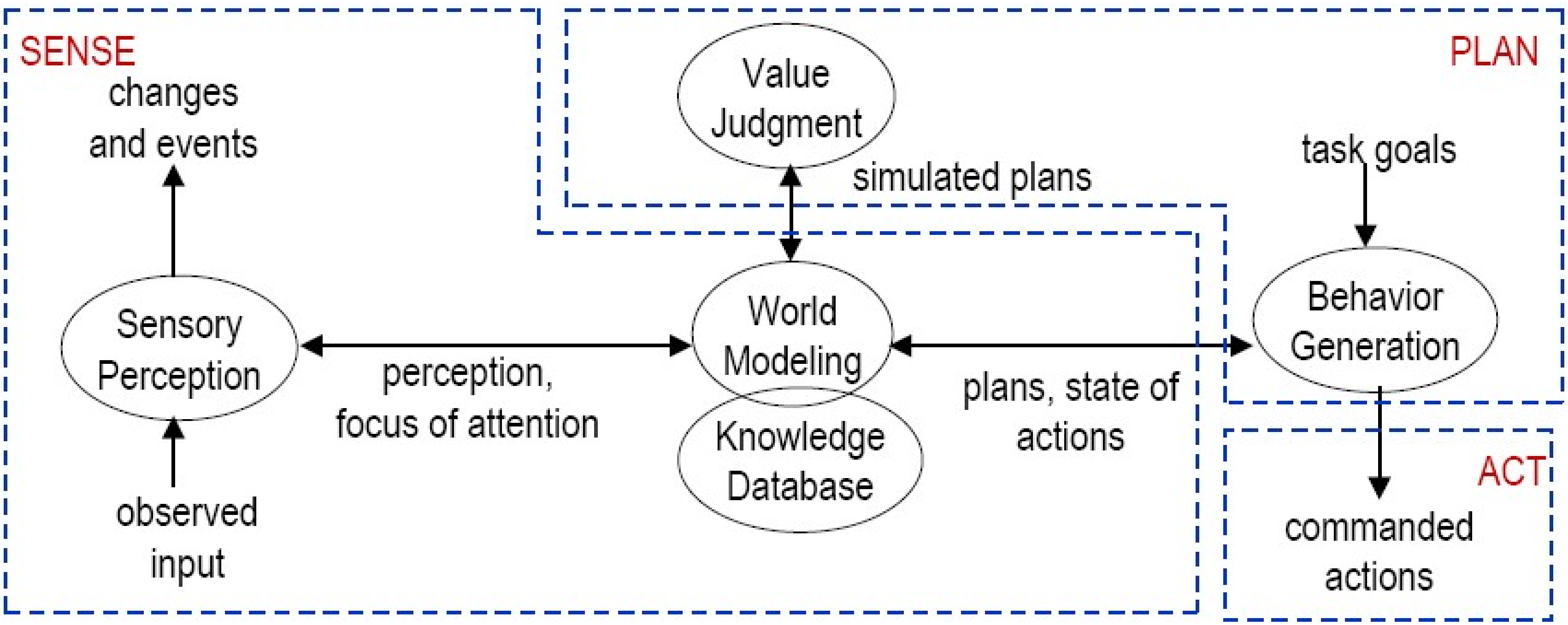
- After the robot has executed the commands from Pilot, it senses again and updates the world model
  - the entire planning cycle does not need to be repeated
  - it checks the changes, and repeats the corresponding part
    - e.g., if the robot has reached its waypoint, the Pilot informs the Navigator; if the waypoint isn't the goal, then pass the next waypoint for the robot to reach; if the waypoint is the goal, then the Navigator informs the Mission Planner; the Mission Planner may issue a new goal, etc.
    - e.g., if the robot has encountered an obstacle to its path, the Pilot passes control back to the Navigator; the Navigator computes a new path, and gives it to the Pilot to carry out

Advantage/Disadvantage of NHC

- Advantage:
  - Interleaves planning and action (unlike STRIPS)
    - Plan is changed if world is different from expected
  - Note that the decomposition is inherently hierarchical in intelligence and scope
    - Mission planner is smarter than Navigator, who is smarter than the Pilot
- Disadvantage:
  - Planning decomposition is only appropriate for navigation tasks

NIST RCS & NASREM

- NIST RCS (National Institute of Standards and Technology Real-time Control System): developed to serve as standard for manufactures' development of more intelligent robots
- Similar in design to NHC
  - Primary difference: sensory perception includes preprocessing (feature extraction, or focus of attention)



Details of RCS

- Value judgment plans and the simulates the plans to ensure they will work
- The planner gives the plan to Behavior Generation module, which converts the plans into actions

Multiple Layers of NIST RCS & NASREM

- Each layer has:
  - Sensory processing
  - World modeling
  - Task decomposition
  - Value judgment

- All layers joined by global memory through which representational knowledge is shared

- Perception is not tied directly to action
- Perception is still integrated into a global world model

NHC and RCS/NASREM: Well-Suited for Semi-Autonomous Control

- Human operator could:
  - Provide world model
  - Decide mission
  - Decompose mission into plan
  - Decompose plan into actions

- Lower-level controller (robot) could:
  - Execute actions

Evaluations of Hierarchical Architectures

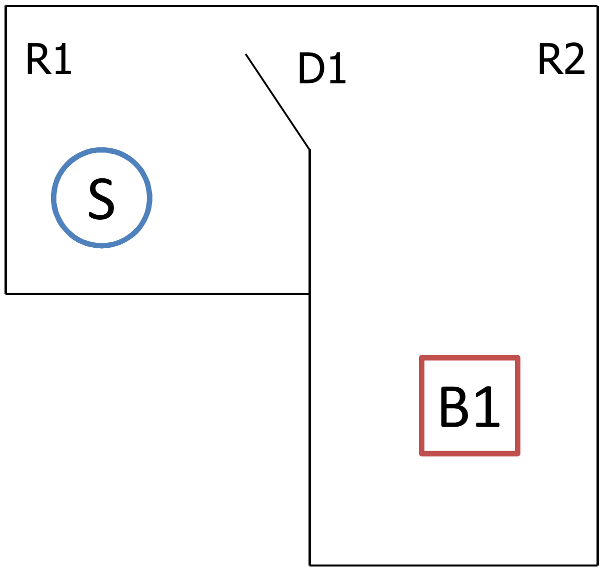
- Robots (other than the Walter's tortoise and Braitenberg's vehicles) built before 1985 typically used hierarchical style

- Primary advantage:
  - provides an ordering of the relationship between sensing, planning, and acting

- Primary disadvantage:
  - planning: every update cycle, robot had to update global world model, then plan, which introduces a bottleneck
  - sensing and acting are disconnected
  - appropriate hierarchical decomposition is application-dependent
  - uncertainty not well handled

Exercise: STRIPS Example

- Predicates:
  - INROOM(x, r), where x is a movable object, r is a room
  - NEXTTO(x, t), where x is a movable object, t is a door or movable object
  - STATUS(d, s), where d is a door, s means OPEN or CLOSED
  - CONNECTS(d, rx, ry)



- Difference table

Operator	Preconditions	add-list	delete-list
OP1: GOTODOOR(S, dx)	INROOM(S, rk) CONNECT(dx, rk, rm)	NEXTTO(S, dx)	
OP2: GOTHRUDOOR(S, dx)	CONNECT(dx, rk, rm) NEXTTO(S, dx) STATUS(dx, OPEN) INROOM(S, rk)	INROOM(S, rm)	INROOM(S, rk)

Exercise

- If I want the robot to move from the position that’s next to door to box B1 in the same room
  - What operator/predicates do we need to add?
  - Construct a plan
  - Show the changes in the world model after each operator is applied

Partial Answer

GotoBox(S, bx)

Preconditions:

INROOM(S, rk), INROOM(bx, rk),

NEXTTO(S, dx)

Add-list:

NEXTTO(S, bx)

Delete-list:

NEXTTO(S, dx)