Hierarchical Paradigm and STRIPS

September 3, 2002

Class Meeting 4
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 NHC and RCS
Recall: Three+ Primary Control Paradigms
(compare to Figure I.3 in Murphy, pg. 7)

1. Hierarchical:
   Sense → Plan → Act

2. Reactive:
   Sense → Act

2+. Behavior-Based:
   Sense → plan → Act
   Sense → plan → Act

3. Hybrid deliberative/reactive:
   Plan
   Sense → Act
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 enables more sensible deliberations of potential solutions
  – 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 descriptions; typically, data structure holding permutations of all possible states
  
  – Representation of goals: goal test and heuristic function to decide desirability
  
  – Representation of plans: solution is sequence of actions; considers only unbroken sequences of actions
Famous Problem Solver Task: “Missionaries and Cannibals”

• “Missionaries and cannibals” problem:
  – Famous in AI

  – Was subject of first paper (Amarel, 1968) that approached problem formulation from analytical viewpoint

  – Problem statement:
    • 3 missionaries and 3 cannibals on one side of river, with boat that can hold 1-2 people
    • Find: way to get everyone to other side of river, without ever leaving group of missionaries in one place outnumbered by cannibals in that place
Formalizing Missionaries and Cannibals Problem

• Ignore all irrelevant parts of the problem (e.g., weather conditions, crocodiles, etc.)
• Define states, operators, goal test, path cost:
  – States: ordered sequence of 3 numbers:
    • (# missionaries, # cannibals, #boats on initial riverbank)
    • E.g.: Start state = (3, 3, 1)
  – Operators:
    • Take 1 missionary, 1 cannibal, 2 missionaries, 2 cannibals, or one of each across in boat.
    • Take care to avoid illegal states
  – Goal test:
    • We’ve reached state (0, 0, 0)
  – Path cost:
    • # of crossings
Solving Missionaries and Cannibals Problem

(3, 3, 1)

X (2, 3, 0) (3, 2, 0) X (1, 3, 0) (3, 1, 0) (2, 2, 0)

(3, 2, 1) X (2, 3, 1)

(2, 2, 0) X (1, 2, 0) (3, 0, 0) (2, 1, 0) X

(3, 1, 1)

(1, 1, 0)

(2, 2, 1)

(2, 2, 1) etc. (0, 0, 0)
Search Strategies for Problem Solvers

• Key criteria:

– Completeness: is strategy 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 highest-quality solution?
Alternative Search Strategies for Problem Solvers

• Variety of search approaches:
  – Uninformed search (no information about path cost from current state to goal):
    • Breadth-first
    • Uniform cost search
    • Depth-first
    • Depth-limited search
    • Iterative deepening
    • Bidirectional
    • Etc.
  – Informed search
    • Greedy
    • A*
    • Hill-climbing/gradient descent
    • Simulated annealing
    • Etc.
Important Points Regarding Problem Solvers

• Search-based problem solver:

  – Representation of states:  *complete* state descriptions; typically, data structure holding permutations of all possible states

  – Representation of goals: goal test and heuristic function to decide desirability; cannot “look inside” to select actions that might be useful for achieving goal

  – Representation of plans: solution is sequence of actions; considers only unbroken sequences of actions from start to goal
From Problem Solvers to Planners

• Key ideas:
  – “Open up” representation of states, goals, and actions
    • Use descriptions in a formal language – usually first-order logic
    • States/goals represented by sets of sentences
    • Actions represented by logical descriptions of preconditions and effects
    • Enables planner to make direct connections between states and actions
  
  – Planner can add actions to plan whenever needed, rather than in strictly incremental fashion
    • No necessary connection between order of planning and order of execution
    • Note that planner states are equivalent to entire classes of problem-solver states

  – Most parts of the world are independent of most other parts of the world
    • Conjunctions can be separated and handled independently
    • Divide-and-conquer algorithms: efficient because it is easier to solve several small subproblems rather than one big problem
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:**
  - Representation of actions: programs that generate successor state descriptions, defining preconditions and effects
  - Representation of states: data structure describing current situation
  - Representation of goals: what is to be achieved
  - Representation of plans: solution is a sequence of actions

*First-order logic and theorem proving enable planning of strategies from start state to goal*
Recall: First Order Predicate Calculus in AI

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

• Elementary components:
  – Predicate symbols (e.g., WRITES(), LIVES(), OWNS(), MARRIED())
  – Variable symbols (e.g., x, y)
  – Function symbols (e.g., father(x) returns the father of x)
  – Constant symbols (e.g., HOUSE-1, NERO, GEORGE-BUSH)
  – Connectives
    • and, or, negation, implies \( \land, \lor, \neg, \Rightarrow \)
  – Quantification
    • Universal \( \forall x \)
    • Existential \( \exists x \)

• NOTE: First order means quantification over predicates or functions not allowed
Situation Calculus

- **Situation calculus:**
  - Refers to a particular way of describing change in first-order logic
  - Conceives of world as consisting of a sequence of *situations*, each of which is a snapshot of the state of the world

- All relations/properties of world that can change over time:
  - Specify extra *situation argument* to predicate
  - Instead of: $At(\text{agent}, \text{location})$
  - Indicate: $At(\text{agent}, \text{location}, S_i)$, where $S_i$ is a particular situation, or point in time

- Represent how world changes from one situation to next:
  - $\text{Result(Forward, } S_0) = S_1$
  - $\text{Result(Turn(Right), } S_1) = S_2$
  - $\text{Results(Forward, } S_2) = S_3$
Example of World Represented by Situation Calculus

- Result(Forward, S_0) = S_1
- Result(Turn(Right), S_1) = S_2
- Result(Forward, S_2) = S_3
STRIPS-Based Approach to Robot Control

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

• STRIPS language:
  – “Classical” approach that most planners use
  – Lends itself to efficient planning algorithms
  – Retains expressiveness of situation calculus

Shakey (SRI), 1960’s
• **States:** Conjunctions of function-free ground literals (i.e., predicates applied to constant symbols, possibly negated)
  – Example: \( \text{At(Home)} \quad \text{Have(Milk)} \quad \text{Have(Bananas)} \)

  – Common assumption: If state description does not mention a given positive literal, then the literal is assumed to be false

• **Goals:** Conjunctions of literals, which can include variables
  – Example: \( \text{At(x)} \quad \text{Sells(x,Milk)} \)

  – Assumption: variables are existentially quantified
STRIPS Representation -- Actions

• STRIPS operators consist of:
  – Action description: name for what an agent does
  – Precondition: conjunction of positive literals (atoms) saying what must be true before operator can be applied
  – Effect: conjunction of literals (positive or negative) that describes how the situation changes when operator is applied
    • Organize effect as:
      – ADD LIST
      – DELETE LIST
Recall: Simple Example of STRIPS Blocks-World Problem

• Goal State: ON(A,B)
• Start state: ON(A, Table); ON(B, Table); EMPTYTOP(A); EMPTYTOP(B)
• Operator:
  – MOVE(x,y):
    • Preconditions: ON(x,Table); EMPTYTOP(y)
    • Add-List: ON(x,y)
    • Delete-List: EMPTYTOP(y); ON(x,Table)
Shakey’s STRIPS World

• Types of actions Shakey can make (at least in simulation):
  – Move from place to place:
    Go(y):
    PRECOND: \text{At}(\text{Shakey},x) \
             \text{In}(x,r) \land \text{In}(y,r) \land \text{On}(\text{Shakey},\text{Floor})
    EFFECT: \text{At}(y)
  – Push movable objects:
    Push(b, x, y):
    PRECOND: \text{Pushable}(b) \land \text{At}(b,x) \land \text{At}(\text{Shakey},x) \land \text{In}(x,r) \land \text{In}(y,r) \land \text{On}(\text{Shakey},\text{Floor})
    EFFECT: \text{At}(b,y)
Shakey’s STRIPS World (con’t.)

- Types of actions Shakey can make (at least in simulation):
  - Climb onto rigid objects:
    Climb(b):
    PRECOND: Climbable(b)
    At(Shakey,x) ∧ At(b,x)
    On(Shakey,Floor)
    EFFECT: On(Shakey,b)
  - Climb down from rigid objects:
    (etc.)
  - Turn light switches on and off:
    (etc.)
STRIPS Representation -- Plans

• A Plan is a data structure consisting of:

  – A set of plan steps.
    • Each step is one of operators for the problem.
  – A set of step ordering constraints. \( S_i \prec S_j \)
    • Ordering constraints specify that an action has to occur sometime before another action
  – A set of variable binding constraints.
    • Variable constraints are of form \( v=x \), where \( v \) is variable at some step, and \( x \) is either a constant or a variable
  – A set of causal links.
    • Record the purpose of the steps in the plan.
    • E.g., purpose of \( S_i \) is to achieve the precondition \( c \) of \( S_j \).
Generating Plans in STRIPS

• Use principle of least commitment:
  – Make choices only regarding things you care about; leave others for later
  – Leave ordering of steps unspecified if it doesn’t matter for now
  – Plan in which some steps are ordered and others aren’t: partial order plan (as compared to a total order plan):

• “Instantiate”: bind a variable to a constant
  – Example:
    • At(Shakey, x) ➔ At(Shakey, Home)
    • Variable x is bound to constant “Home”

• Fully instantiated plans:
  – Plans in which every variable is bound to a constant
Partial Order vs. Total Order Plans

**Partial Order Plan:**

Start → Left Sock → LeftSockOn → Left Shoe → LeftShoeOn → Finish

Start → Right Sock → RightSockOn → Right Shoe → LeftShoeOn → Finish

**Total Order Plans:**

Start → Right Sock → Left Sock → Right Shoe → Left Shoe → Finish

Start → Left Sock → Right Sock → Right Shoe → Left Shoe → Finish

Start → Right Sock → Right Shoe → Left Shoe → Left Shoe → Finish

Etc.
Solutions in STRIPS

- **Solution**: plan that an agent can execute, and that guarantees achievement of goal

- **Easy way to make guarantee**:  
  - Require only fully instantiated, totally ordered plans

- **However, not satisfactory, because**: 
  - Easier for planners to return partially ordered plan than to arbitrarily select from alternative total orderings 
  - Some agents can perform actions in parallel 
  - Want to maintain flexibility of plan execution, in case other constraints arise later
• Therefore, solution is:
  – A Complete, Consistent plan

• Complete plan:
  – One in which every precondition of every step is achieved by some other step
  – One in which preconditions are not “un-done” by another step prior to its need in a given step

• Consistent plan:
  – One in which there are no contradictions in the ordering or binding constraints
Example Algorithm for Partial-Order Planner (POP)

(See Russell handout, page 356 for more details)

POP: A sound, complete partial order planner using STRIPS representation

```plaintext
function POP(initial, goal, operators) returns plan
    plan ← MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        S_{need}, c ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, S_{need}, c)
        RESOLVE-THREATS(plan)
    end

where:
* c is a precondition of a step S_{need}
* RESOLVE-THREATS: orders steps as needed to ensure intermediate steps don’t undo preconditions needed by other steps
```
Summary of the 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, capturing much of the expressive power of situational calculus.
- It is infeasible to search through space of situations; therefore, search through space of plans.
- Principle of least commitment is preferred.
- POP is a sound and complete algorithm for planning using STRIPS representation.
Hierarchical Architectures

• “Architecture”: a method of implementing a control paradigm, of embodying certain principles in a concrete way

• Hierarchical paradigm:

• 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
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
Advantage/Disadvantage of NHC

• Advantage:
  – Interleaves planning and acting (unlike STRIPS)
    • Plan is changed if world is different from expected

• Disadvantage:
  – Planning decomposition is only appropriate for navigation tasks

Final note: NHC never implemented on physical robot, due to hardware costs
A Second Hierarchical Architecture: NIST RCS & NASREM

- NIST RCS: developed to serve as standard for manufacturers’ development of more intelligent robots
- Similar in design to NHC
  - Primary difference: sensory perception includes preprocessing (feature extraction, or focus of attention)
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
NASREM Endorsed as Standard Model in 1980’s

• Six levels capture specific functionalities:
  – Servo
  – Primitive
  – Elemental move
  – Task
  – Service bay
  – Service mission

• Despite government endorsement, only limited acceptance; considered to detailed and restrictive by many AI researchers
Multiple Levels of NIST RCS & NASREM

- **Level 7**
  - Sensory Processing
  - Global memory
  - World modeling
  - Value judgment
  - Task decomposition

- **Level 6**
  - Sensory Processing
  - Global memory
  - World modeling
  - Value judgment
  - Task decomposition

- **Level 5**
  - Sensory Processing
  - Global memory
  - World modeling
  - Task decomposition

- **Level 4**
  - Sensory Processing
  - Global memory
  - World modeling
  - Task decomposition

- **Level 3**
  - Sensory Processing
  - Global memory
  - World modeling
  - Task decomposition

- **Level 2**
  - Sensory Processing
  - Global memory
  - World modeling
  - Task decomposition

- **Level 1**
  - Sensory Processing
  - Global memory
  - World modeling
  - Task decomposition

**Example levels:**
- Service Mission
- Service Bay
- Task
- Elemental Move
- Primitive
- Servo
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

• As robot gets “smarter”, replace more functions and “move up” autonomy hierarchy
Evaluation of Hierarchical Architectures

- Robots built before 1990 typically used hierarchical style of software organization

- Primary advantage of Hierarchical Paradigm:
  - Provides ordering of relationship between sensing, planning, and acting

- Primary disadvantages of Hierarchical Paradigm:
  - Planning
    - Every update cycle, robot had to update global world model, then plan
  - Sensing and acting are always disconnected
  - Appropriate hierarchical decomposition is application-dependent
  - Uncertainty not well-handled
Preview of Next Class

• Begin looking at Reactive and Behavior-Based Paradigms

• Biological Foundations of Reactive and Behavior-Based Systems