Classical AI Planning: STRIPS Planning

Jane Hsu

 1

Remember: Problem-Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT( p) returns an action
    inputs: p, a percept
    static: s, an action sequence, initially empty
            state, some description of the current world state
            g, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, p)
    if s is empty then
        g ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, g)
        s ← SEARCH( problem)
    action ← RECOMMENDATION(s, state)
    s ← REMAINDER(s, state)
    return action
```

Note: This is *offline* problem-solving. *Online* problem-solving involves acting w/o complete knowledge of the problem and environment

 2

The Planning Problem

To find an executable sequence of actions that achieves a given goal when performed starting in a given state.

START

Initial State

Operator Instances → **PLAN**

Goal State

FINISH

 3

Roots of Planning

- problem solving
- state-space search

- EXAMPLE: “Shakey” the Robot (SRI)
  - a robot that roamed the halls of SRI in the early 1970’s
  - actions were based on STRIPS plans

 4

A Simple Planning Agent

```
function SIMPLE-PLANNING-AGENT(percept) returns an action
    static:        KB, a knowledge base (includes action descriptions)
                   p, a plan (initially, NoPlan)
                   t, a time counter (initially 0)
    local variables:G, a goal
                   current, a current state description
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    current ← STATE-DESCRIPTION(KB, t)
    if p = NoPlan then
        G ← ASK(KB, MAKE-GOAL-QUERY(t))
        p ← IDEAL-PLANNER(current, G, KB)
    if p = NoPlan or p is empty then
        action ← NoOp
    else
        action ← FIRST(p)
        p ← REST(p)
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t+1
    return action
```

5

Algorithm: A Simple Planning Agent

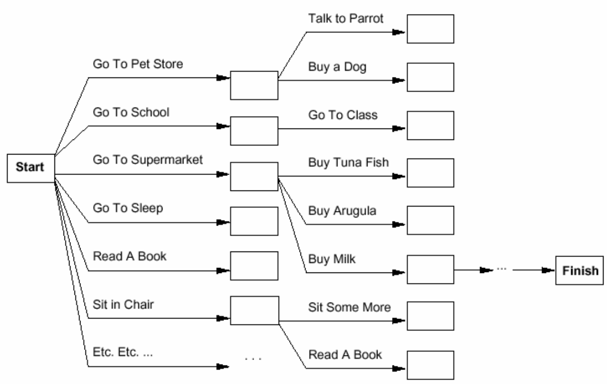
1. Generate a goal to achieve
2. Construct a plan to achieve goal from the current state
3. Execute plan until finished
4. Begin again with new goal
   - Use percepts to build a model of the current world state
   - IDEAL-PLANNER: Given a goal, algorithm generates a plan of actions
   - STATE-DESCRIPTION: given percept, return initial state description in format required by planner
   - MAKE-GOAL-QUERY: used to ask KB what the next goal should be

 6

Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

7

Search vs. planning

Planning systems do the following:

1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

8

Basic Representation for Planning

- Most widely used approach: uses STRIPS language
- states: conjunctions of function-free ground literals (I.e., predicates applied to constant symbols, possibly negated); e.g.,

  At(Home) ∧ ¬Have(Milk) ∧ ¬Have(Bananas) ∧ ¬Have(Drill) …

- goals: also conjunctions of literals; e.g.,

  At(Home) ∧ Have(Milk) ∧ Have(Bananas) ∧ Have(Drill)

  but can also contain variables (implicitly universally quant.); e.g.,

  At(x) ∧ Sells(x, Milk)

 9

Planning in Situation Calculus

$PlanResult(p, s)$ is the situation resulting from executing p in s

$$PlanResult([], s) = s$$
$$PlanResult([a|p], s) = PlanResult(p, Result(a, s))$$

Initial state $At(Home, S_0) \land \neg Have(Milk, S_0) \land \ldots$

Actions as Successor State axioms

$Have(Milk, Result(a, s)) \Leftrightarrow$
$[(a = Buy(Milk) \land At(Supermarket, s)) \lor (Have(Milk, s) \land a \neq \ldots)]$

Query

$s = PlanResult(p, S_0) \land At(Home, s) \land Have(Milk, s) \land \ldots$

Solution

$p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS), \ldots]$

Principal difficulty: unconstrained branching, hard to apply heuristics

Planner vs. Theorem Prover

- Planner: ask for sequence of actions that makes goal true if executed
- Theorem prover: ask whether query sentence is true given KB

 11

The Frame Problem

- Assumption: actions have local effects
- We need **frame axioms** for each action and each fluent that does not change as a result of the action
  - example: frame axioms for *move*:
  - If a block is on another block and *move* is not relevant, it will stay the same.
    - Positive:
      $$[On(x, y, s) \wedge (x \neq u)] \rightarrow On(x, y, do(move(u, v, z), s))$$
    - Negative
      $$(\neg On(x, y, s) \wedge [(x \neq u) \vee (y \neq z)]) \rightarrow \neg On(x, y, do(move(u, v, z), s)$$

 12

Other Problems in Planning

- **The qualification problem**: qualifying the antecedents for all possible exception. Needs (but *impossible*!) to enumerate all exceptions
  - ~heavy and ~glued and ~armbroken → can-move
  - ~bird and ~cast-in-concrete and ~dead… → flies
  - Solutions: default logics, nonmonotonic logics
- **The ramification problem:**
  - If a robot carries a package, the package will be where the robot is. But what about the frame axiom, when can we infer about the effects of the actions and when we cannot.
- Not everything true can be inferred
  On(C,F1) remains true but cannot be inferred

 13

Assumptions in Classical Planning

- Percepts
  - Perfect perception
  - Complete knowledge agent is omniscient
- Actions
  - Instantaneous actions
  - Atomic time
  - No concurrent actions allowed
  - Deterministic actions (effects are completely specified)
- Environment
  - Static environment
  - Completely observable environment
  - Agent is the sole cause of change in the world

 14

Assumptions

- Closed World Assumption
  - If ground terms cannot be proved to be true, they can be assumed to be false.
- Domain Closure Assumption
  - All objects in the domain are explicitly named.
- Unique Names Assumption
  - If ground terms cannot be proved to be equal, they can be assumed to be unequal.

 15

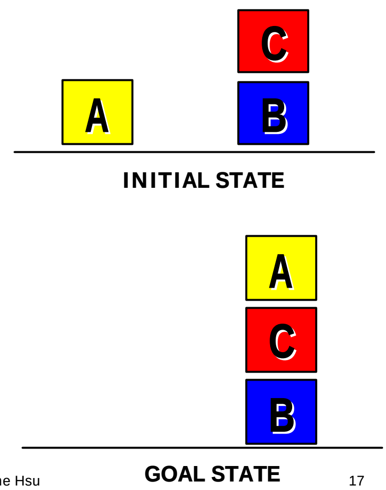
STRIPS Representation

- STRIPS is the simplest and the second oldest representation of operators in AI.
- When that the initial state is represented by a database of positive facts, STRIPS can be viewed as being simply a way of specifying an update to this database.

 16

Example: “The Blocks World”

- Domain:
  - set of cubic blocks sitting on a table
- Actions:
  - blocks can be stacked
  - can pick up a block and move it to another position
  - can only pick up one block at a time
- Goal:
  - to build a specified stack of blocks



 17

Representing States & Goals

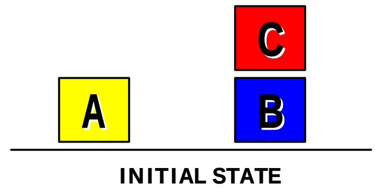
- STRIPS:
  - describes states & operators in a restricted language
- States:
  - a conjunction of “facts” (ground literals that do not contain variable symbols)
- Goals:
  - a conjunction of positive literals

 18

Representing States & Goals

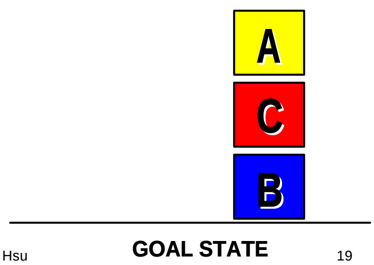
Initial State:

ontable(A) $\wedge$
ontable(B) $\wedge$
on(C, B) $\wedge$
clear(A) $\wedge$
clear(C) $\wedge$
handempty



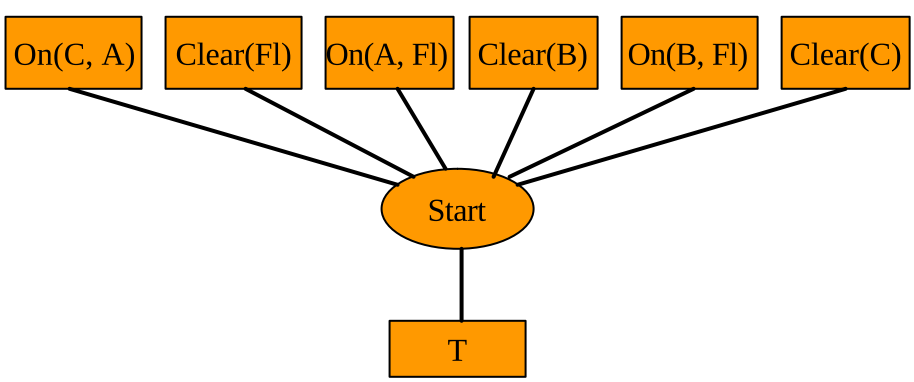
Goal:

ontable(B) $\wedge$
on(C,B) $\wedge$
on(A,C) $\wedge$
clear(A) $\wedge$
handempty



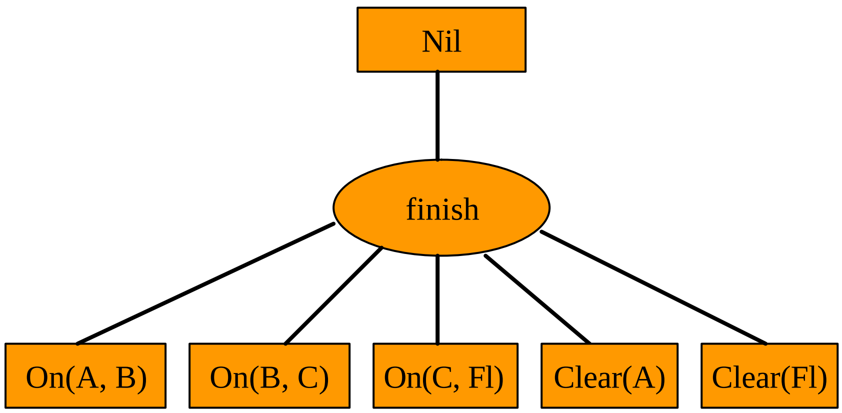
 19

Graphical Representation: Initial State



 20

Graphical Representation: Goal State



Representing Actions

1. Action description
   Name: pickup(x)
2. Precondition
   Preconditions: ontable(x), clear(x), handempty
3. Effect
   Effect: holding(x), ~ontable(x), ~clear(x), ~handempty

Actions in "Blocks World"

- **pickup(x)**
  - picks up block '*x*' from table
- **putdown(x)**
  - if holding block '*x*', puts it down on table
- **stack(x,y)**
  - if holding block '*x*', puts it on top of block '*y*'
- **unstack(x,y)**
  - picks up block '*x*' that is currently on block '*y*'

An operator is APPLICABLE if all preconditions are satisfied.

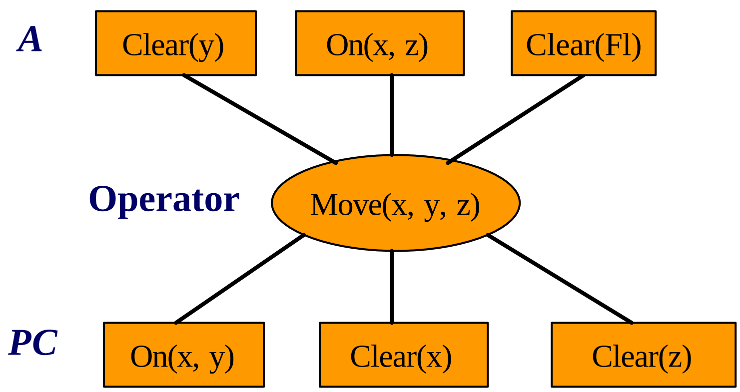
 23

Blocks World - Operator

- Move(x, y, z)
  - Move block x that is above y to above z
  - ***PC***: On(x,y), Clear(x), Clear(z)
  - ***D***: Clear(z), On(x, y)
  - ***A***: On(x,z), Clear(y), Clear(Fl)

 24

Graphical Representation: Operator

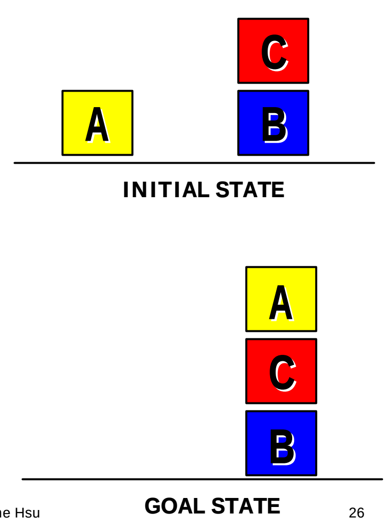


 25

Progression Situation-Space

Algorithm:

1. Start from initial state
2. Find all operators whose preconditions are true in the initial state
3. Compute effects of operators to generate successor states
4. Repeat steps 2-3, until a new state satisfies the goal conditions

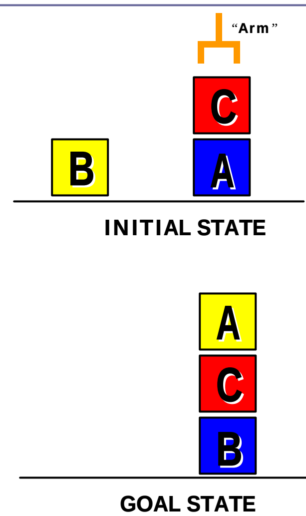


 26

Regression Situation-Space

Algorithm:

1. Start with goal node corresponding to goal to be achieved
2. Choose an operator that will *add* one of the goals
3. Replace that goal with the operator's preconditions
4. Repeat steps 2-3 until you have reached the initial state



 27

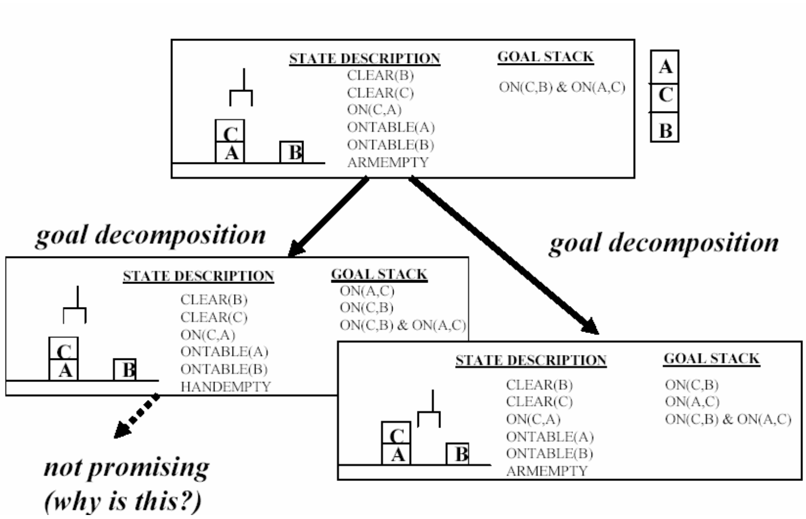
STRIPS: Goal-Stack Planning

Given a goal stack:

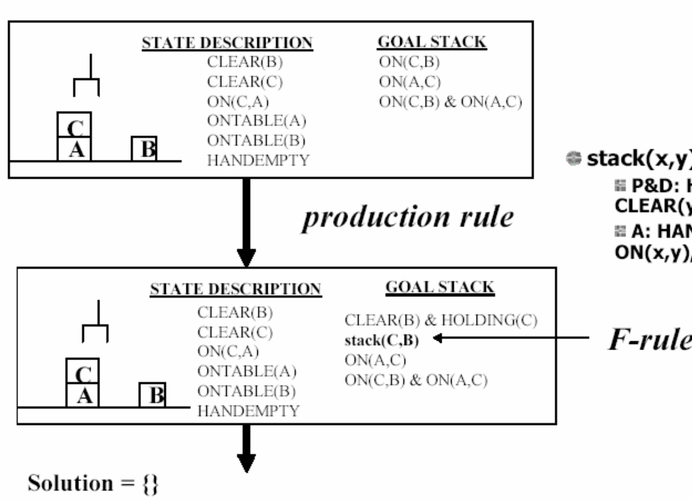
1. Initialize: Push the goal to the stack.
2. If the top of the stack is satisfied in the current state, pop.
3. Otherwise, if the top is a conjunction, push the individual conjuncts to the stack.
4. Otherwise, check if the add-list of any operator can be unified with the top, push the operator and its preconditions to the stack.
5. If the top is an action, pop and execute it.
6. Loop 2-5 till stack is empty.

 28

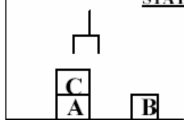
STRIPS in Action



STRIPS in Action (Continued)

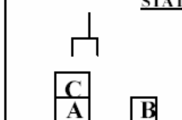


- **stack(x,y)**
  - **P&D: HOLDING(x), CLEAR(y)**
  - **A: HANDEMPTY, ON(x,y), CLEAR(x)**



STATE DESCRIPTION	GOAL STACK
CLEAR(B)	CLEAR(B) & HOLDING(C)
CLEAR(C)	**stack(C,B)**
ON(C,A)	ON(AC,)
ONTABLE(A)	ON(C,B) & ON(A,C)
ONTABLE(B)	
HANDEMPTY	

production rule

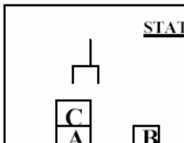


STATE DESCRIPTION	GOAL STACK
CLEAR(B)	HOLDING(C)
CLEAR(C)	CLEAR(B)
ON(C,A)	CLEAR(B) & HOLDING(C)
ONTABLE(A)	**stack(C,B)**
ONTABLE(B)	ON(AC,)
HANDEMPTY	ON(C,B) & ON(A,C)

goal decomposition

Solution = {}

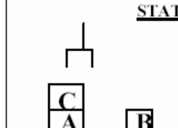
 31



STATE DESCRIPTION	GOAL STACK
CLEAR(B)	HOLDING(C)
CLEAR(C)	CLEAR(B)
ON(C,A)	CLEAR(B) & HOLDING(C)
ONTABLE(A)	**stack(C,B)**
ONTABLE(B)	ON(AC,)
HANDEMPTY	ON(C,B) & ON(A,C)

production rule

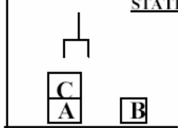
- **unstack(x,y)**
  - **P&D: HANDEMPTY, CLEAR(x), ON(x,y)**
  - **A: HOLDING(x), CLEAR(y)**



STATE DESCRIPTION	GOAL STACK
CLEAR(B)	HANDEMPTY & CLEAR(C) & ON(C, y)
CLEAR(C)	**unstack(C, y)**
ON(C,A)	CLEAR(B)
ONTABLE(A)	CLEAR(B) & HOLDING(C))
ONTABLE(B)	**stack(C,B)**
HANDEMPTY	ON(AC,)
	ON(C,B) & ON(A,C)

Solution = {}

 32



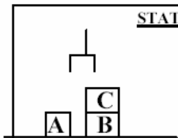
STATE DESCRIPTION

CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK

HANDEMPTY & CLEAR(C) & ON(C, y)
unstack(C, y)
CLEAR(B)
CLEAR(B) & HOLDING(C)
stack(C,B)
ON(AC,)
ON(C,B) & ON(A,C)

***Substitute* {A/y}, *then apply unstack*(C,A) *then stack*(C,B)**



STATE DESCRIPTION

CLEAR(C)
CLEAR(A)
ON(C,B)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

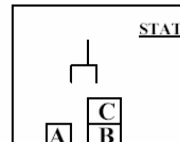
GOAL STACK

ON(A,C)
ON(C,B) & ON(A,C)

- **unstack(x,y)**
  - **P&D: HANDEMPTY, CLEAR(x), ON(x,y)**
  - **A: HOLDING(x), CLEAR(y)**
- **stack(x,y)**
  - **P&D: HOLDING(x), CLEAR(y)**
  - **A: HANDEMPTY, ON(x,y), CLEAR(x)**

Solution = {unstack(C,A), stack(C,B)}

 33



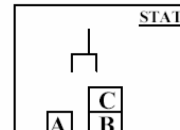
STATE DESCRIPTION

CLEAR(C)
CLEAR(A)
ON(C,B)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK

ON(A,C)
ON(C,B) & ON(A,C)

production rule



STATE DESCRIPTION

CLEAR(C)
CLEAR(A)
ON(C,B)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

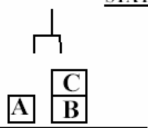
GOAL STACK

CLEAR(C) & HOLDING(A)
stack(A,C)
ON(C,B) & ON(A,C)

- **stack(x,y)**
  - **P&D: HOLDING(x), CLEAR(y)**
  - **A: HANDEMPTY, ON(x,y), CLEAR(x)**

Solution = {unstack(C,A), stack(C,B)}

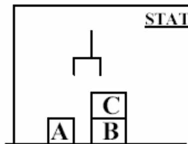
 34



STATE DESCRIPTION	GOAL STACK
CLEAR(C) CLEAR(A) ON(C,B) ONTABLE(A) ONTABLE(B) HANDEMPTY	CLEAR(C) HOLDING(A) CLEAR(C) & HOLDING(A) **stack(A,C)** ON(C,B) & ON(A,C)

goal decomposition

production rule

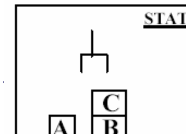


STATE DESCRIPTION	GOAL STACK
CLEAR(C) CLEAR(A) ON(C,B) ONTABLE(A) ONTABLE(B) HANDEMPTY	ONTABLE(A) & CLEAR(A) & HANDEMPTY **pickup(A)** CLEAR(C) & HOLDING(A) **stack(A,C)** ON(C,B) & ON(A,C)

- **pickup(x)**
  - **P&D: ONTABLE(x), CLEAR(x), HANDEMPTY**
  - **A: HOLDING(x)**

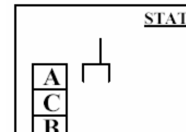
Solution = {unstack(C,A), stack(C,B)}

 35



STATE DESCRIPTION	GOAL STACK
CLEAR(C) CLEAR(A) ON(C,B) ONTABLE(A) ONTABLE(B) HANDEMPTY	ONTABLE(A) & CLEAR(A) & HANDEMPTY **pickup(A)** CLEAR(C) & HOLDING(A) **stack(A,C)** ON(C,B) & ON(A,C)

Apply pickup(A)
And then stack(A,C)



STATE DESCRIPTION	GOAL STACK
ON(A,C) ON(C,B) CLEAR(A) ONTABLE(B) HANDEMPTY	NIL

- **pickup(x)**
  - **P&D: ONTABLE(x), CLEAR(x), HANDEMPTY**
  - **A: HOLDING(x)**
- **stack(x,y)**
  - **P&D: HOLDING(x), CLEAR(y)**
  - **A: HANDEMPTY, ON(x,y), CLEAR(x)**

Solution plan = {unstack(C,A), stack(C,B), pickup(A), stack(A,C)}

 36