RN, Chapter 11

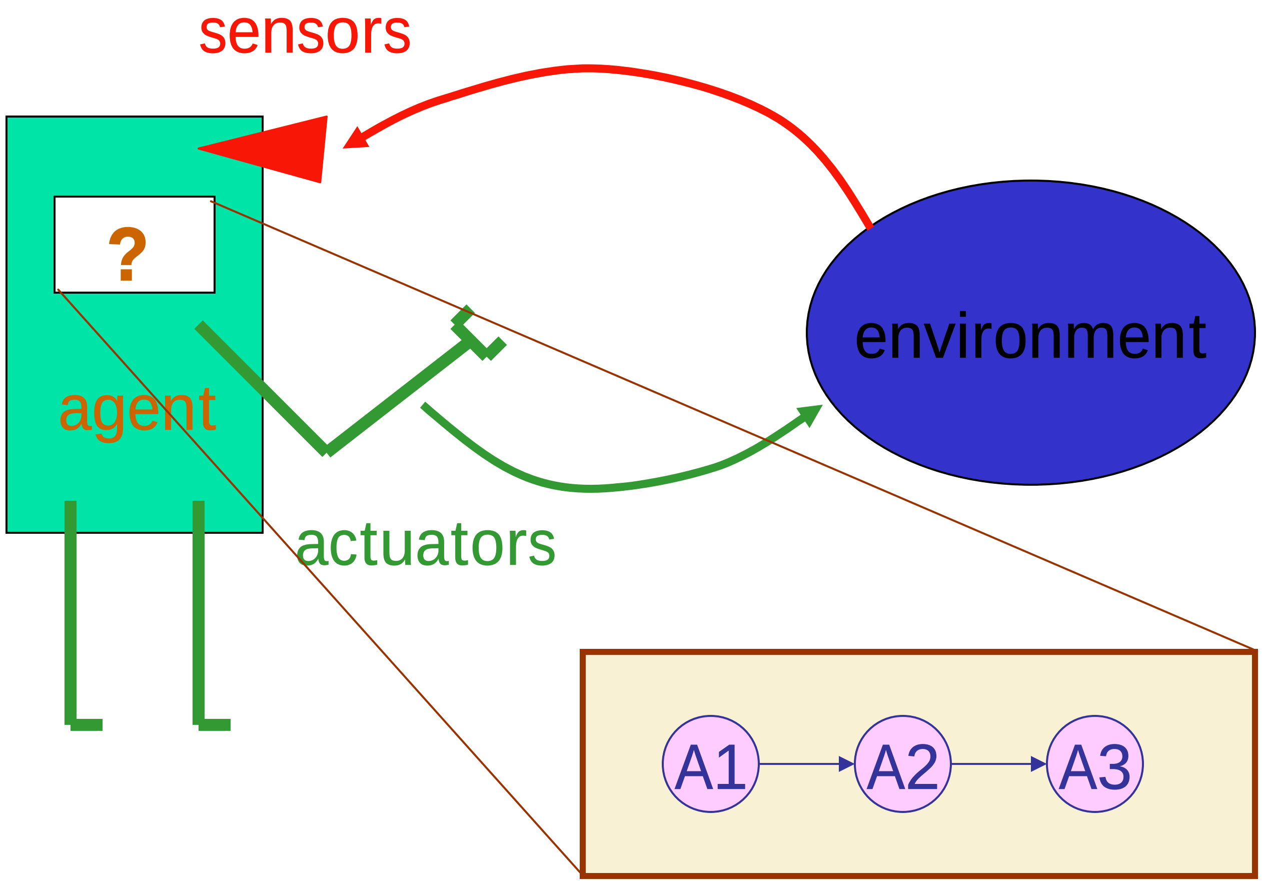
Planning

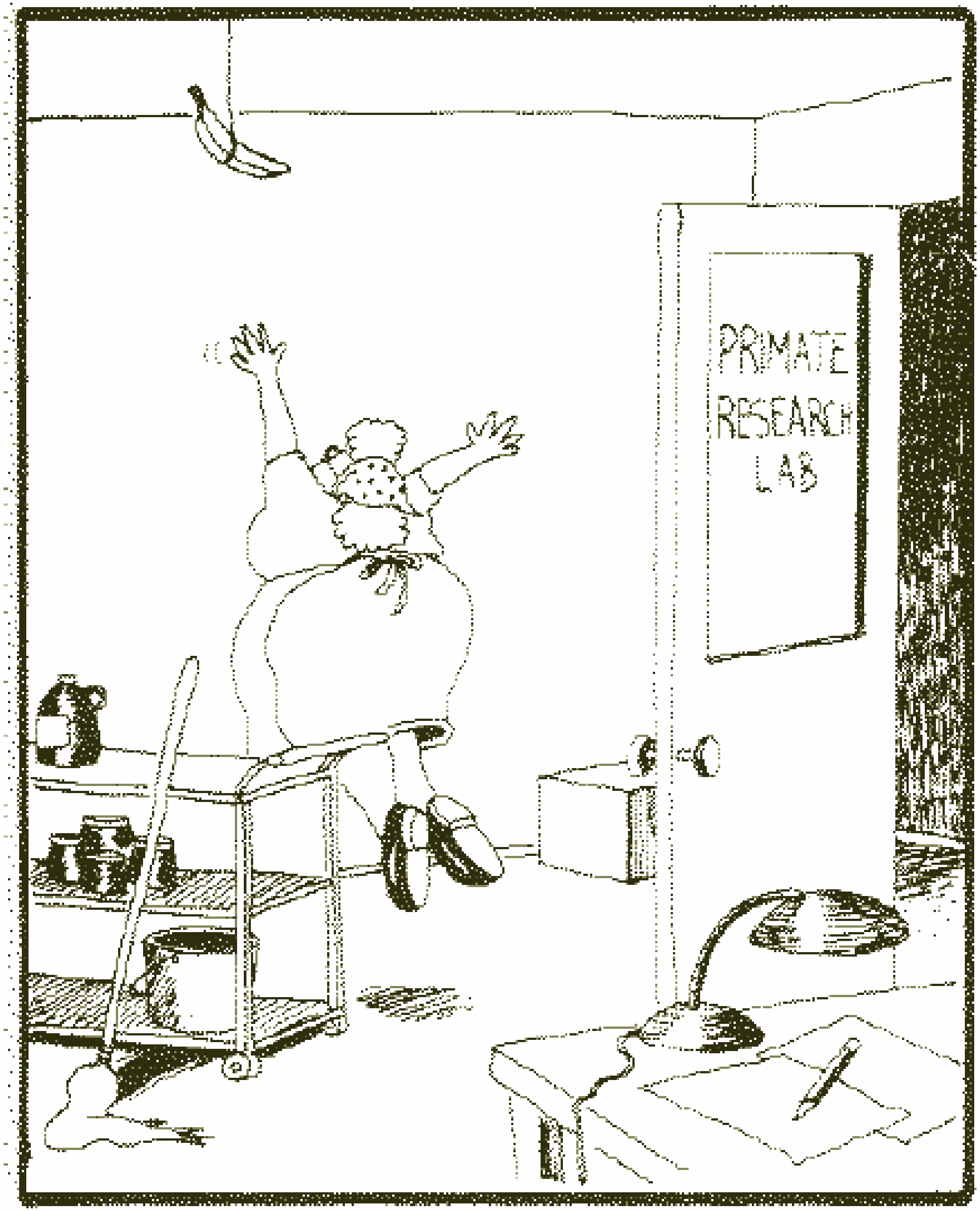
Some material taken from D. Lin, J-C Latombe

Logical Agents

- Reasoning [Ch 6]
- Propositional Logic [Ch 7]
- Predicate Calculus
  - Representation [Ch 8]
  - Inference [Ch 9]
- Implemented Systems [Ch 10]
- **Planning [Ch 11]**
  - Representations in planning (Strips)
  - Representation of action:
    preconditions + effects
  - Forward planning
  - Backward chaining
  - Partial-order planning

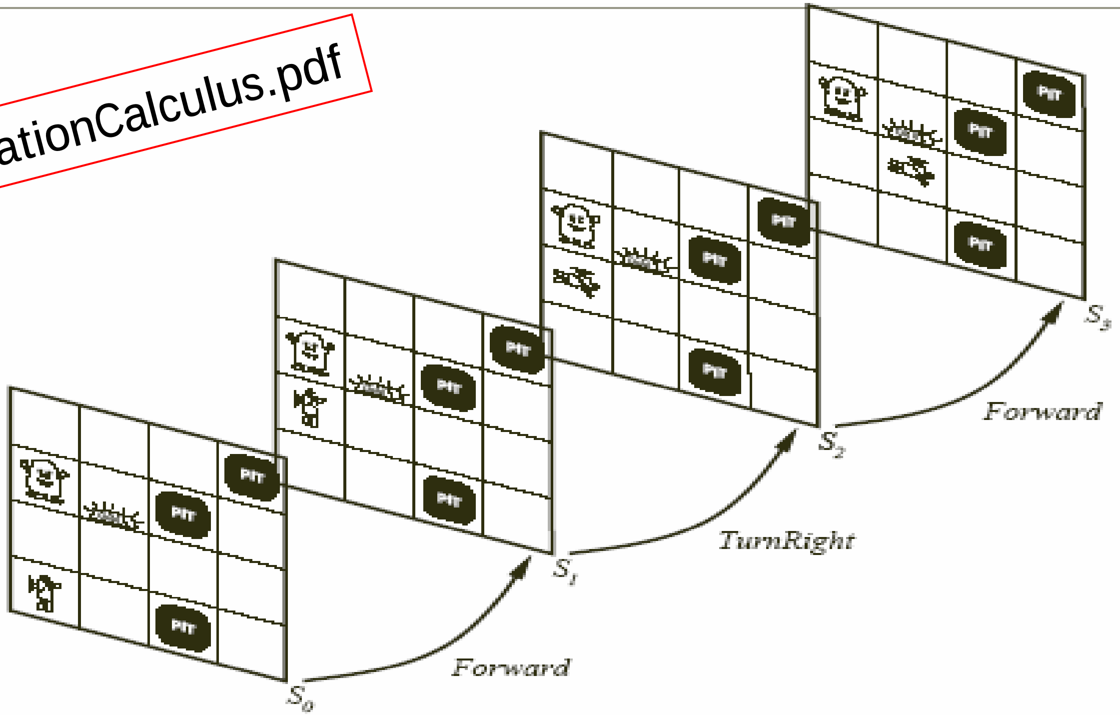
Planning Agent




PRIMATE
RESEARCH
LAB

Updating State, Based on Action

See 10.3-SituationCalculus.pdf



$$S_1 = \text{Result}(\text{Forward}, S_0)$$

$$S_2 = \text{Result}(\text{TurnRight}, S_1)$$
$$= \text{Result}(\text{TurnRight}, \text{Result}(\text{Forward}, S_0))$$

$$S_3 = \text{Result}(\text{Forward}, S_2)$$
$$= \text{Result}(\text{Forward}, \text{Result}(\text{TurnRight}, \text{Result}(\text{Forward}, S_0)))$$

$$\text{Result}: \text{Action} \times \text{State} \mapsto \text{State}$$

Planning in Situation Calculus

- Given:
  - Initial: $At(Home, S_0)$ & $\neg Have(Milk, S_0)$
  - Goal: $\exists s$ At(Home,s) & Have(Milk,s)
  - Operators: $\forall a$, s Have(Milk, Result(a,s)) $\Leftrightarrow$
    [(a = Buy(Milk) & At(Store, s))
    v (Have(Milk,s) & a $\neq$Drop(Milk))]

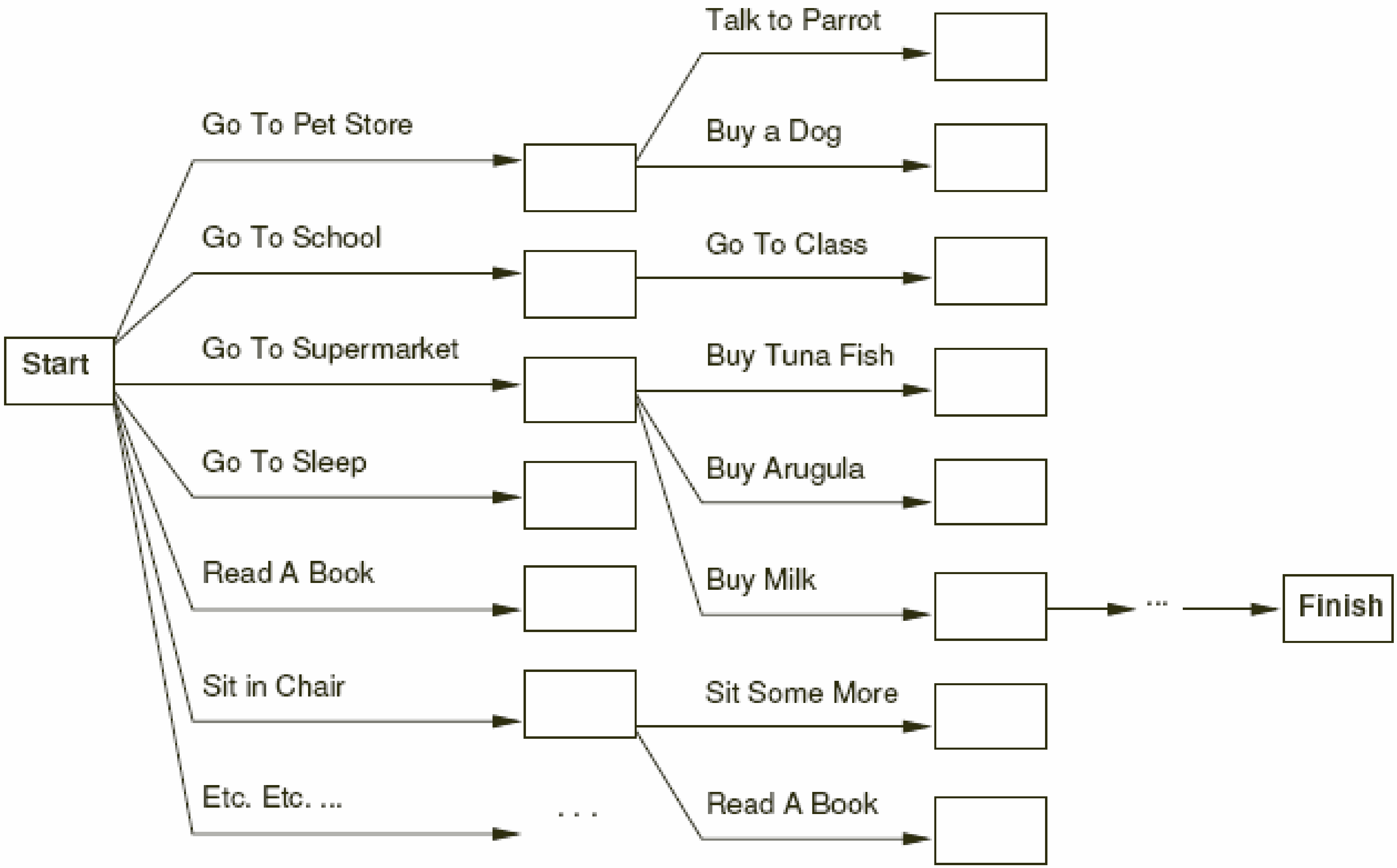
    ...
- Find: Sequence of operators $[o_1, ..., o_k]$ where
  $S = Result(o_k, Result(... Result(o_1, S_0) ...))$
  s.t. At(Home, S) & Have(Milk, S)

- but... Standard Problem Solving is inefficient
  As goal is “black box”,   just generate-&-test!

Naïve Problem Solving

- Goal:
  "At home; have Milk, Bananas, and Drill"
  $\exists$ s At(Home, s) & Have(Milk, s) & Have(Banana, s) & Have(Drill, s)
- Initial: "None of these; at home"

  At(Home, S_0) & $\neg$Have(Milk, S_0) & $\neg$Have(Banana, S_0) & $\neg$Have(Drill, S_0)
- Operators:
  Goto(y), SitIn(z), Talk(w), Buy(q), ...


Start
Go To Pet Store
Go To School
Go To Supermarket
Go To Sleep
Read A Book
Sit in Chair
Etc. Etc. ...
Talk to Parrot
Buy a Dog
Go To Class
Buy Tuna Fish
Buy Arugula
Buy Milk
Sit Some More
Read A Book
Finish

General Issues

- Done?
- General problems:
  - Problem solving is P-space complete
  - Logical inference is only semidecidable
  - .. plan returned may go from initial to goal, but extremely inefficiently (NoOp, [A, A^{-1}], ...)
- Solution
  - Restrict language
  - Special purpose reasoner

  $\Rightarrow$ **PLANNER**

Key Ideas

1. Open up representation … to connect States to Actions
   If goal includes "Have(Milk)", and "Buy(x) achieves Have(x)", then consider action "Buy(Milk)"

2. Add actions ANYWHERE in plan ... Not just to front!
   Order of adding actions ≠ order of execution!
   Eg, can decide to include Buy(Milk) BEFORE deciding where? ... how to get there? . . .
   Note: Exploits decomposition:
   doesn't matter which Milk-selling store,
   whether agent currently has Drill, . . .
   . . . avoid arbitrary early decisions …

3. Subgoals tend to be nearly independent
   ⇒ divide-&-conquer
   Eg, going to store does NOT interfere with borrowing from neighbor…

Goal of Planning

- Choose actions to achieve a certain goal
- Isn't PLANNING ≡ Problem Solving ?
- Difficulties with problem solving:

  Successor function is a **black box**:

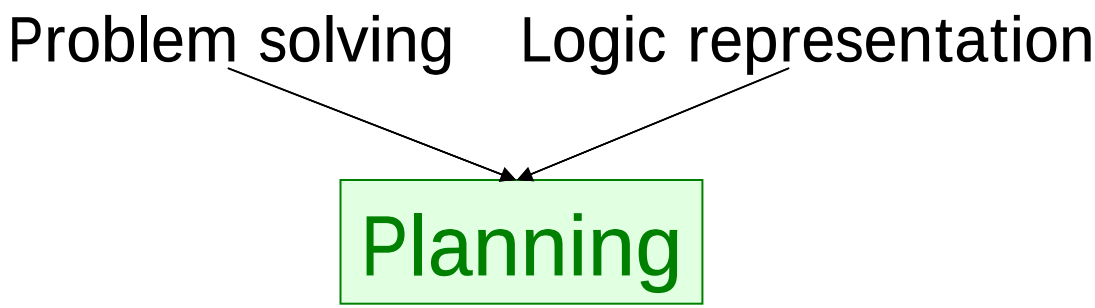
  it must be "applied" to a state to know

  - which actions are possible in each state
  - the effects of each action

Representations in Planning

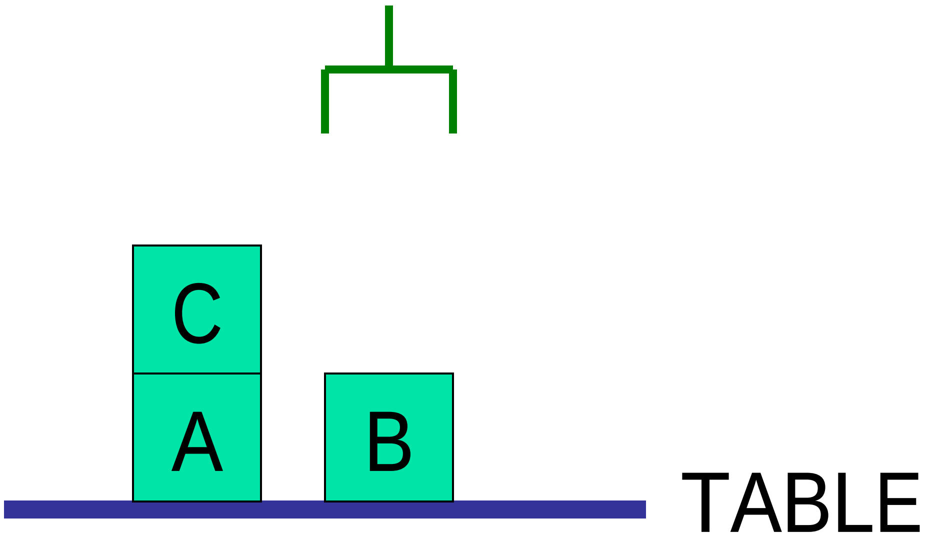
Planning opens up the black-boxes by using logic to represent:

- Actions
- States
- Goals



One possible language: STRIPS

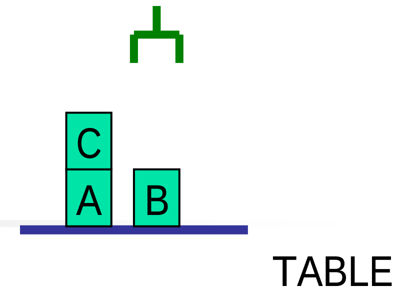
State Representation

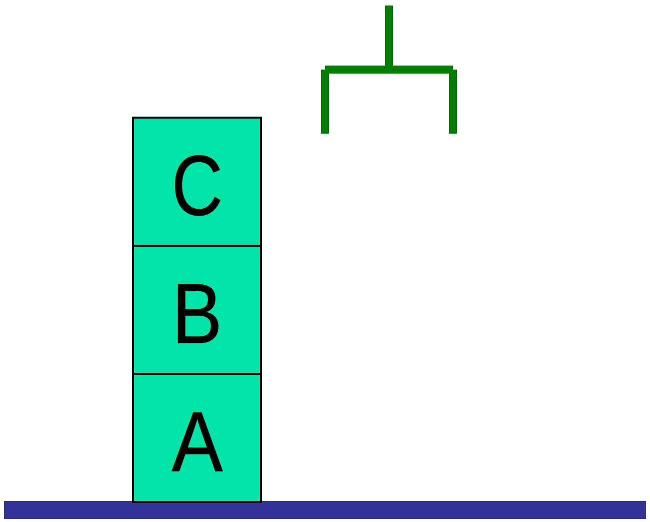


Conjunction of propositions:

BLOCK(A), BLOCK(B), BLOCK(C),
ON(A,TABLE), ON(B,TABLE), ON(C,A),
CLEAR(B), CLEAR(C), HANDEMPTY

Goal Representation



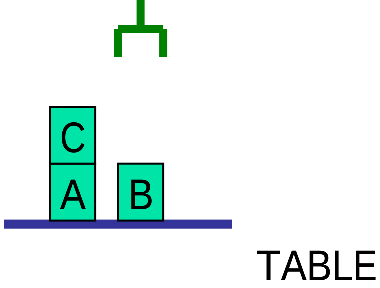


Conjunction of propositions:

ON(A,TABLE), ON(B,A), ON(C,B)

Goal *G* is achieved in state S
iff
all the propositions in *G* are in S

Action Representation



Unstack(x, y)

- P = HANDEMPTY, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
- E = ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x), ¬ ON(x, y), CLEAR(y)

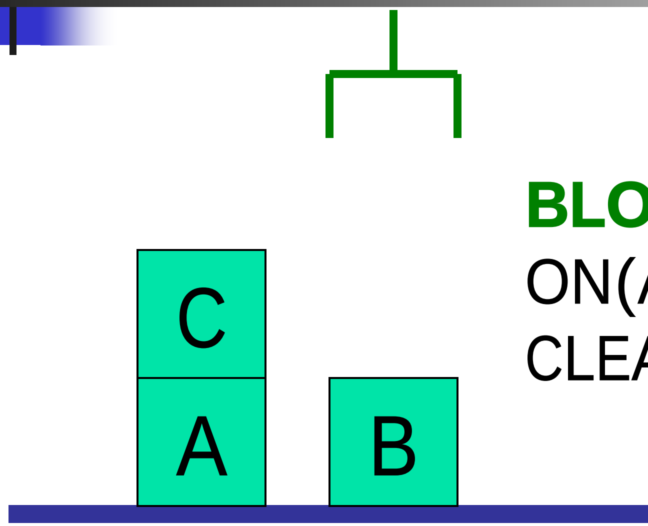
Effect: list of literals

Precondition: conjunction of propositions

"¬" means: Remove HANDEMPTY from state

Means: Add HOLDING(x) to state

Example

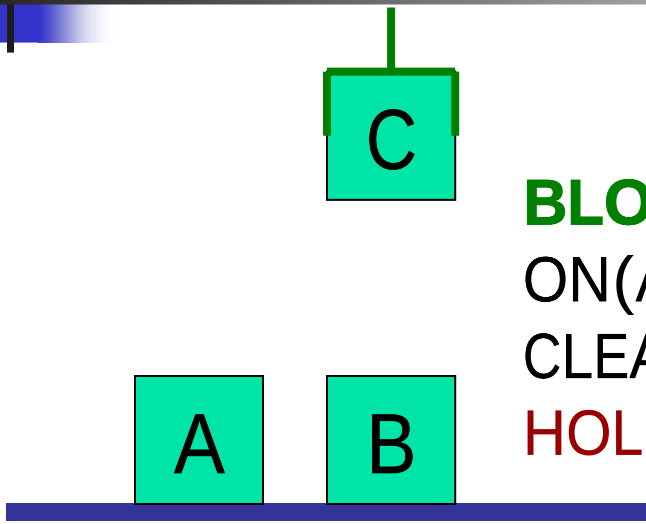


BLOCK(A), BLOCK(B), **BLOCK(C)**, ON(A,TABLE), ON(B,TABLE), **ON(C,A)**, CLEAR(B), **CLEAR(C)**, **HANDEMPTY**

Unstack(C,A)

- P = HANDEMPTY, BLOCK(C), BLOCK(A), CLEAR(C), ON(C,A)
- E = ¬HANDEMPTY, ¬CLEAR(C), HOLDING(C), ¬ON(C,A), CLEAR(A)

Example

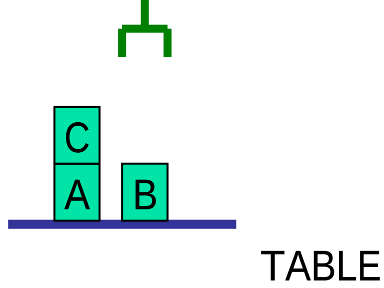


BLOCK(A), BLOCK(B), **BLOCK(C)**,
ON(A,TABLE), ON(B,TABLE), ~~**ON(C,A)**~~,
CLEAR(B), ~~**CLEAR(C)**~~, ~~**HANDEMPTY**~~
HOLDING(C), CLEAR(A)

Unstack(C,A)

- P = HANDEMPTY, BLOCK(C), BLOCK(A), CLEAR(C), ON(C,A)
- E = ¬HANDEMPTY, ¬CLEAR(C), HOLDING(C), ¬ON(C,A), CLEAR(A)

Action Representation



Unstack(x,y)

- P = HANDEMPTY, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
- E = ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x), ¬ON(x,y), CLEAR(y)

Stack(x,y)

- P = HOLDING(x), BLOCK(x), BLOCK(y), CLEAR(y)
- E = ON(x,y), ¬CLEAR(y), ¬HOLDING(x), CLEAR(x), HANDEMPTY

Pickup(x)

- P = HANDEMPTY, BLOCK(x), CLEAR(x), ON(x,TABLE)
- E = ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x), ¬ON(x,TABLE)

PutDown(x)

- P = HOLDING(x)
- E = ON(x,TABLE), ¬HOLDING(x), CLEAR(x), HANDEMPTY

Summary of STRIPS language features

- Representation of **states**
  - Decompose the world into logical conditions;
    state ≡ *conjunction of positive literals*
  - *Closed world assumption*:
    Conditions not mentioned in state assumed to be *false*
- Representation of **goals**
  - Partially specified state;
    conjunction of positive ground literals
  - A goal ***g*** is *satisfied* at state ***s*** iff
    s contains all literals in goal ***g***

Summary of STRIPS language features

Representations of *actions*

- Action = PRECONDITION + EFFECT
  - *Header:*
    - Action name and parameter list
  - *Precondition:*
    - conj of function-free literals
  - *Effect:*
    - conj of function-free literals
    - Add-list & delete-list

Semantics

Executing action a
in state s
produces state s'

- s' is same as s except
  - Every positive literal *P* in *a:Effect* is added to *s*
  - Every negative literal *¬P* in *a:Effect* is removed from *s*

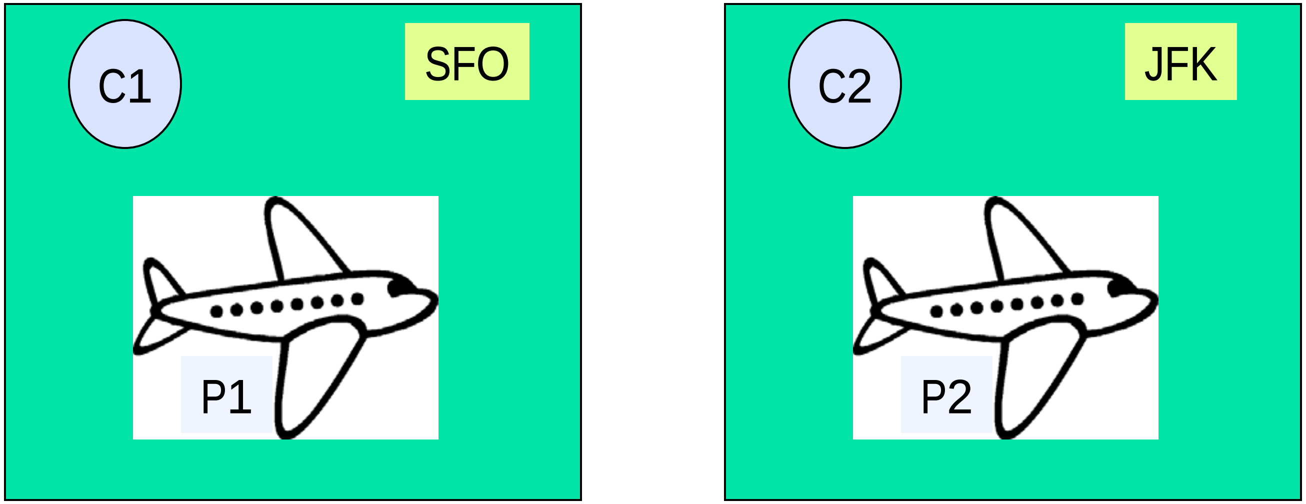
- STRIPS assumption:
  Every literal NOT in the effect remains unchanged
  - (avoids representational frame problem)

Expressiveness

- STRIPS is not arbitrary FOL
  - Important limit: *function-free literals*
  - Allows for propositional representation
- Function symbols lead to infinitely many states and actions
- Recent extension:
  Action Description language (ADL)

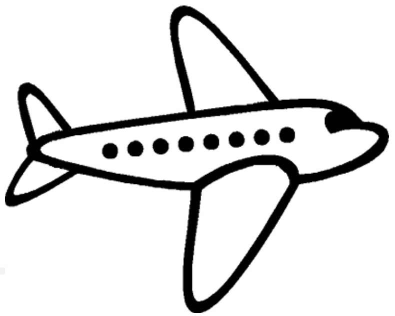
Example:
Air Cargo Transport

Init(Cargo(C1) & Cargo(C2) & Plane(P1) & Plane(P2) & Airport(JFK) & Airport(SFO) & At(C1, SFO) & At(C2,JFK) & At(P1,SFO) & At(P2,JFK))



Goal(At(C1,JFK) & At(C2,SFO))

Example:
Air Cargo Transport



Init(Cargo(C1) & Cargo(C2) & Plane(P1) & Plane(P2) & Airport(JFK) & Airport(SFO) & At(C1, SFO) & At(C2,JFK) & At(P1,SFO) & At(P2,JFK))

Goal(At(C1,JFK) & At(C2,SFO))

Action(Load(c,p,a)
PRECOND: *At(c,a) &At(p,a) &Cargo(c) & Plane(p) &Airport(a)*
EFFECT: *¬At(c,a) &In(c,p))*
Action(Unload(c,p,a)
PRECOND: *In(c,p) & At(p,a) &Cargo(c) & Plane(p) &Airport(a)*
EFFECT: *At(c,a) & ¬In(c,p))*
Action(Fly(p,from,to)
PRECOND: *At(p,from) & Plane(p) & Airport(from) & Airport(to)*
EFFECT: *¬At(p,from) & At(p,to))*

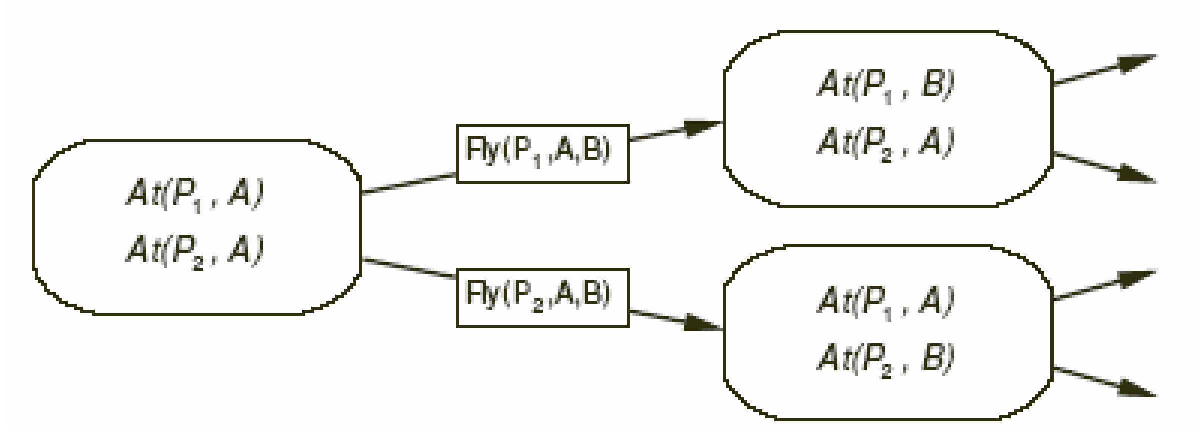
[Load(C1,P1,SFO), Fly(P1,SFO,JFK), Unload(C1, P1, JFK), Load(C2,P2,JFK), Fly(P2,JFK,SFO), Unload(C2, P2, SFO)]

Planning with State-space Search

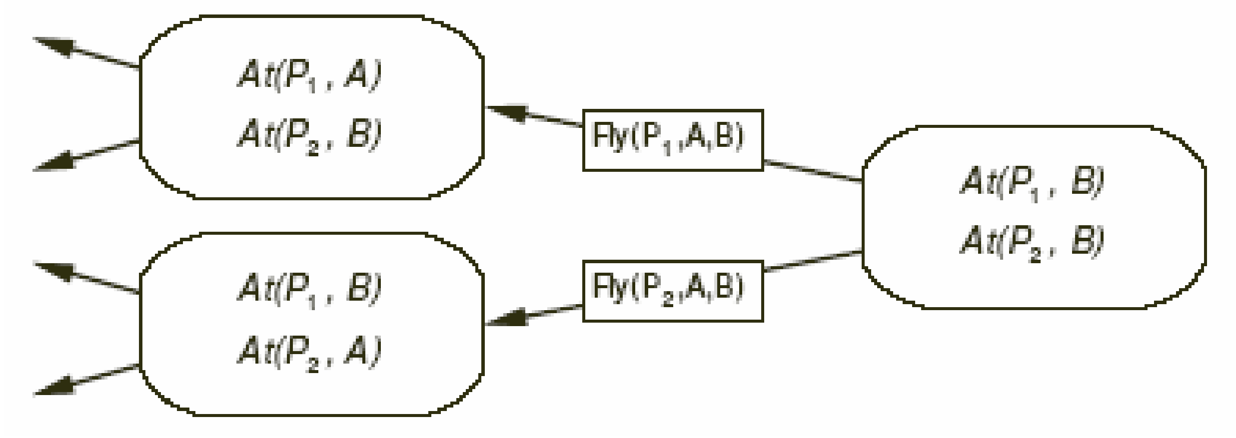
- Forward search vs Backward search
- Progression planners
  - Forward state-space search
  - Consider the *effects* of all possible actions in a given state
- Regression planners
  - Backward state-space search
  - To achieve a goal,
    what must have been true in the *previous* state

Progression vs Regression

- Progressive



- Regressive

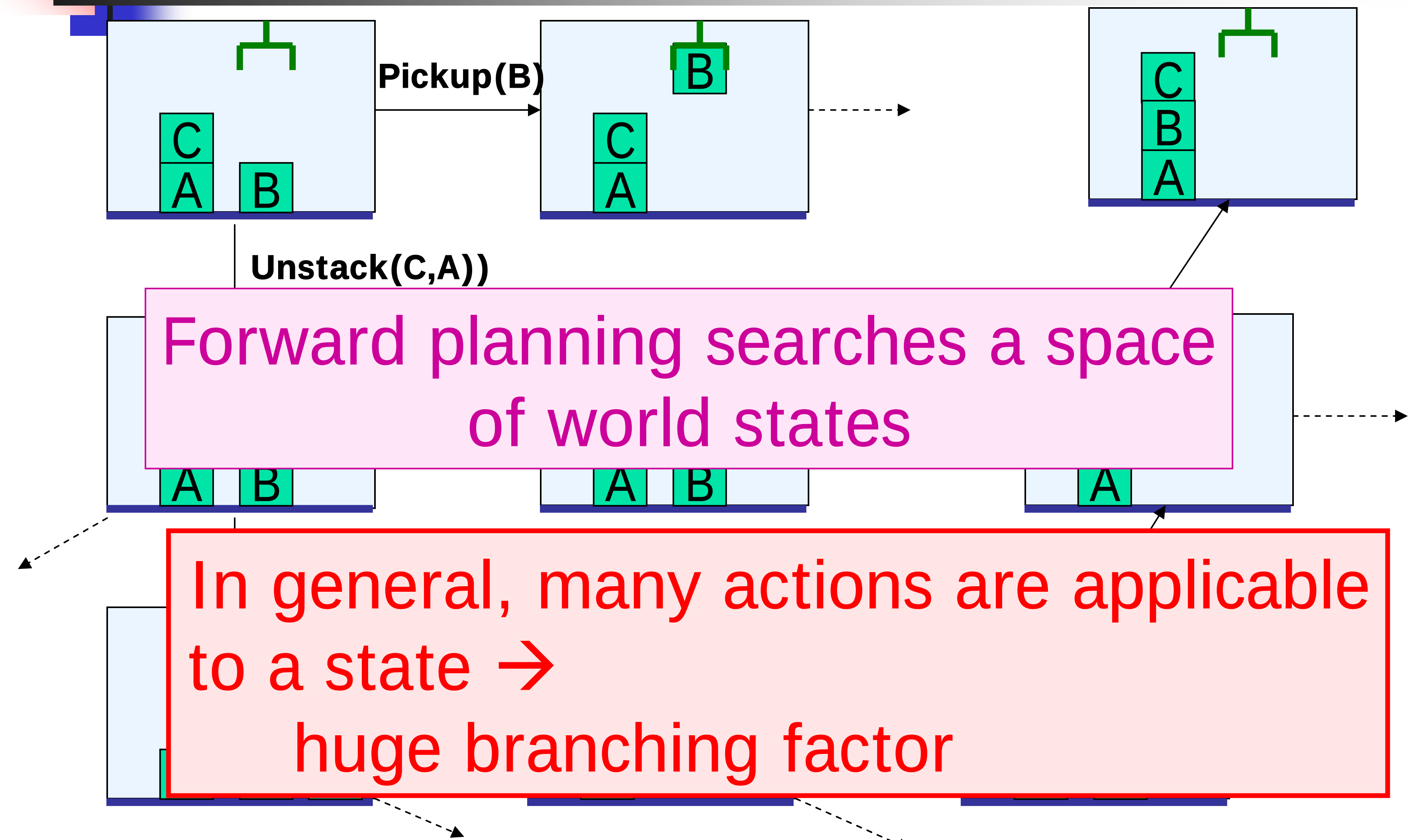


Progression Planning Algorithm

- Formulation as state-space search problem:
  - Initial state = initial state of the planning problem
    - ... literals not appearing are *false*
  - Actions = (just actions whose preconditions are satisfied)
    - Add positive effects, delete negative effects
  - Goal test = does the state satisfy the goal?
  - Step cost = each action costs 1
- Any graph search that is complete is a complete planning algorithm.
  (No functions)

- Inefficient:
  (1) irrelevant action problem
  (2) good heuristic required for efficient search

Progression (Forward) Planning



Forward planning searches a space of world states

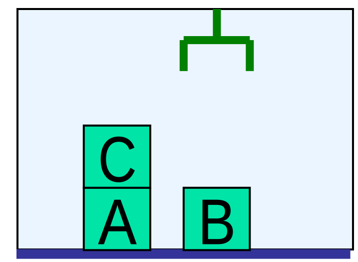
In general, many actions are applicable to a state → huge branching factor

Regression (Backward Chaining)

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

Stack(C,B)

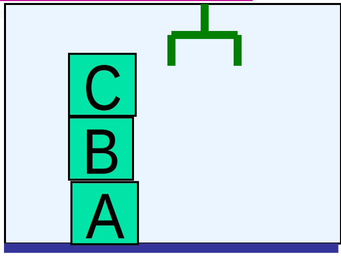
ON(B,A), HOLDING(C), CLEAR(B)



Typically...
#[actions relevant to a goal] <
#[actions applicable to a state]
→
Backward chaining has smaller branching factor than forward planning

Backw

Backward planning searches a space of goals



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

Stack(C,B)

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

Pickup(C)

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

Stack(B,A)

CLEAR(A), HOLDING(B), CLEAR(C), ON(C,TABLE)

Pickup(B)

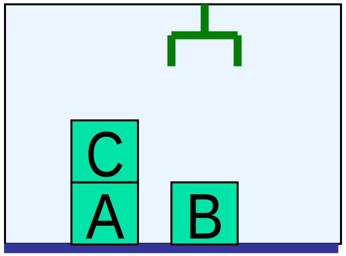
CLEAR(A), HANDEMPTY, CLEAR(B), ON(B,TABLE), CLEAR(C), ON(C,TABLE)

Putdown(C)

CLEAR(A), HOLDING(C), CLEAR(B), ON(B,TABLE)

Unstack(C,A)

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



Regression Algorithm

- How to determine predecessors?
  - What *S* can lead to goal *G*, by applying an action *a* ?

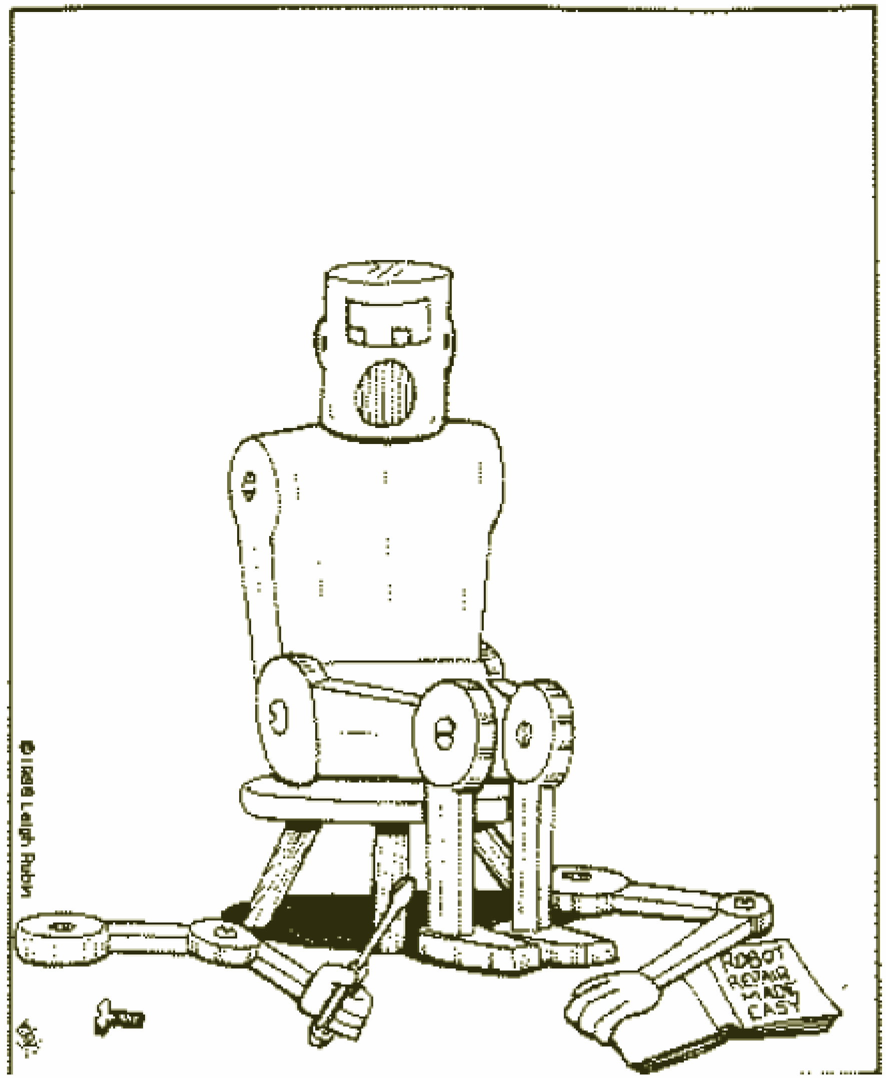
    Goal state = *At(C1, B) & At(C2, B) & … & At(C20, B)*

    Action relevant for first conjunct*: Unload(C1,p,B)*

    (Works only if pre-conditions are satisfied)

    Previous state= *In(C1, p) & At(p, B) & At(C2, B) & … & At(C20, B)*

    Subgoal At(C1,B) should not be present in this state.
- Actions must not undo desired literals (consistent)
- Main advantage:
  Only relevant actions are considered!
  - Often much smaller branching factor than forward search


ROBOT REPAIR MADE EASY
Leigh Rubin

Heuristics for State-space Search

- Neither progression nor regression are efficient ... without a good heuristic.
  - How many actions are needed to achieve the goal?
  - Exact solution is NP-hard, ... need a good heuristic:
- Two ways to find admissible heuristic:
  - Optimal solution to relaxed problem
    - Remove all preconditions from actions
  - Subgoal independence assumption:

    Approximate
      cost of solving a conjunction of subgoals
    by
      sum of the costs of solving the subproblems independently

Partial-order Planning

- Progression and regression planning are ***totally ordered plan*** *search* forms
  - Must decide on complete action sequence on all subproblems
  - Operates on “sequences”, in order

$\Rightarrow$ Does not take advantage of problem decomposition

Search the Space of Partial Plans

- Start with partial plan
  Expand plan until producing complete plan
- Refinement operators: add constraints to partial plan

Eg: Adding an action

  Imposing order on actions

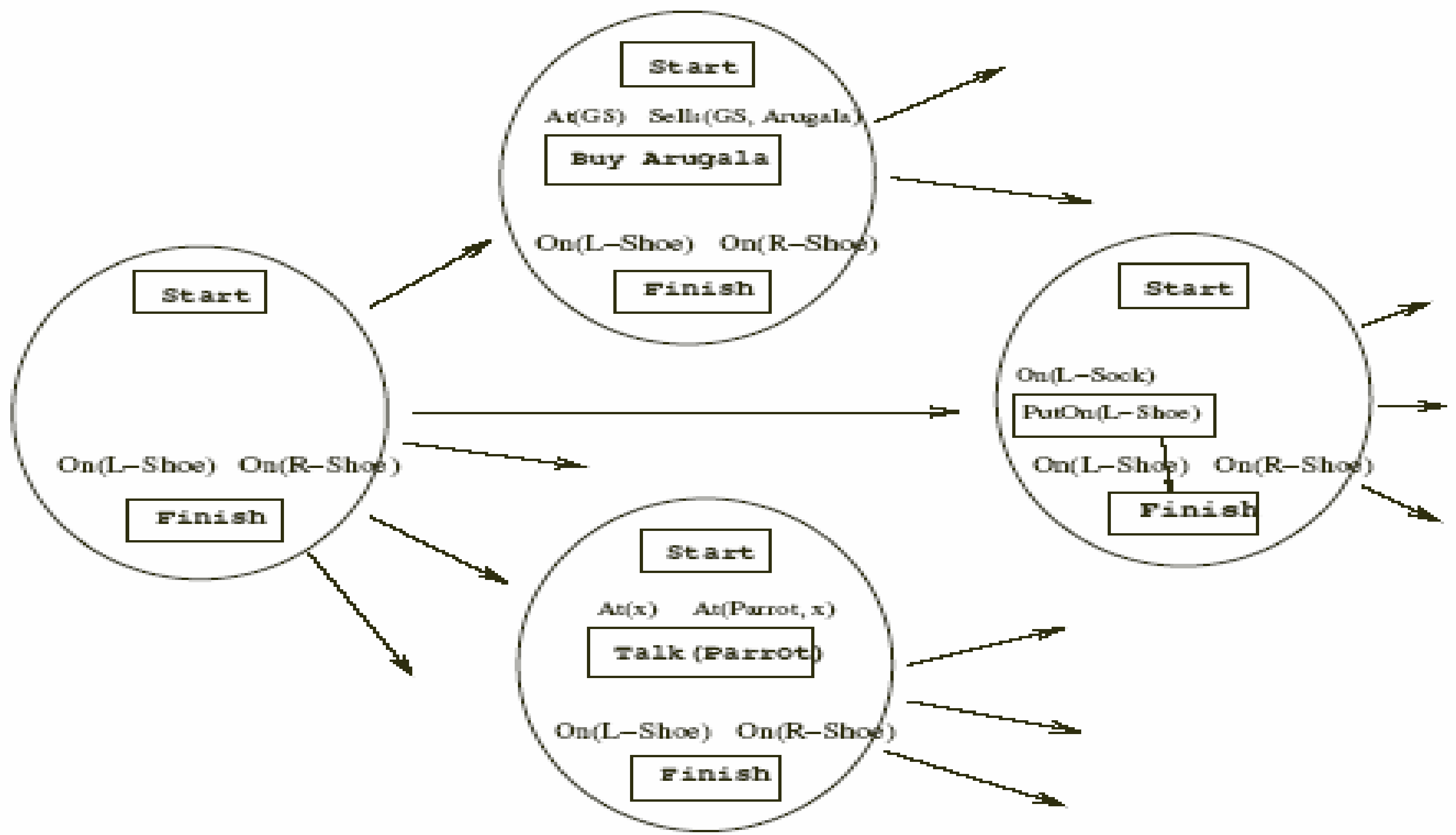
  Instantiating unbound variable

  ...

  (View "partial plan" as set of "completed" plans...
  Each refinement REMOVES some plans.)

- \+ Modification Operators
  other changes – "debugging" bad plans

Searching in Space of "Partial Plans"



Shoe Example

Goal(RightShoeOn $\wedge$ LeftShoeOn)

(Init()

Action(RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)

Action(RightSock, PRECOND: EFFECT: RightSockOn)

Action(LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)

Action(LeftSock, PRECOND: EFFECT: LeftSockOn)

)

Planner: combine two action sequences

- ⟨ LeftSock, LeftShoe ⟩
- ⟨ RightSock, RightShoe ⟩

Initial Partial Plan (Shoes)

Consider: Goal: RShoeOn & LShoeOn

Initial: {}

Operators:

Op(RShoe, PreC: RSockOn, Eff: RShoeOn)

Op(LShoe, PreC: LSockOn, Eff: LShoeOn)

Op(RSock, PreC: fg, Eff: RSockOn)

Op(LSock, PreC: fg, Eff: LSockOn)

- Initially… just dummy actions:

  S_s (Start): no PreC; Effects are FACTs

  S_f (Finish): PreC = Goal; no Effects

Plan(

- Actions: { S_s: Act(Start; PreC: {}; E: {})
  S_f : Act(Finish; PreC: RShoeOn & LShoeOn) }
- Orderings: { $S_s \prec S_f$ }
- CausalLinks: {}
- Open-PreC: { RShoeOn, LShoeOn }

)

Shoe Plan #2

Plan

- *Actions:*
  - S_s: Act(Start; PreC: {}; Eff: {})
  - S_f: Act(Finish; PreC: RShoeOn $\wedge$ LShoeOn)
- *Orderings:* { $S_s \prec S_f$ }
- *CausalLinks:* {}
- *Open-PreC:* { RShoeOn, LShoeOn }

- "*Open-PreC*" $\neq$ {}

  $\Rightarrow$ NOT DONE!

- Next partial plan:

  Try to achieve "RShoeOn" $\in$ *Open-PreC*

  . . . using "RShoe"

Plan

- *Actions:*
  - S_s: Act(Start; PreC: {}; Eff: {})
  - S_f: Act(Finish; PreC: RShoeOn $\wedge$ LShoeOn)
  - S_{rs}: Act(RShoe; PreC: RSockOn; Eff: RShoeOn)
- *Orderings:* { $S_s \prec S_f$, $S_s \prec S_{rs} \prec S_f$ }
- *CausalLinks:* { RSock $\xrightarrow{\text{On(RSockOn)}}$ RShoe }
- *Open-PreC:* { RSockOn, LShoeOn }

Comments on Partial Plans

- Every action is between S_s and S_f

  S_s for start, before everything

  S_e for finish, only when all goals achieved

- $\prec$ means BEFORE

  not necessarily IMMEDIATELY before

  "$S_i \xrightarrow{PreC} S_j$" means before

  with no "intervening clobber-er"

- In "Planning Space":

  Move from $\boxed{\text{Plan}_i}$ to $\boxed{\text{Plan}_j}$ by

  ⋆ Adding new Action S

  ⋆ Adding new Ordering $\prec$; CausalLink $S_i \xrightarrow{PreC} S_j$,

  Value for Variable, ...

Q: When to add S?
A: If S :*Effect* matches *Open-PreC*

Q: How to add S?
A: Add S to *Actions*
Add $S \prec T$ to *Ordering*
Add $S \xrightarrow{PreC} T$ to *CausalLinks*
Add S:*PreC* to Open-PreC
as necessary
+ more. . .

Shoe Plan #3

Plan(Actions:

$$\left\{ \begin{array}{lll} S_s: & \text{Act(Start;} & \text{PreC: } \{\}; \text{ Eff: } \{\}) \\ S_e: & \text{Act(Finish;} & \text{PreC: RShoeOn} \wedge \text{LShoeOn}) \\ S_{rs}: & \text{Act(RShoe;} & \text{PreC: RSockOn; Eff: RShoeOn}) \\ S_{rx}: & \text{Act(RSock;} & \text{PreC: } \{\}; \text{ Eff: RSockOn}) \\ S_{lx}: & \text{Act(LSock;} & \text{PreC: } \{\}; \text{ Eff: LSockOn}) \\ S_{ls}: & \text{Act(LShoe;} & \text{PreC: LSockOn; Eff: LShoeOn}) \end{array} \right\}$$

Orderings: $\left\{ \begin{array}{l} S_s \prec S_e,\ S_s \prec S_{lx},\ S_s \prec S_{rx}, \ldots \\ S_{lx} \prec S_{ls},\ S_{rx} \prec S_{rs} \\ S_{ls} \prec S_e,\ S_{rs} \prec S_e,\ \ldots \end{array} \right\}$

CausalLinks: $\left\{ \begin{array}{l} S_{lx} \xrightarrow{LSockOn} S_{ls} \\ S_{rx} \xrightarrow{RSockOn} S_{rs} \end{array} \right\}$

Open-PreC: {}

)

Notes: As "Open-PreC = {}, can stop
(Still another check) . . .

Partial Plans

Plan(Actions:

$$\left\{\begin{array}{ll} S_s: & \text{Act(Start; PreC: \{\}; Eff: \{\})} \\ S_e: & \text{Act(Finish; PreC: RShoeOn} \land \text{LShoeOn)} \\ S_{rs}: & \text{Act(RShoe; PreC: RSockOn; Eff: RShoeOn)} \\ S_{rx}: & \text{Act(RSock; PreC: \{\}; Eff: RSockOn)} \\ S_{lx}: & \text{Act(LSock; PreC: \{\}; Eff: LSockOn)} \\ S_{ls}: & \text{Act(LShoe; PreC: LSockOn; Eff: LShoeOn)} \end{array}\right\}$$

Orderings: $\left\{\begin{array}{l} S_s \prec S_e,\ S_s \prec S_{lx},\ S_s \prec S_{rx}, \ldots \\ S_{lx} \prec S_{ls},\ S_{rx} \prec S_{rs} \\ S_{ls} \prec S_e,\ S_{rs} \prec S_e,\ \ldots \end{array}\right\}$

CausalLinks: $\left\{\begin{array}{l} S_{rx} \xrightarrow{RSockOn} S_{rs} \\ S_{lx} \xrightarrow{LSockOn} S_{ls} \end{array}\right\}$

Open-PreC: {})

- ≈ ⟨RSock, RShoe⟩ and ⟨LSock, LShoe⟩

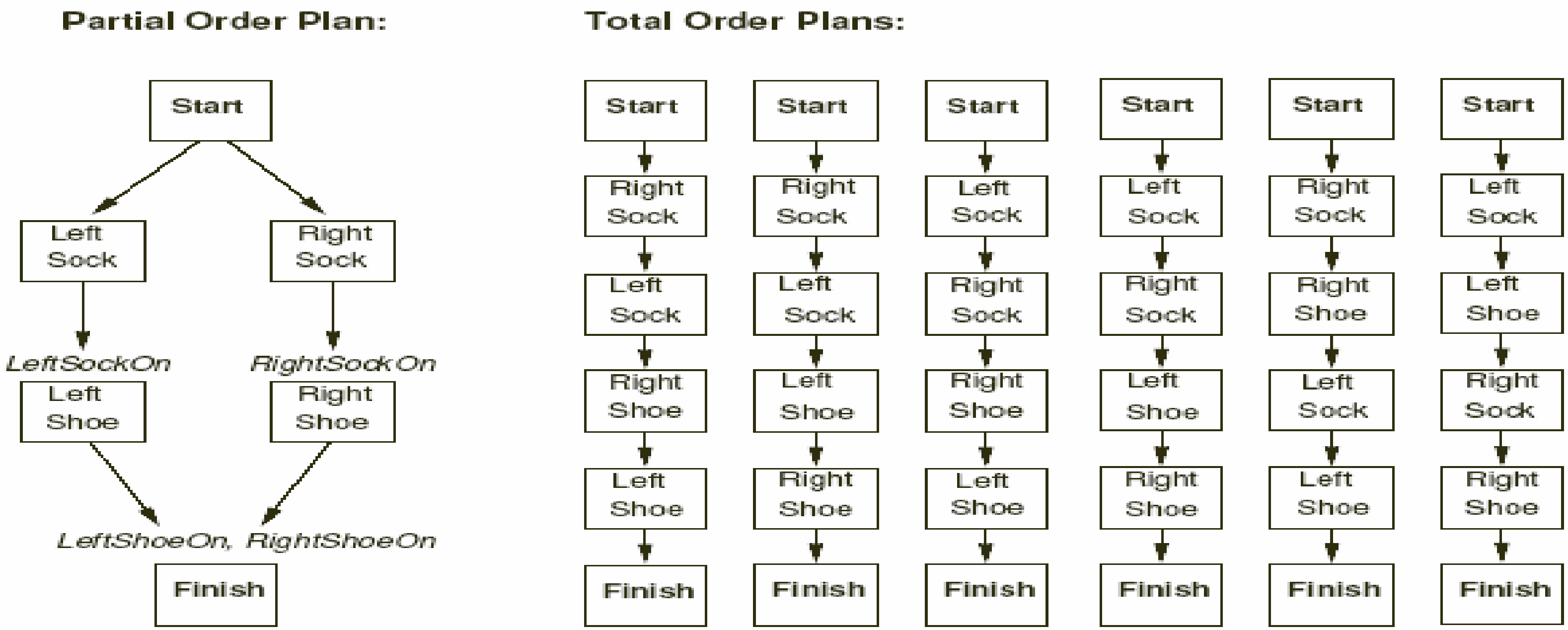
Q: Should they be combined, to produce LINEAR plan??

A: Why?

If left PARTIALLY specified, more options later
. . . when we have more constraints!

- Principle of least commitment:
  - Don't make decisions until necessary.
  - Only order actions that HAVE to be ordered
  - Only instantiate variables when needed
    (Don't decide on store until have all constraints)

Partial- vs Total- Order Plan



- LeftSock before LeftShoe
  RightSock before RightShoe

  But nothing else specified!

Constraints on PO-Plans

- "Partial order plan"
  - ⋆ *Some* constraints on order of actions
  - ⋆ Must be *consistent*

    NOT: $S_a \prec S_b$ and $S_b \prec S_a$

- A *linearization* of (partial) plan

  completely orders *all* actions

  ... producing a "totally ordered plan"

- "Causal Link" $A \xrightarrow{\rho} B$

  ... A achieves ρ for B

  - ⋆ connects action A to action B

    where $A{:}Effect = \rho = B{:}PreCond$

  If $C{:}Effect = \neg\rho$,

  must not add C between A and B

Solution

A solution ≡ a (partial) plan that
agent can execute and
guarantees achievement of goal(s).

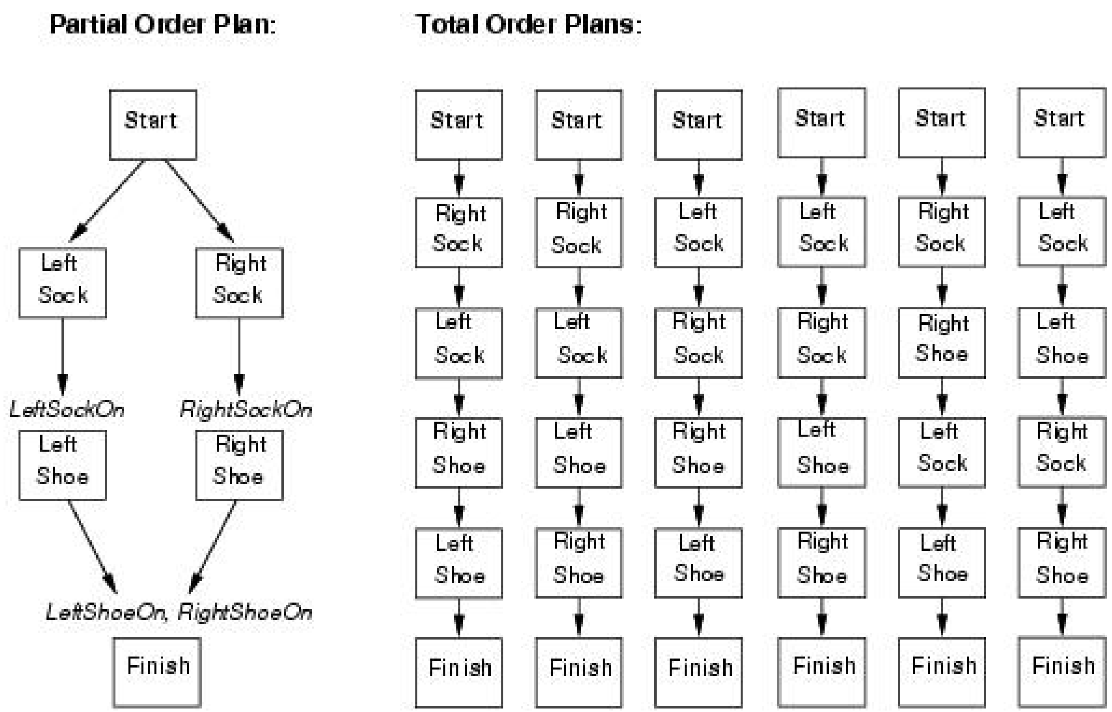
≡ a complete, consistent plan:

- **Complete**: Open-PreC= {}
  Each precond ρ of each action A is achieved by some other action B s.t.
  $B \prec A$ and
  $\neg\exists C$ s.t. C undoes ρ and $B \prec C \prec A$
  $\forall\ A \in \text{Actions(Plan)},\ \forall\ \rho \in \text{PreC}(A);$
  $\exists B\quad B \prec A \quad \&\quad \rho \in \text{Eff}(A)$
  $\& \neg\exists C\ B \prec C \prec A\ \ \&\ \ \neg\ \rho \in \text{Eff}(C)$
- **Consistent**: No contradictions in ordering constraints.
- Note: Need not be a TOTAL plan.
  … but every linearization is correct!

Partial-order Planning

A **Partial-order planner** is a planning algorithm

- that can place two actions into a plan
- without specifying which comes first



Recent Progress

- SAT-plan
  - Convert Planning Task to SAT problem; Send to SAT solver
  - WORKS very well!
- GraphPlan
  - Create graph structure of states+actions
  - Find traversal, until levels out...
  - It works too!
- More expressive descriptions, ...
  - Action Description language (ADL)
- Re-planning
- Not “open loop”, but reactive
- Stochastic outcomes...$\Rightarrow$ Markov Decision Process
  - ... Reinforcement Learning

Comparison of Strips vs ADL

Strips language	ADL language
Positive literals in states Poor ∧ Unknown	Positive and negative literals in states: ¬Rich ∧ ¬Famous
Closed World Assumption: Unmentioned literals are false	Open World Assumption: Unmentioned literals are unknown
Effect $P \vee \neg Q$ means add P and delete Q	Effect $P \vee \neg Q$ means add "P and $\neg Q$" and delete "$\neg P$ and Q"
Only ground literals in goals: Rich ∧ Famous	Quantified variables in goals: $\exists x\ At(P_1, x) \wedge At(P_2, x)$ is goal of having P_1 and P_2 in same place
Goals are conjunctions Rich ∧ Famous	Goal can include conjunctions and disjunctions ¬Poor ∧ (Famous ∨ Smart)
Effects are conjunctions	Conditional effects allowed: "when P : E" means E is an effect only if P is satisfied
No support for equality	Equality predicate $(x = y)$ is built in
No support for types	Variables can have types, as in $(p : Plane)$

Summary

- Representations in planning
- Representation of action:
  preconditions + effects
- Forward planning
- Backward chaining
- Partial-order planning

Limits of Strips-Based Planners

- Hierarchical plans

  “Prepare booster, prepare capsule, load cargo, launch"
  then achieve each sub-part, recursively . . .

- Complex conditions

  Strips: Simple Proposition literals
  Better: “Launch causes ALL items to go into space"
  “If . . .THEN . . . "

- Time

  Strips: discrete, sequential,. . .
  Better: deadlines, actions have durations, time windows,. . .

- Resources

  Global constraints on TOTAL resources allowed
  . . . of allowed at instant,. . .

POPlaning Example: Changing a Tire

Flat-Tire Domain

Fl= Flat; Sp= Spare; Ax= Axel; Tr= Trunk; Gr= Ground

- Initial: At(Fl, Ax) ∧ At(Sp, Tr)

$$Op\left(\begin{array}{ll} \text{Start} & \\ \text{PreC:} & \{\} \\ \text{Eff:} & At(Fl, Ax) \wedge At(Sp, Tr) \end{array}\right)$$

- Goal: At(Sp, Ax)

$$Op\left(\begin{array}{ll} \text{Finish} & \\ \text{PreC:} & At(Sp, Ax) \\ \text{Eff:} & \{\} \end{array}\right)$$

- Actions:

$$Op\left(\begin{array}{ll} \text{TakeOutSpare} & \\ \text{PreC:} & At(Sp, Tr) \\ \text{Eff:} & \neg At(Sp, Tr) \wedge At(Sp, Gr) \end{array}\right)$$

$$Op\left(\begin{array}{ll} \text{RemoveFlat} & \\ \text{PreC:} & At(Fl, Ax) \\ \text{Eff:} & \neg At(Fl, Ax) \wedge At(Fl, Gr) \end{array}\right)$$

$$Op\left(\begin{array}{ll} \text{PutOnSpare} & \\ \text{PreC:} & At(Sp, Gr) \wedge \neg At(Fl, Ax) \\ \text{Eff:} & \neg At(Sp, Gr) \wedge At(Sp, Ax) \end{array}\right)$$

$$Op\left(\begin{array}{ll} \text{LeaveOverNight} & \\ \text{PreC:} & \{\} \\ \text{Eff:} & \neg At(Sp, Gr) \wedge \neg At(Sp, Ax) \wedge \neg At(Sp, Tr) \\ & \wedge \neg At(Fl, Gr) \wedge \neg At(Fl, Ax) \end{array}\right)$$

Tire – Planning #1

- Initial configuration:

$$Plan\left(\begin{array}{ll} Acts: & \left\{\begin{array}{l} S_s\text{: Act(Start; PreC: \{\}; Eff: At(Sp,Tr)} \wedge \text{At(Fl,Ax) } \\ S_f\text{: Act(Finish; PreC: At(Sp,Ax); Eff: \{\})} \end{array}\right\} \\ Orderings: & \{ S_s \prec S_f \} \\ CausalLinks: & \{\} \\ \text{Open-PreC:} & \{ \texttt{At(Sp,Ax)} \} \end{array}\right)$$

Start — At(Sp,Tr), At(Fl,Ax)

At(Sp,Ax) — Finish

- Only "*Open-PreC*": `At(Sp,Ax)`

  Given

$$\texttt{Op}\left(\begin{array}{ll} \texttt{PutOnSpare} & \\ \texttt{PreC:} & \texttt{At(Sp,Gr)} \wedge \neg\texttt{At(Fl,Ax)} \\ \texttt{Eff:} & \neg\texttt{At(Sp,Gr)} \wedge \boxed{\texttt{At(Sp,Ax)}} \end{array}\right)$$

- New (partial) plan . . .

$$Plan\left(\begin{array}{ll} Acts: & \left\{\begin{array}{l} S_s\text{: Act(Start; PreC: \{\}; Eff: At(Sp,Tr)} \wedge \text{At(Fl,Ax)} \\ S_f\text{: Act(Finish; PreC: At(Sp,Ax); Eff: \{\})} \\ S_1\text{: Act(PutOnSpare; PreC: At(Sp,Gr)} \wedge \neg\text{At(Fl,Ax)} \\ \qquad\qquad\text{Eff: } \neg\text{At(Sp,Gr)} \wedge \text{At(Sp,Ax))} \end{array}\right\} \\ Orderings: & \{ S_s \prec S_1 \prec S_f \} \\ CausalLinks: & \{ S_1 \xrightarrow{At(Sp,Ax)} S_f \} \\ \text{Open-PreC:} & \left\{\begin{array}{l} \texttt{At(Sp,Gr)} \\ \neg\texttt{At(Fl,Ax)} \end{array}\right\} \end{array}\right)$$

Tire – Planning #2

Plan:

Acts:
- S_s: Act(Start; PreC: {}; Eff: At(Sp,Tr) ∧ At(Fl,Ax)
- S_f: Act(Finish; PreC: At(Sp,Ax); Eff: {})
- S_1: Act(PutOnSpare; PreC: At(Sp,Gr) ∧ ¬At(Fl,Ax)
  Eff: ¬At(Sp,Gr) ∧ At(Sp,Ax))

Orderings: $\{ S_s \prec S_1 \prec S_f \}$

CausalLinks: $S_1 \xrightarrow{At(Sp,Ax)} S_f$

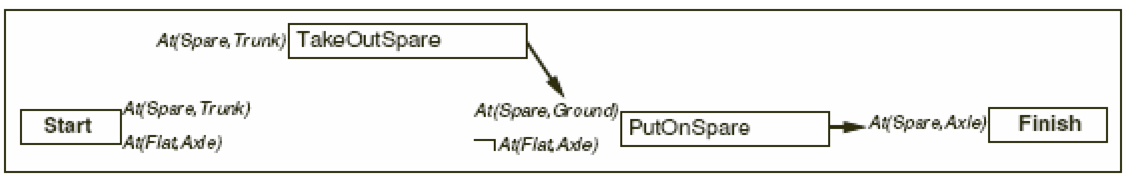
Open-PreC: { At(Sp,Gr), ¬At(Fl,Ax) }

- Now work on *Open-PreC*: At(Sp,Gr)

  (PutOnSpare's pre-cond)

  Only action achieving this condition:

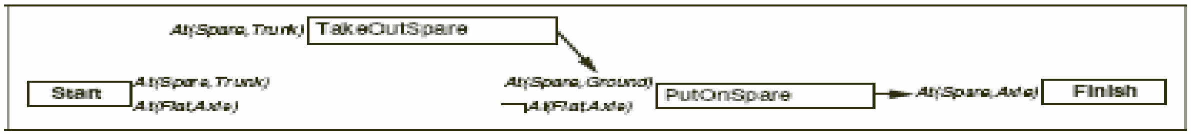
  TakeOutSpare



- Now: *Open-PreC* = { At(Sp,Tr), ¬At(Fl,Ax) }

  To process ¬At(Fl,Ax) . . .

“Clobbering”



- For ¬At(Fl,Ax):
  Spse LeaveOvernight

$$\text{Op}\left(\begin{array}{ll} \text{LeaveOverNight} & \\ \text{PreC:} & \{\} \\ \text{Eff:} & \neg\text{At(Sp, Gr)} \wedge \neg\text{At(Sp, Ax)} \wedge \neg\text{At(Sp, Tr)} \\ & \wedge \neg\text{At(Fl, Gr)} \wedge \neg\text{At(Fl, Ax)} \wedge \end{array}\right)$$

- BUT...
  LeaveOvernight:Effects includes ¬At(Sp,Gr),
  which clobbers

$$\text{TakeOutSpare} \xrightarrow{\text{At(Sp,Gr)}} \text{PutOn(Sp,Ax)}$$

- Ie, sequence

$$\langle \text{TakeOutSpare}, \boxed{\text{LeaveOvernight}}, \text{PutOn(Sp,Ax)} \rangle$$

  will NOT work:
  when about to perform PutOn(Sp,Ax)
  its precondition At(Sp,Gr) is NOT true!

Protected Links

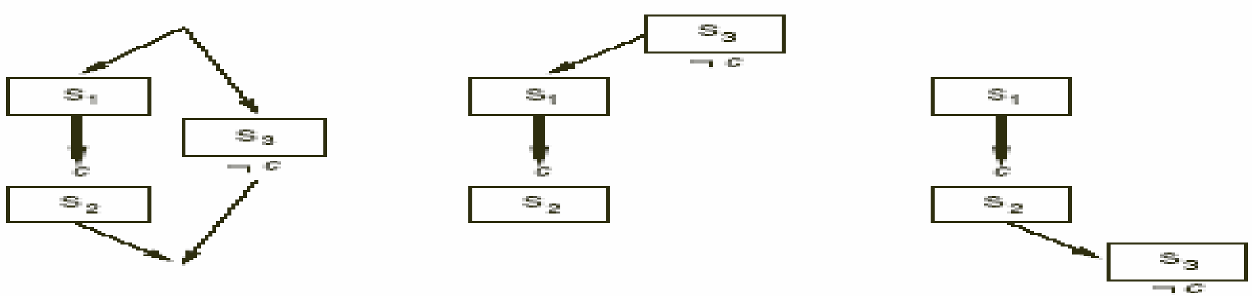
Problem: A step S_3 threatens causal link $S_1 \xrightarrow{\rho} S_2$ if effect of S_3 is deleting (clobbering) ρ

Eg: LeaveOvernight threatens TakeOutSpare $\xrightarrow{At(Sp,Gr)}$ PutOnSpare

Solution: Add *ordering constraints* to keep S_3 from intervening between S_1 and S_2

Option1: *Demotion* (before S_1): (b)

Option2: *Promotion* (after S_2): (c)



(a) (b) (c)

Where to add *LeaveOvernight*?

⇒ need to move LeaveOvernight to

1. before TakeOutSpare
2. after PutOnSpare

2. does NOT work:
as LeaveOvernight:Effects includes ¬At(Sp,Ax), which clobbers

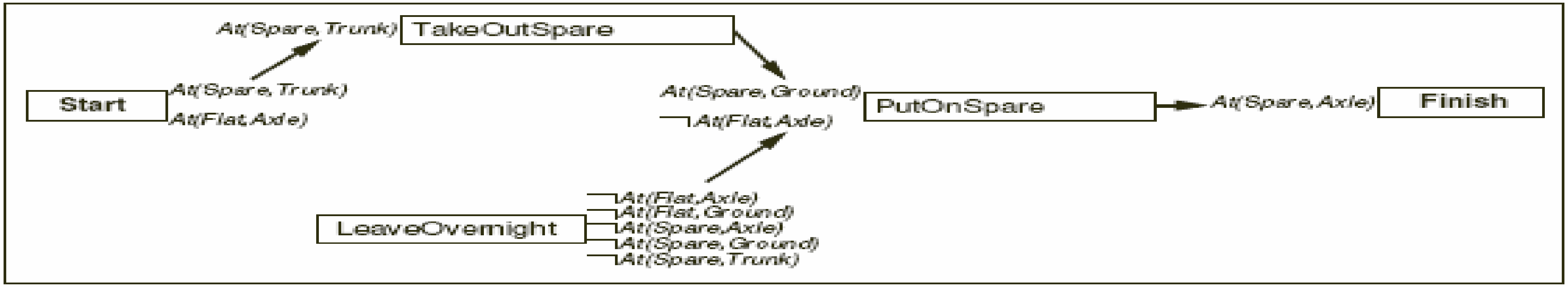
$$\text{PutOn(Sp,Ax)} \xrightarrow{\text{At(Sp,Ax)}} \text{Finish}$$

⇒ AFTER Finish

...but NOTHING can be AFTER Finish...

⇒ need to use 1.

see dotted-line (for $\prec$)



Problem… backtrack …

- Here, *Open-PreC* = { At(Sp,Tr) }
  Start is only action w/Effect = At(Sp,Tr)

$\Rightarrow$ need Start $\xrightarrow{\text{At(Sp,Tr)}}$ TakeOutSpare,

But... LeaveOvernight threatens

Start $\xrightarrow{\text{At(Sp,Tr)}}$ TakeOutSpare,

Promote or Demote?

1. Move LeaveOvernight BEFORE Start ?
   Not allowed!
2. Move LeaveOvernight AFTER TakeOutSpare?
   No – as LeaveOvernight $\prec$ TakeOutSpare

Neither is possible!

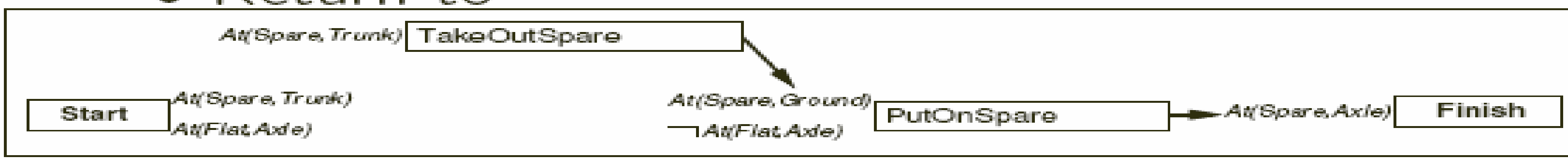
- Planner has proven

LeaveOvernight is NOT (this) part of changing tire!

$\Rightarrow$ Need to backtrack!

Tire – Planning #3

- Return to



How else to achieve ¬At(Fl,Ax) ?

- Try

$$\text{Op}\left(\begin{array}{ll} \text{RemoveFlat} & \\ \text{PreC:} & At(Fl,\ Ax) \\ \text{Eff:} & \neg At(Fl,\ Ax) \wedge At(Fl,\ Gr) \end{array}\right)$$

- *Open-PreC* = { At(Fl,Ax), At(Sp,Tr) }

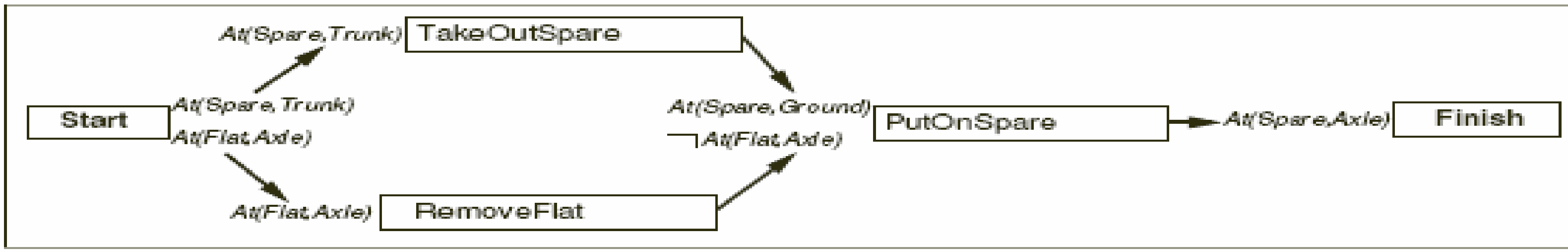
  Use

  Start $\xrightarrow{At(Fl,Ax)}$ RemoveFlat

  Start $\xrightarrow{At(Sp,Tr)}$ TakeOutSpare

- *Open-PreC* = {} ... DONE!

Comments



- Partial Order: 2 linearizations

  (in general, can be MANY extensions)

  Added FLEXIBITY

  if events later impose other constraints

- Further improvements

  Might unlink Start $\xrightarrow{\texttt{At(Fl,Ax)}}$ RemoveFlat,
  then re-link it

  Should use Dependency-Directed backtracking!

- Other complications if FirstOrder

  and have unbound variables

- Some good heuristics

  ⋆ most-constrained variables (CSP)

POP: Partial Order Planner

```
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
----------------------------------------------------------------
function SELECT-SUBGOAL(plan) returns S_need, c

   pick a plan step S_need from STEPS(plan)
        with a precondition c that has not been achieved
   return S_need, c
----------------------------------------------------------------
procedure CHOOSE-OPERATOR(plan, operators, S_need, c)

   choose a step S_add from operators or STEPS(plan) that has c as an effect
   if there is no such step then fail
   add the causal link S_add --c--> S_need to LINKS(plan)
   add the ordering constraint S_add ≺ S_need to ORDERINGS(plan)
   if S_add is a newly added step from operators then
        add S_add to STEPS(plan)
        add Start ≺ S_add ≺ Finish to ORDERINGS(plan)
----------------------------------------------------------------
procedure RESOLVE-THREATS(plan)

   for each S_threat that threatens a link S_i --c--> S_j in LINKS(plan) do
        choose either
             Promotion: Add S_threat ≺ S_i to ORDERINGS(plan)
             Demotion: Add S_j ≺ S_threat to ORDERINGS(plan)
        if not CONSISTENT(plan) then fail
   end
```

Comments on POP

- Starts from
- S_{need} is step (in current partial plan) with an unsatisfied precondition, c

  Try to achieve c from
  - an existing step, or
  - some operator
- Link S_{need} to that step.
- Resolve any new threats

- POP is REGRESSION planner
- Sound and Complete!

Dealing with Variables

- So far, everything PROPOSITIONAL
  In general: Variables!

- $$\text{Op}\left(\begin{array}{lllll} \texttt{Move(b,x,y)} & & & & \\ \texttt{PreC:} & \texttt{On(b,x)} & \wedge\ \texttt{Clear(b)} & \wedge\ \texttt{Clear(y)} & \\ \texttt{Eff:} & \texttt{On(b,y)} & \wedge\ \texttt{Clear(x)} & \wedge\ \neg\texttt{On(b,x)} & \wedge\ \neg\texttt{Clear(y)} \end{array}\right)$$

  SubGoal: On(A,B)

- Use

  $$\text{Op}\left(\begin{array}{lllll} \texttt{Move(A,x,B)} & & & & \\ \texttt{PreC:} & \texttt{On(A,x)} & \wedge\ \texttt{Clear(A)} & \wedge\ \texttt{Clear(B)} & \\ \texttt{Eff:} & \texttt{On(A,B)} & \wedge\ \texttt{Clear(x)} & \wedge\ \neg\texttt{On(A,x)} & \wedge\ \neg\texttt{Clear(B)} \end{array}\right)$$

  ...move A from *somewhere* to B
  "least committment principle"

- If initially On(A,D): could be "Move(A, $\boxed{D}$,B)"
  Or if reach On(A,Q), then "Move(A, $\boxed{Q}$,B)"
  ...

Q: Is Move(A,x,B) $\xrightarrow{\texttt{On(A,B)}}$ Finish
threatened by
M_2 where M_2:Effect = On(Q,z) ?

A: Only if $z = A$ or $z = B$

So need to record $z \neq A$, $z \neq B$

RealWorld Planning

- Optimum-AIV
  - used by European Space Agency
  - assembly, integration, verification of space-craft

- Generate plans, monitor their execution + replan as required

---

- DS1: NASA probe

- CelCorp ...

- Job-Shop scheduling

- Scheduling for space mission
  Hubble telescope
  ...

Comparison

- Operation Research tools
  (eg, PERT chart, Critical Path method)

Input: Hand-constructed complete partial-order plan

Output: Find optimal schedule

action = object that
takes time,
have ordering constraints
. . . effects ignored

Note: Ordering constraints hard-coded
no additional knowledge engineering

- Don't have info needed to replan
  or to handle many different planning task

- Here, need general information . . .

  + sophisticated planner . . .

- Most time spent finding what the constraints really are