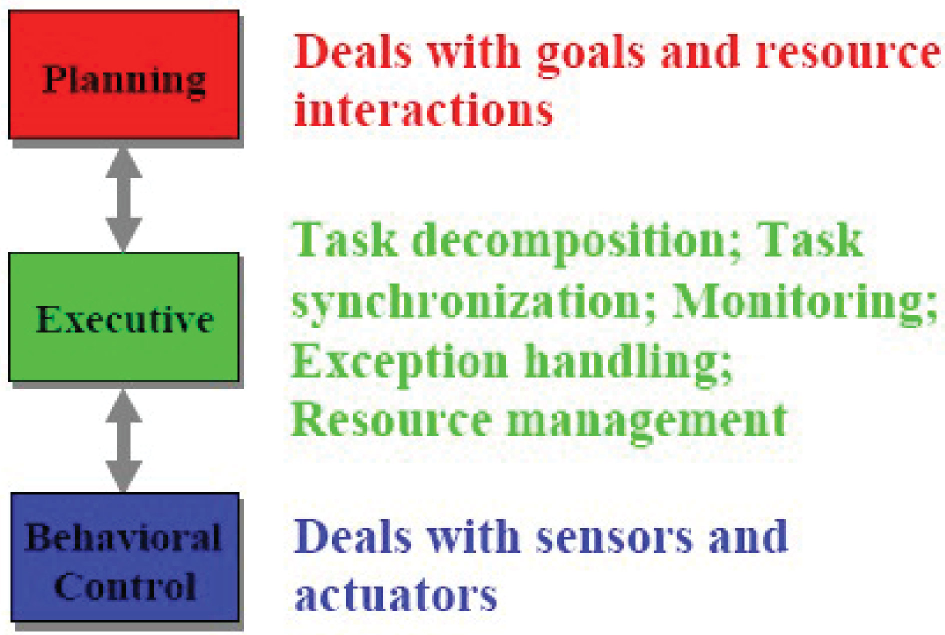
Task Planning

Architetture Robotiche

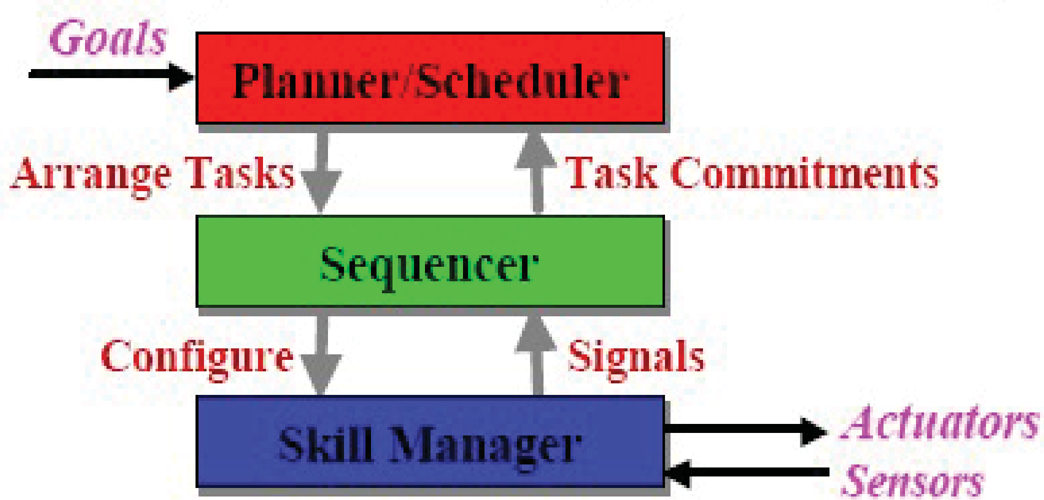
Three Layered Architecture

- Deliberative:
  planning, reasoning, decision
- Executive:
  Execution monitoring, task decomposition, resource management, Command sequencing, failure detection, diagnosis and repair, reconfigure/replan/adjust
- Functional:
  Sensorimotor processes, mapping, localization, avoidance, path/trajectory planning, etc.



Three Layered Architecture

- Explicit Separation of Planning, Sequencing, and Control
  - Upper layers provide *control flow* for lower layers
  - Lower layers provide ***status*** (state change) and ***synchronization*** (success/failure) for upper layers
- Heterogeneous Architecture
  - Each layer utilizes algorithms tuned for its particular role
  - Each layer has a representation to support its reasoning

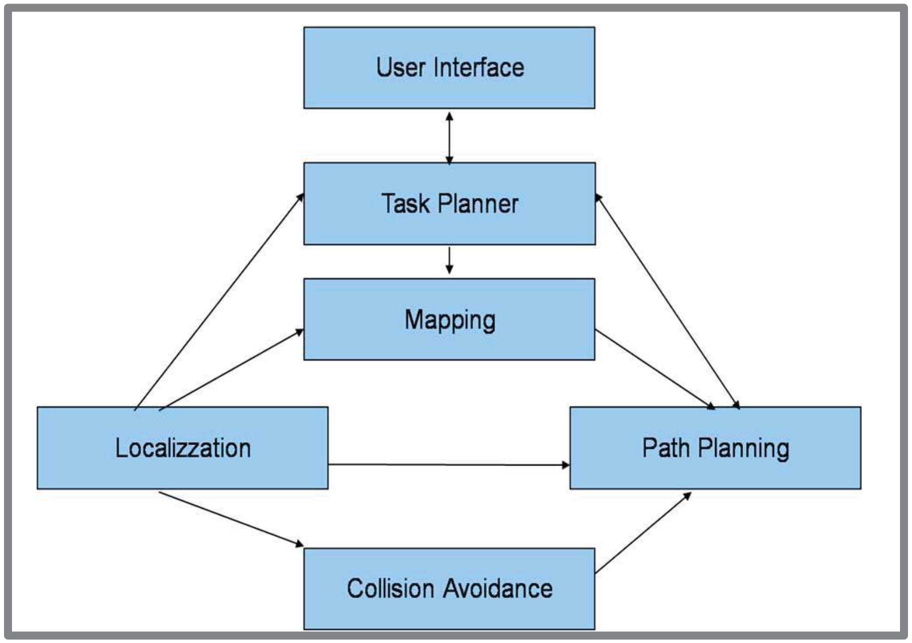


Three Layered Architecture

Architettura di RHINO la guida robotica del museo di Bonn (1995); simile MINERVA (1998) ad Atlanta

Architettura a 3 Livelli per un robot mobile:

1. Funzionale:
   Mapping, Localizzazione, Avoidance
2. Esecutivo:
   Sequencer, monitor
3. Deliberativo:
   Task Planner



Architetture di RIHINO

Rhino, 1997

Minerva, 1998

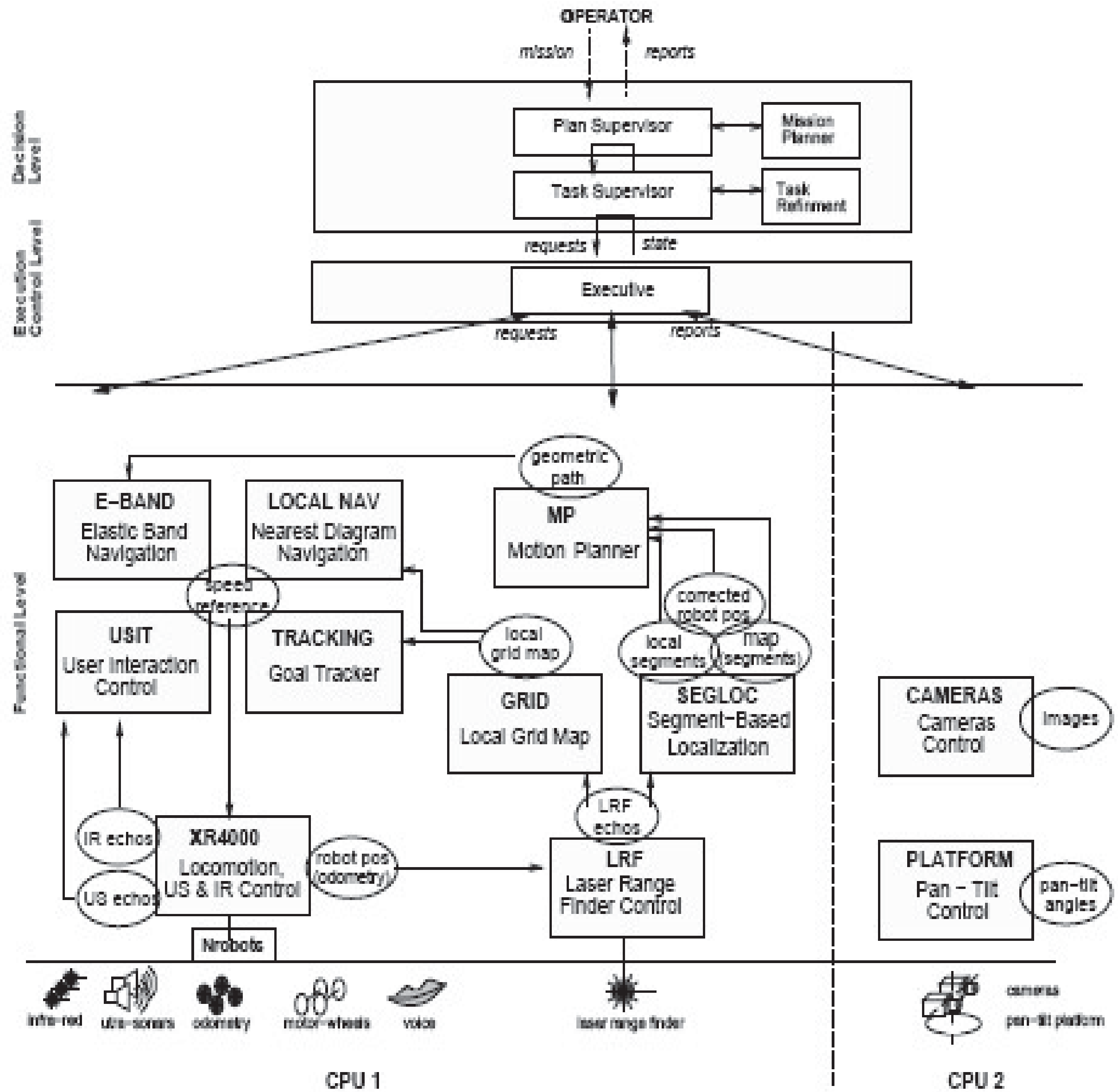
Three Layered Architecture

- LAAS architecture:

Tre Livelli:

1. Deliberativo
   (temporal planner)
2. Esecutivo
   (PRS)
3. Funzionale
   (GENOME)

Controllo di Rover



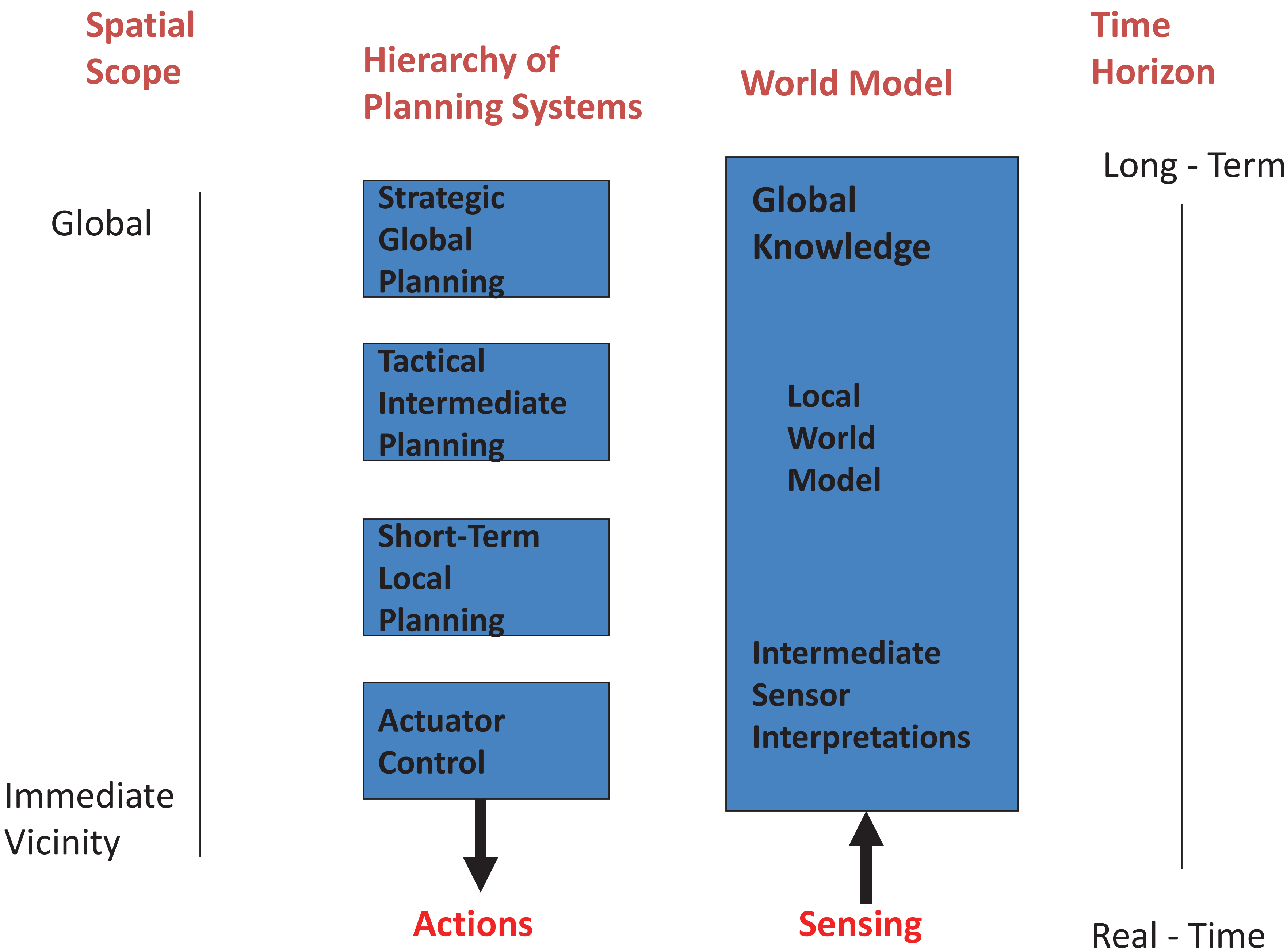
Three Layered Architecture

Xavier Architecture (1995)

Task Planning (Prodigy)
Path Planning (Decision-Theoretic)
Map-Based Navigation (POMDPs)
Local Obstacle Avoidance (Curvature Velocity Method)
Servo-Control (Commercial)

Planner Hierarchy

- Hierarchical planning systems typically share a structured and clearly identifiable subdivision of functionality regarding **distinct program modules** that communicate with each other in a **predictable and predetermined manner**.
- At a hierarchical planner's highest level, the most global and least specific plan is formulated (deliberative planner).
- At the lowest levels, **rapid real-time** response is required, but **the planner is concerned only** with its immediate surroundings and has lost the sight of the big picture.


Spatial Scope
Hierarchy of Planning Systems
World Model
Time Horizon
Global
Immediate Vicinity
Strategic Global Planning
Tactical Intermediate Planning
Short-Term Local Planning
Actuator Control
Global Knowledge
Local World Model
Intermediate Sensor Interpretations
Long - Term
Real - Time
Actions
Sensing

Hierarchical Planners vs. BBS

Hierarchical Planners

- Rely heavily on world models,
- Can readily integrate world knowledge,
- Have a broad perspective and scope.

BB Control Systems

- afford modular development,
- Real-time robust performance within a changing world,
- Incremental growth
- are tightly coupled with arriving sensory data.

Hybrid Control

- **The basic idea is simple**: we want the best of both worlds (if possible).
- The goal is to **combine** closed-loop and open-loop **execution.**
- That means to **combine reactive and deliberative control.**
- This implies combining the **different time-scales and representations.**
- This mix is called hybrid control.

Hybrid robotic architectures believe that a union of deliberative and behavior-based approaches can potentially yield the best of both worlds.

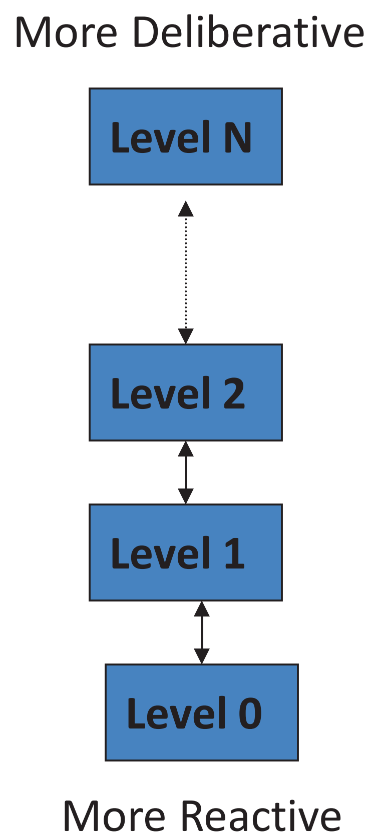
Organizing Hybrid Systems

Planning and reaction can be tied:

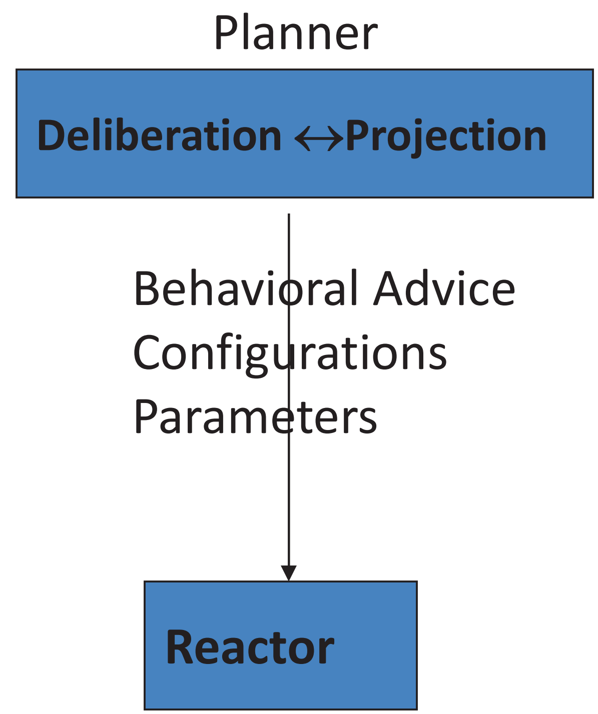
A: hierarchical integration - planning and reaction are involved with different activities, time scales

B: Planning to guide reaction - configure and set parameters for the reactive control system.

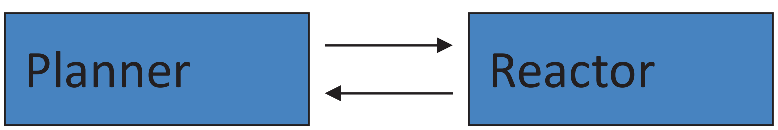
C: coupled - concurrent activities



A



B



C

Organizing Hybrid Systems

It was observed that the emerging architectural design of choice is:

– multi-layered hybrid comprising of

* a **top-down** planning system and
* a **lower-level** reactive system.

– **the interface** (middle layer between the two components) **design** is a central issue in differentiating different hybrid architectures.

In summary, a modern hybrid system typically consists of three components:

♦ **a reactive layer**

♦ **a planner**

♦ **a layer that puts the two together.**

=> Hybrid architectures are often called **three-layer architectures**.

The Magic Middle: Executive Control

- The middle layer has a hard job:
  1) compensate for the limitations of both the planner and the reactive system
  2) reconcile their different time-scales.
  3) deal with their different representations.
  4) reconcile any contradictory **commands** between the two.
- This is **the challenge** of hybrid systems

  => **achieving the right compromise between the two ends.**

Executive Control

Reusing Plans

- **Some frequently useful planned decisions may need to be reused**, so to avoid planning, an intermediate layer **may cache** and look those up. These can be:
  - – **intermediate-level actions** (ILAs): stored in contingency tables.
  - – **macro operators**: plans compiled into more general operators for future use.

Dynamic Re-planning

- Reaction **can influence** planning.
- Any **"important" changes** discovered by the low-level controller are passed back to the planner **in a way that** the planner can use to re-plan.
- The planner is interrupted when **even a partial answer** is needed in real-time.
- The reactive controller (and thus the robot) **is stopped** if it must wait for the planner to tell it *where to go*.

Executive Control

Planner - Driven Reaction

- Planning can also influence reaction.
- Any **"important" optimizations** the planner discovers are passed down to the reactive controller.
- The planner's suggestions **are used if they are** possible and safe.

=> Who has priority, planner or reactor? It depends, as we will see...

Types of "Reaction ↔ Planning" Interaction

- ♦ **Selection**: Planning is viewed as configuration.
- ♦ **Advising:** Planning is viewed as advice giving.
- ♦ **Adaptation**: Planning is viewed as adaptation of controller.
- ♦ **Postponing**: Planning is viewed as a least commitment process.

Universal Plans

- Suppose for a given problem, **all possible plans** **are generated** **for all possible situations in advance** and stored.
- If for each situation a robot has a pre-existing optimal plan, **it can react optimally**, be reactive and optimal.
- It has a universal plan (These are complete reactive mappings).

Viability of Universal Plans

- A system with a universal plan **is reactive**; the planning is done at compile-time, **not at run-time**.
- Universal plans are **not viable in most domains**, because:
  - the **world** must be deterministic.
  - the **world** must not change.
  - the **goals** must not change.
  - the **world** is too complex (state space is too large).

Classical Planning Problem

Newell and Simon 1956

- Given the *actions* available in a task domain.
- Given a problem specified as:
  - an initial *state* of the world,
  - a set of *goals* to be achieved.
- Find a *solution* to the problem, i.e., a *way* to transform the initial state into a new state of the world where the goal statement is true.

Action Model, State, Goals

Classical Planning

- Action Model: complete, deterministic, correct, rich representation
- State: single initial state, fully known
- Goals: complete satisfaction

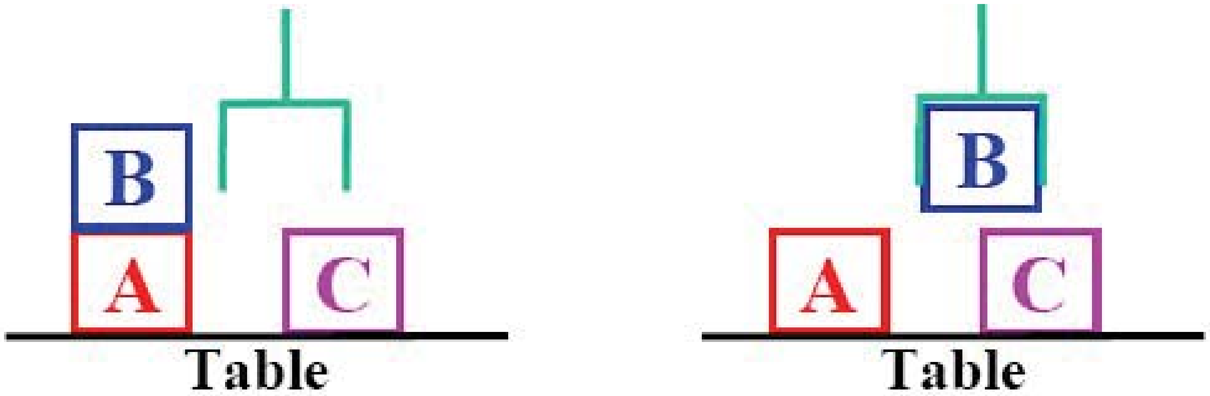
Several different planning algorithms

Classical Planning

- States, Actions, Goal
  - Actions induce transitions form state to state
  - Goal are termination states

- Representation:
  - Implicit representation of the states (predicates)
  - Planning Domain to represent the actions as modifications of states (symbolic transitions)

Example: Blocks World



- Blocks are picked up and put down by the arm
- Blocks can be picked up only if they are clear, i.e., without any block on top
- The arm can pick up a block only if the arm is empty, i.e., if it is not holding another block, i.e., the arm can be pick up only one block at a time
- The arm can put down blocks on blocks or on the table

Planning Domain

- Frame Problem
  - How to represent unchanged facts?
  - Example: I go from home (state S) to the store (state S'). In S': The house is still there, Rome is still the largest city in Italy, my shoes are the same, etc..
  - Path Planning has not this issue (sub-symbolic representation)

- Ramification Problem:
  - How to represent indirect effect of the actions
  - I go from home (state S) to the store (state S'). In S':
    The number of people in the store went up by 1,
    The contents of my pockets are now in the store, etc..

STRIPS Domain

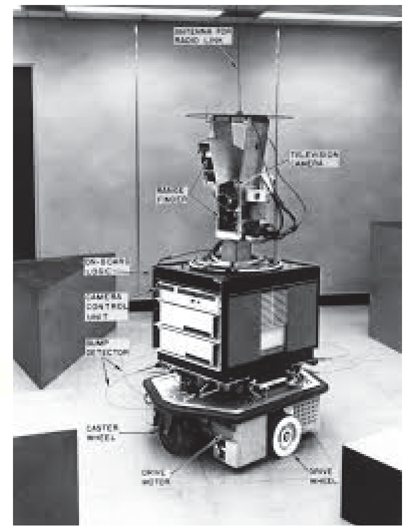
STanford Research Institute Problem Solver [Fikes, Nilsson, 1971]

Pickup_from_table(b)
- Pre: Block(b), Handempty
  Clear(b), On(b, Table)
- Add: Holding(b)
- Delete: Handempty,
  On(b, Table)

Pickup_from_block(b, c)
- Pre: Block(b), Handempty
  Clear(b), On(b, c), Block(c)
- Add: Holding(b), Clear(c)
- Delete: Handempty,
  On(b, c)

Putdown_on_table(b)
- Pre: Block(b), Holding(b)
- Add: Handempty,
  On(b, Table)
- Delete: Holding(b)

Putdown_on_block(b, c)
- Pre: Block(b), Holding(b)
  Block(c), Clear(c), $b \neq c$
- Add: Handempty, On(b, c)
- Delete: Holding(b), Clear(c)



Init: On(a,Table), On(b,table), On(c,table)

Goal: On(a,table),On(b,a), On(c,b)

STRIPS-like Domain

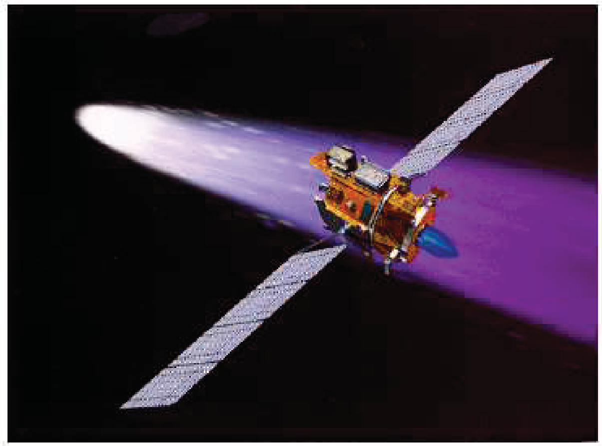
Observation-1
- target → pointing
- instruments → calibrated

Observation-2

Observation-3

Observation-4

...



TakeImage (?target, ?instr):
  Pre: Status(?instr, Calibrated), Pointing(?target)
  Eff: Image(?target)

Calibrate (?instrument):
  Pre: Status(?instr, On), Calibration-Target(?target), Pointing(?target)
  Eff: ¬Status(?inst, On), Status(?instr, Calibrated)

Turn (?target):
  Pre: Pointing(?direction), ?direction ≠ ?target
  Eff: ¬Pointing(?direction), Pointing(?target)

STRIPS Domain

States:

- set of well-formed formulas (wffs: conjunction of literals)

Set of Actions, each represented with

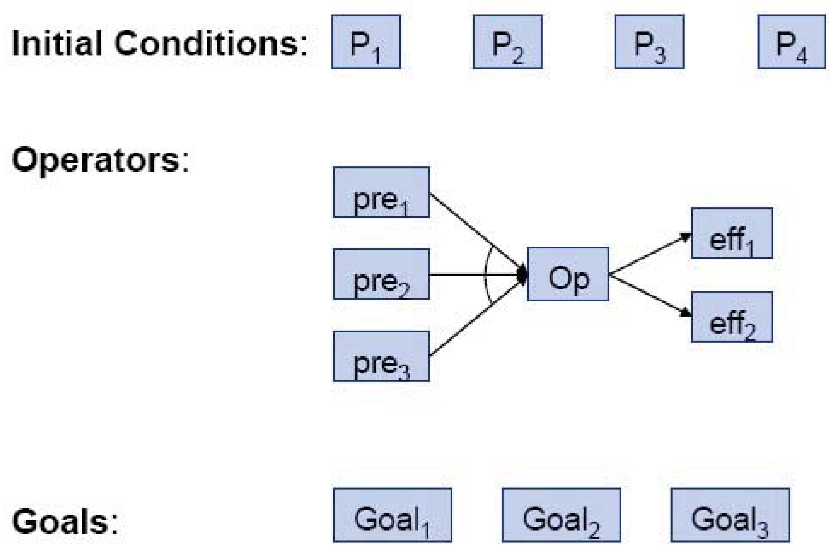
– Preconditions (list of predicates that should hold)
– Delete list (list of predicates that will become invalid)
– Add list (list of predicates that will become valid) Actions thus allow variables

A goal condition:

- well-formed formula

Planning Problem

- **Planning Domain:**
  - Operators as preconditions and effects
- **Planning Problem:**
  - Initial State, Planning Domain, Goals



PDDL Domain

Planning Domain Definition Language
(standard language for classical AI planning)

Components of a PDDL planning task:

- Objects: Things of interest
- Predicates: Relevant properties of objects (can be true or false)
- Initial state: The initial state of the world
- Goal specification: Desiderata
- Actions/Operators: Means to change the state of the world

Planning Domain: predicates and actions.
Planning Problem: initial state and goal specification.

PDDL Domain

Planning Domain Definition Language
(standard language for classical AI planning)

Planning Domain:

(define (domain <domain name>)
<PDDL code for predicates>
<PDDL code for first action>
[...]
<PDDL code for last action>
)

Planning Problem

(define (problem <problem name>)
(:domain <domain name>)
<PDDL code for objects>
<PDDL code for initial state>
<PDDL code for goal specification>
)

(:objects rooma roomb ball1 ball2 ball3 ball4 left right)

(:predicates (ROOM ?x) (BALL ?x) (GRIPPER ?x) (at-robby ?x) (at-ball ?x ?y) (free ?x) (carry ?x ?y))

(:init (ROOM rooma) (ROOM roomb) (BALL ball1) (BALL ball2) (BALL ball3) (BALL ball4) (GRIPPER left) (GRIPPER right) (free left) (free right) (at-robby rooma) (at-ball ball1 rooma) (at-ball ball2 rooma) (at-ball ball3 rooma) (at-ball ball4 rooma))

(:goal (and (at-ball ball1 roomb) (at-ball ball2 roomb) (at-ball ball3 roomb) (at-ball ball4 roomb)))

PDDL Domain

Planning Domain Definition Language
(standard language for classical AI planning)

Planning Domain:

(define (domain <domain name>)
<PDDL code for predicates>
<PDDL code for first action>
[...]
<PDDL code for last action>
)

Planning Problem

(define (problem <problem name>)
(:domain <domain name>)
<PDDL code for objects>
<PDDL code for initial state>
<PDDL code for goal specification>
)

(:action move :parameters (?x ?y)
:precondition (and (ROOM ?x) (ROOM ?y) (at-robby ?x)) :effect (and (at-robby ?y) (not (at-robby ?x))))

(:action pick-up :parameters (?x ?y ?z)
:precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z) (at-ball ?x ?y) (at-robby ?y) (free ?z)) :effect (and (carry ?z ?x) (not (at-ball ?x ?y)) (not (free ?z))))

AI Planning Paradigms

- Classical Planning
- Temporal Planning
- Conditional Planning
- Decision Theoretic Planning
- …
- Least-Commitment Planning
- HTN planning
- …

AI Planning Paradigms

Classical planning

(STRIPS, operator-based, first-principles)

"generative"

Hierarchical Task Network planning

"practical" planning

MDP & POMDP planning

planning under uncertainty

Planning Algorithms

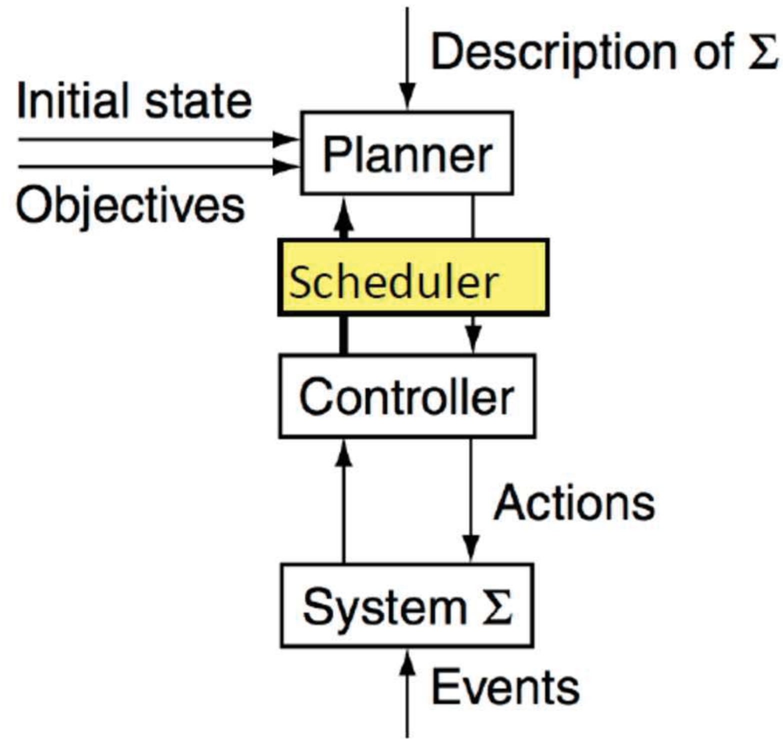
- Soundness
  - A planning algorithm is sound if all solutions found are legal plans
    - All preconditions and goals are satisfied
    - No constraints are violated
- Completeness
  - A planning algorithm is complete if a solution can be found whenever one exists
  - A planning algorithm is strictly complete if all solutions are included in the search space
- Optimality
  - A planning algorithm is optimal if the order in which solutions are provided is consistent with some measure of plan quality

Three Main Types of Planners

1. Domain-specific
   - Made or tuned for a specific planning domain
   - Won't work well (if at all) in other planning domains
2. Domain-independent
   - In principle, works in any planning domain
   - In practice, need restrictions on what kind of planning domain
3. Configurable
   - Domain-independent planning engine
   - Input includes info about how to solve problems in some domain

Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Planning Versus Scheduling



- Scheduling
  - Decide when and how to perform a given set of actions
    - Time constraints
    - Resource constraints
    - Objective functions
  - Typically NP-complete
- Planning
  - Decide what actions to use to achieve some set of objectives
  - Can be much worse than NP-complete; worst case is undecidable
- Scheduling problems may require replanning

Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Restrictive Assumptions

A0: Finite system:

- finitely many states, actions, events

A1: Fully observable:

- the controller always Σ's current state

A2: Deterministic:

- each action has only one outcome

A3: Static (no exogenous events):

- no changes but the controller's actions

A4: Attainment goals:

- a set of goal states S_g

A5: Sequential plans:

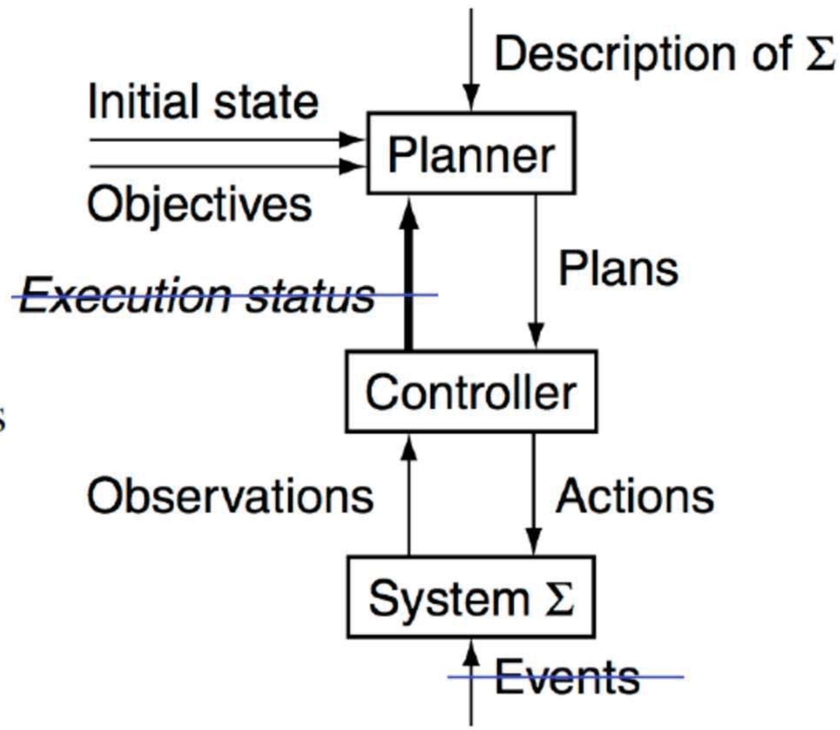
- a plan is a linearly ordered sequence of actions $(a_1, a_2, \ldots a_n)$

A6: Implicit time:

- no time durations; linear sequence of instantaneous states

A7: Off-line planning:

- planner doesn't know the execution status



Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

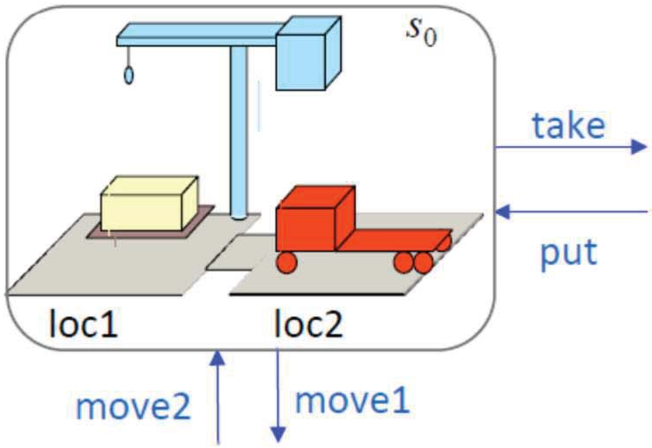
Classical Planning (Chapters 2–9)

- Classical planning requires all eight restrictive assumptions
  - Offline generation of action sequences for a deterministic, static, finite system, with complete knowledge, attainment goals, and implicit time
- Reduces to the following problem:
  - Given a planning problem $\mathcal{P} = (\Sigma, s_0, S_g)$
  - Find a sequence of actions $(a_1, a_2, \ldots a_n)$ that produces a sequence of state transitions $(s_1, s_2, \ldots, s_n)$ such that s_n is in S_g.
- This is just path-searching in a graph
  - Nodes = states
  - Edges = actions
- Is this trivial?

Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Classical Planning (Chapters 2–9)

- Generalize the earlier example:
  - 5 locations,
  - 3 robot vehicles,
  - 100 containers,
  - 3 pallets to stack containers on
  - Then there are 10^{277} states



- Number of particles in the universe is only about 10^{87}
  - The example is more than 10^{190} times as large
- Automated-planning research has been heavily dominated by classical planning
  - Dozens (hundreds?) of different algorithms

Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Linear Planning

- A linear planner is a classical planner such that:
  - no importance distinction of goals
  - all (sub)goals are assumed to be independent
  - (sub)goals can be achieved in arbitrary order

- Plans that achieve subgoals are combined by placing *all steps* of one subplan *before or after all* steps of the others (=non-interleaved)

STRIPS Planning

- STRIPS (*initial-state, goals*)
  - *state* = *initial-state*; *plan* = []; *stack* = []
  - Push *goals* on *stack*
  - Repeat until *stack* is empty
    - If top of *stack* is **goal** that matches *state*, then pop *stack*
    - Else if top of stack is a **conjunctive goal** *g*, then
      - **Select** an ordering for the subgoals of *g*, and push them on *stack*
    - Else if top of *stack* is a **simple goal** *sg*, then
      - **Choose** an operator *o* whose add-list matches goal *sg*
      - Replace goal *sg* with operator *o*
      - Push the preconditions of *o* on the *stack*
    - Else if top of *stack* is an **operator** *o*, then
      - *state* = apply(*o*, *state*)
      - *plan* = [*plan*; *o*]

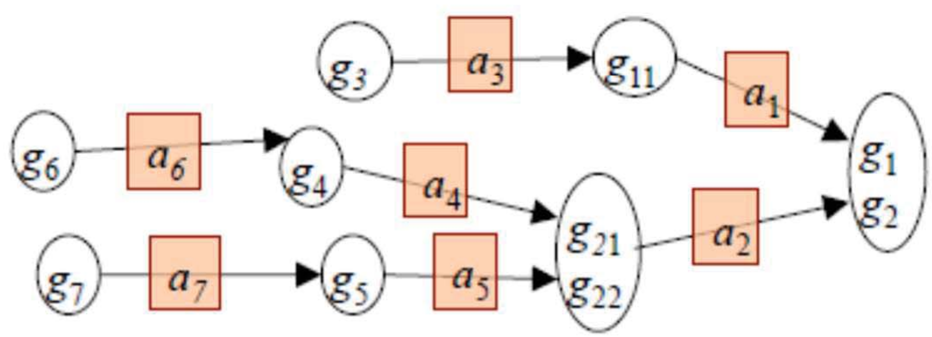
Simmons, Veloso : Fall 2001

STRIPS

- Basic idea: given a compound goal $g = \{g_1, g_1, \ldots\}$, try to solve each g_i separately
  - Works if the goals are *serializable* (can be solved in some linear order)

$\pi \leftarrow$ the empty plan
do a modified backward search from g:
instead of $\gamma^{-1}(s,a)$, each new set of subgoals is just precond(a)
whenever you find an action that's executable in the current state,
go forward on the current search path as far as possible, executing actions and appending them to π
repeat until all goals are satisfied

$\pi = \langle \pi_1, \pi_2 \rangle$ or $\langle \pi_2, \pi_1 \rangle$
$\pi_2 = \langle \pi_{11}, \pi_{12}, a_2 \rangle$ or $\langle \pi_{12}, \pi_{11}, a_2 \rangle$
$\pi_{21} = \langle a_7, a_4 \rangle$
$\pi_{22} = \langle a_7, a_5 \rangle$



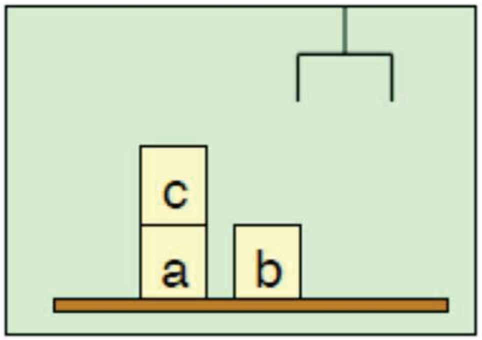
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Linear Planning

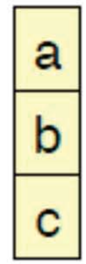
- Advantage:
  - Goals are solved one at a time (ok if independent)
  - Sound

- Disadvantage
  - Suboptimal solutions (number of operators in the plan)
  - incomplete

The Sussman Anomaly



Initial state



goal

- On this problem, STRIPS can't produce an irredundant solution
  - Try it and see

Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Non-Linear Planning

- Basic Idea
  - Goal set instead of goal stack
  - Search space all possible subgoal orderings
  - Goal interactions by interleaving
- Advantages
  - Sound, complete, can be optimal with respect to plan length (depending on search strategy employed)
- Disadvantages
  - Larger search space

Non-Linear Planning

NLP (*initial-state, goals*)

- *state* = *initial-state*; *plan* = []; *goalset* = *goals*; *opstack* = []
- Repeat until *goalset* is empty
  - **Choose** a goal *g* from the *goalset*
  - If *g* does not match *state*, then
    - **Choose** an operator *o* whose add-list matches goal *g*
    - Push *o* on the *opstack*
    - Add the preconditions of *o* to the *goalset*
  - While all preconditions of operator on top of *opstack* are met in *state*
    - Pop operator *o* from top of *opstack*
    - *state* = apply(*o*, *state*)
    - *plan* = [*plan*; *o*]

Simmons, Veloso : Fall 2001

Progressive Planning

Input : a world model and a goal
Output : a plan or fail.

(ignoring variables)

ProgPlan[DB,Goal] =

  If Goal is satisfied in DB, then return empty plan

  For each operator o such that precond(o) is satisfied in the current DB:

    Let DB′ = DB + addlist(o) – dellist(o)

    Let plan = ProgPlan[DB′,Goal]

    If plan $\neq$ fail, then return [act(o) ; plan]

  End for

  Return fail

Brachman & Levesque 2005

Regressive Planning

Input : a world model and a goal

Output : a plan or fail.

(ignoring variables)

RegrPlan[DB,Goal] =

  If Goal is satisfied in DB, then return empty plan

  For each operator o such that dellist(o) $\cap$ Goal = $\{\}$:

    Let Goal′ = Goal + precond(o) – addlist(o)

    Let plan = RegrPlan[DB,Goal′]

    If plan $\neq$ fail, then return [plan ; act(o)]

  End for

  Return fail

Brachman & Levesque 2005

Decidability of Planning

Halting problem

Allow function symbols?	Decidability of PLAN-EXISTENCE	Decidability of PLAN-LENGTH
no[α]	decidable	decidable
yes	semidecidable[β]	decidable

[α]This is ordinary classical planning.

[β]True even if we make several restrictions (see text).

Can cut off the search at every path of length *n*

Next: analyze complexity for the decidable cases

Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

- In this case, can write domain-specific algorithms
  - e.g., DWR and Blocks World: PLAN-EXISTENCE is in P and PLAN-LENGTH is NP-complete

Kind of representation	How the operators are given	Allow negative effects?	Allow negative preconditions?	Complexity of PLAN-EXISTENCE	Complexity of PLAN-LENGTH
classical rep.	in the input	yes	yes/no	EXPSPACE-complete	NEXPTIME-complete
		no	yes	NEXPTIME-complete	NEXPTIME-complete
			no	EXPTIME-complete	NEXPTIME-complete
			no$^{\alpha}$	PSPACE-complete	PSPACE-complete
	in advance	yes	yes/no	PSPACE $^{\gamma}$	PSPACE $^{\gamma}$
		no	yes	NP $^{\gamma}$	NP $^{\gamma}$
			no	P	NP $^{\gamma}$
			no$^{\alpha}$	NLOGSPACE	NP

$^{\alpha}$ no operator has >1 precondition

$^{\gamma}$ PSPACE-complete or NP-complete for some sets of operators

Heuristic Search (Chapter 9)

- Heuristic function like those in A*
  - Created using techniques similar to planning graphs
- Problem: A* quickly runs out of memory
  - So do a greedy search instead

- Greedy search can get trapped in local minima
  - Greedy search plus local search at local minima

- HSP [Bonet & Geffner]
- FastForward [Hoffmann]

Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Heuristics for Forward-Chaining Planning

Several classical planning style are available:
- http://icaps-conference.org/index.php/Main/Competitions

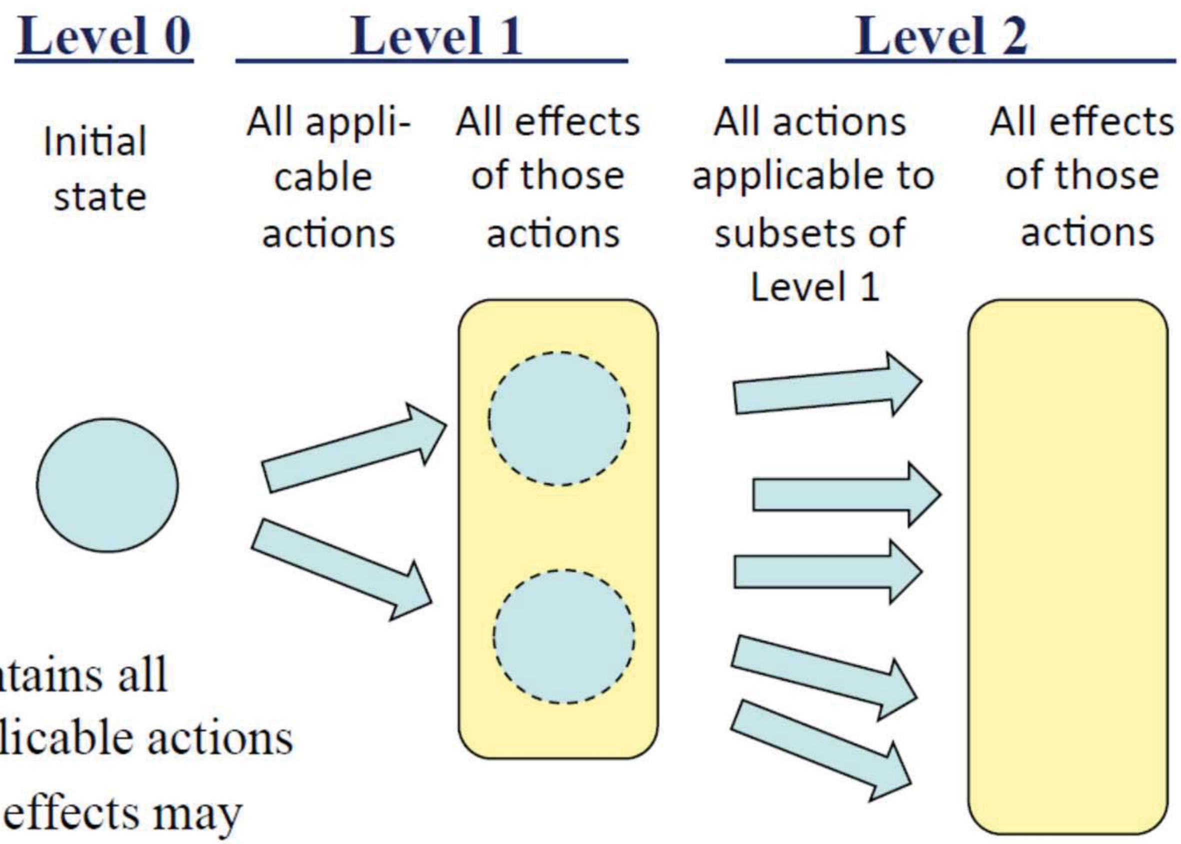
Forward-chaining planners:
- solving an abstraction of the original, hard, planning problem

The most widely used abstraction involves planning using `relaxed actions', where the delete effects of the original actions are ignored.

Examples:
FF [Hoffmann & Nebel 2001], HSP [Bonet & Geffner 2000], UnPOP [McDermott 1996] use relaxed actions as the basis for their heuristic estimates

FF was the first to count the number of relaxed actions in a relaxed plan connecting the goal to the initial state

Planning Graphs (Chapter 6)



- Rough idea:
  - First, solve a *relaxed problem*
    - Each "level" contains all effects of all applicable actions
    - Even though the effects may contradict each other
  - Next, do a state-space search *within the planning graph*
- Graphplan, IPP, CGP, DGP, LGP, PGP, SGP, TGP, ...

Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

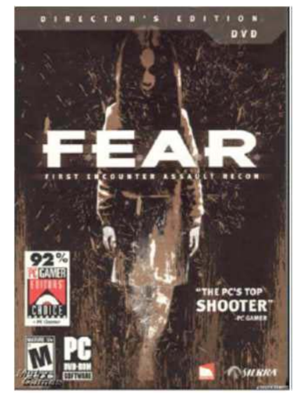
STRIPS and Games

Behavior of Non Player Characters (NPCs) can be described by abstract actions defined in a symbolic world model, e.g. First-Person Shooter (FPS) games

F.E.A.R. (short for First Encounter Assault Recon) is a horror-themed first-person shooter developed by Monolith Productions

– Gamespot's Best AI Award in 2005

– Ranked 2nd in the list of most influential AI games



The agents' behavior is a function of the generated plans based on goals, state, and available actions

Jeff Orkin: Three States and a Plan: The AI of F.E.A.R. *Proceedings of the Game Developer's Conference (GDC)*

Olivier Bartheye and Eric Jacopin: A PDDL-Based Planning Architecture to Support Arcade Game Playing

Summary

- If classical planning is extended to allow function symbols
  - Then we can encode arbitrary computations as planning problems
    - » Plan existence is semidecidable
    - » Plan length is decidable
- Ordinary classical planning is quite complex
    - » Plan existence is EXPSPACE-complete
    - » Plan length is NEXPTIME-complete
  - But those are *worst case* results
    - » If we can write domain-specific algorithms, most well-known planning problems are much easier

Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

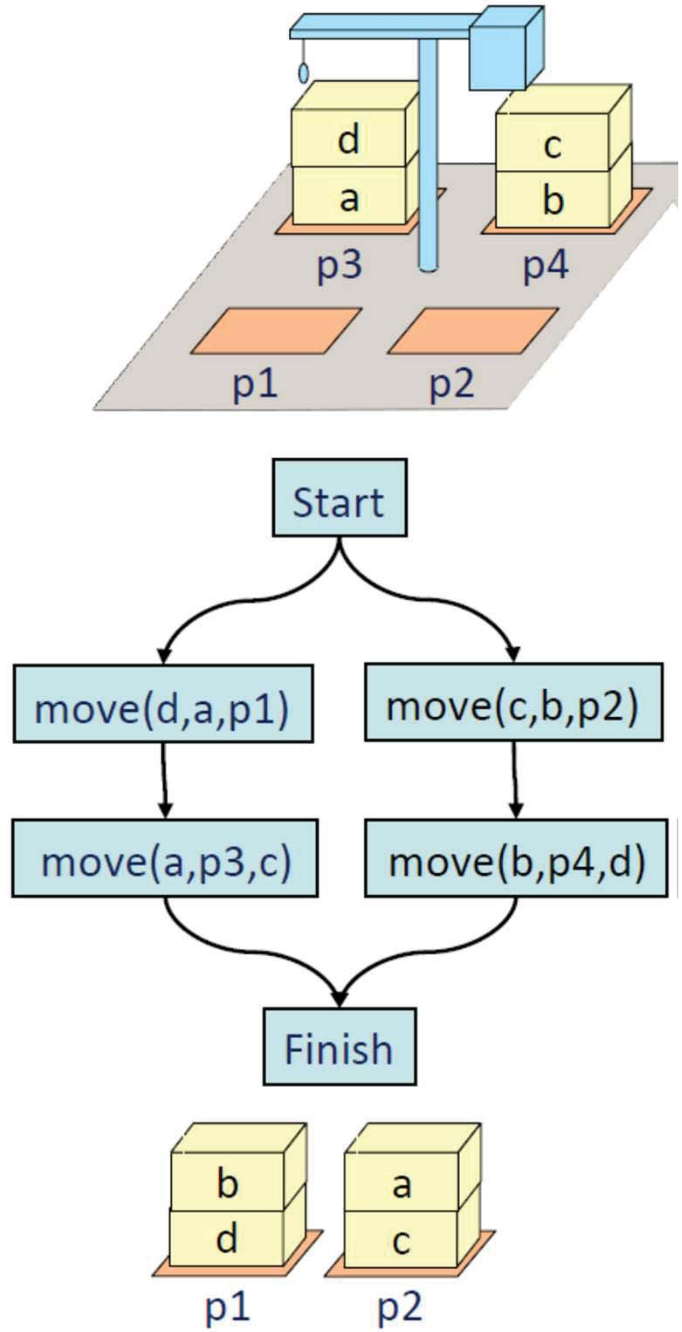
State Space vs. Plan Space

- Planning in the state space:
  - sequence of actions, from the initial state to the goal state
- Planning in the plan space:
  - Sequence of plan transformations, from an initial plan to the final one

Plan-Space Planning (Chapter 5)

- Decompose sets of goals into the individual goals
- Plan for them separately
  - Bookkeeping info to detect and resolve interactions
- Produce a partially ordered plan that retains as much flexibility as possible

- The Mars rovers used a temporal-planning extension of this



Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Plan-State Search

- Search space is set of *partial plans*
- Plan is tuple <*A*, *O*, *B*>
  - *A*: Set of ***actions***, of the form (a_i : Op_j)
  - *O*: Set of ***orderings***, of the form ($a_i < a_j$)
  - *B*: Set of ***bindings***, of the form ($v_i = C$), ($v_i \neq C$), ($v_i = v_j$) or ($v_i \neq v_j$)
- Initial plan:
  - <{*start*, *finish*}, {*start* < *finish*}, {}>
  - *start* has no preconditions; Its effects are the initial state
  - *finish* has no effects; Its preconditions are the goals

State-Space vs Plan-Space

Planning problem

Find a sequence of actions that make instance of the goal true

Nodes in search space

Standard search: node = concrete world state

Planning search: node = partial plan

(Partial) Plan consists of

- Set of operator applications S_i
- Partial (temporal) order constraints $S_i \prec S_j$
- Causal links $S_i \xrightarrow{c} S_j$

  Meaning: "S_i achieves $c \in precond(S_j)$" (record purpose of steps)

Search in the Plan-Space

Operators on partial plans

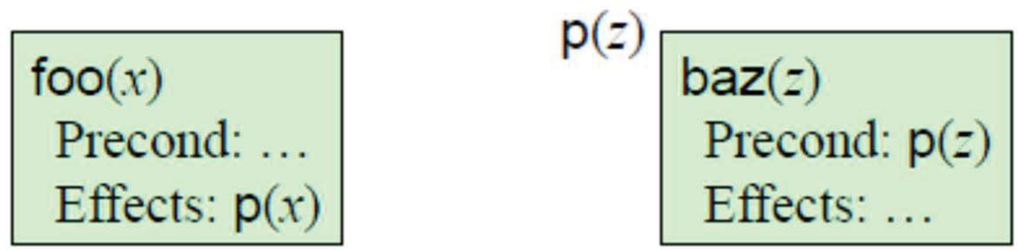
- add an action and a causal link to achieve an open condition
- add a causal link from an existing action to an open condition
- add an order constraint to order one step w.r.t. another

Open condition

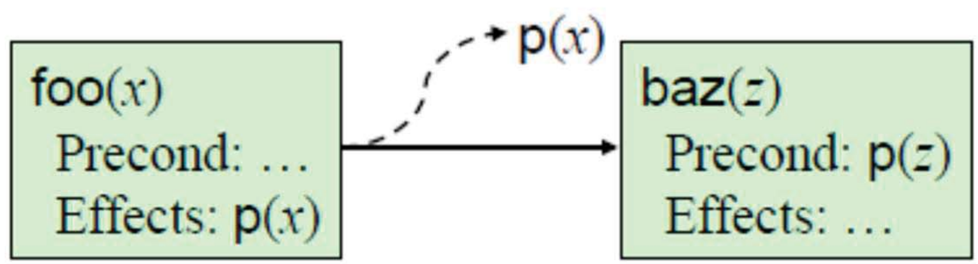
A precondition of an action not yet causally linked

Flaws: 1. Open Goals

- Open goal:
  - An action a has a precondition p that we haven't decided how to establish



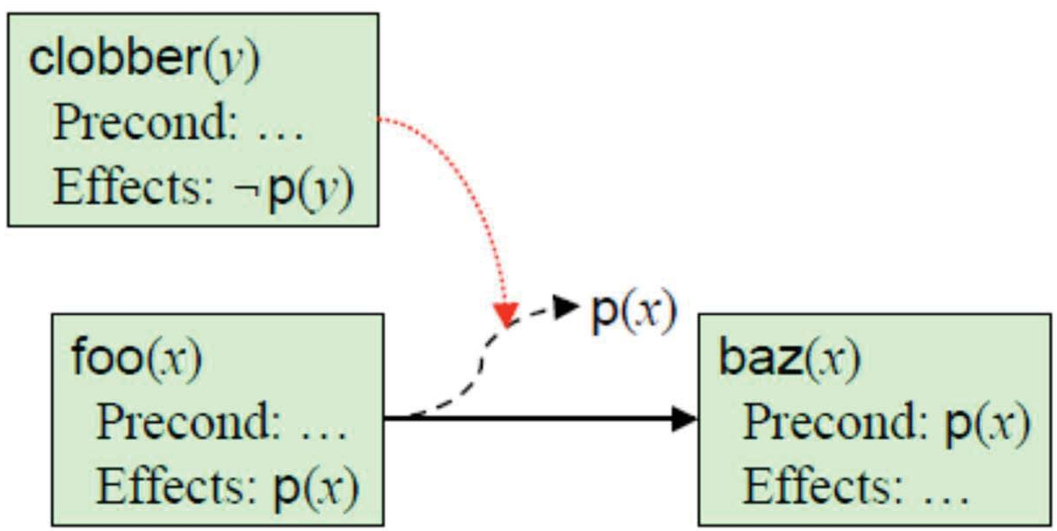
- Resolving the flaw:
  - Find an action b
    - (either already in the plan, or insert it)
  - that can be used to establish p
    - can precede a and produce p
  - Instantiate variables and/or constrain variable bindings
  - Create a causal link



Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Flaws: 2. Threats

- Threat: a deleted-condition interaction
  - Action a establishes a precondition (e.g., pq(x)) of action b
  - Another action c is capable of deleting p
- Resolving the flaw:
  - impose a constraint to prevent c from deleting p
- Three possibilities:
  - Make b precede c
  - Make c precede a
  - Constrain variable(s) to prevent c from deleting p



Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

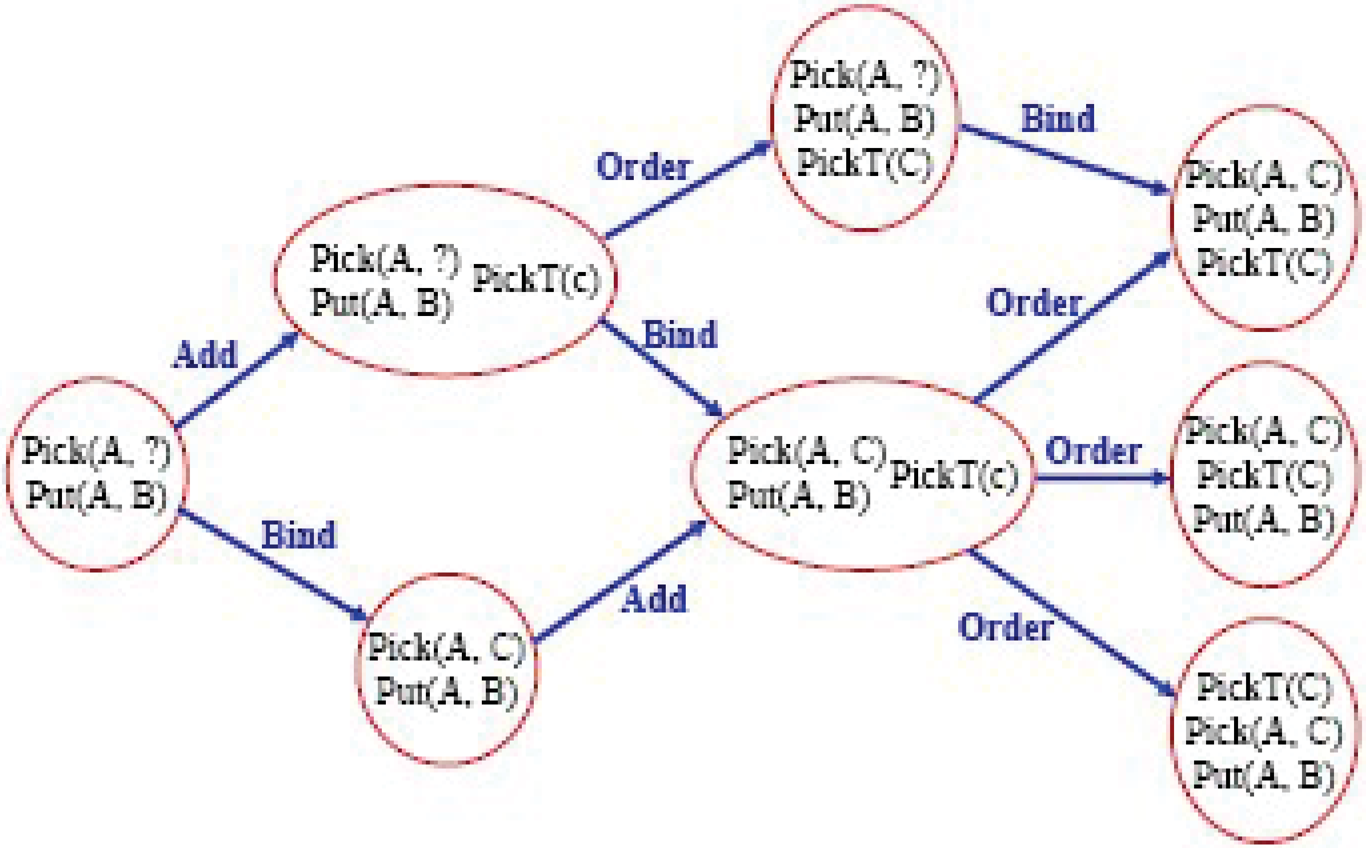
The PSP Procedure

```
PSP(π)
    flaws ← OpenGoals(π) ∪ Threats(π)
    if flaws = ∅ then return(π)
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ, π)
    if resolvers = ∅ then return(failure)
    nondeterministically choose a resolver ρ ∈ resolvers
    π′ ← Refine(ρ, π)
    return(PSP(π′))
end
```

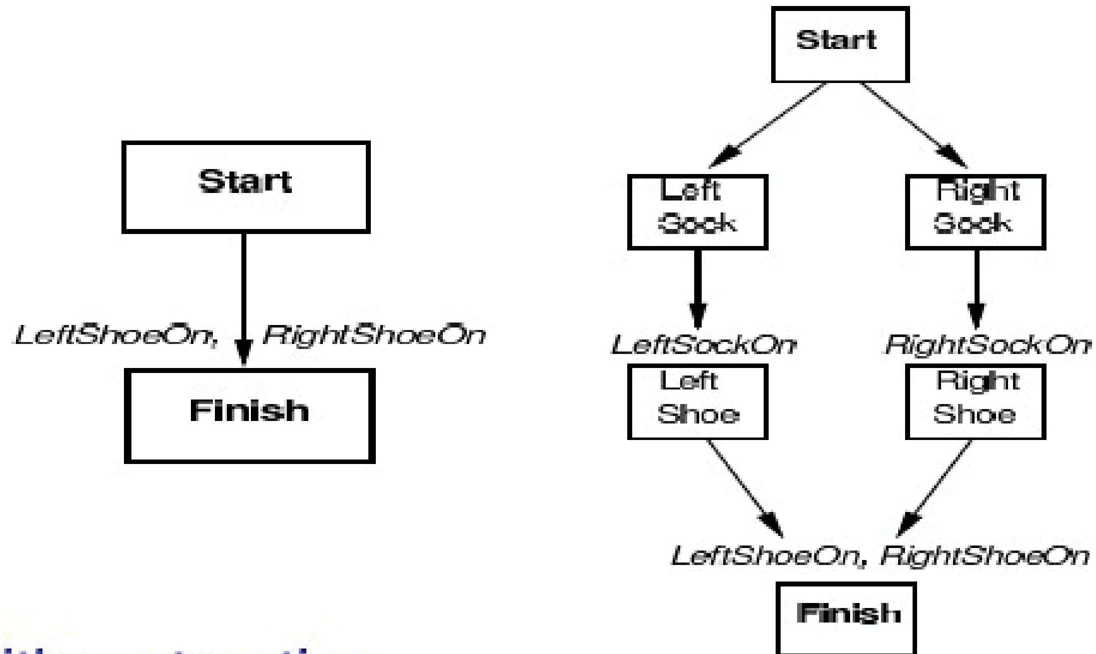
- PSP is both sound and complete
- It returns a partially ordered solution plan
  - Any total ordering of this plan will achieve the goals
  - Or could execute actions in parallel if the environment permits it

Dana Nau: Lecture slides for *Automated Planning*
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Plan-State Search



Partially-Ordered Plans



Special steps with empty action

Start no precond, initial assumptions as effect)

Finish goal as precond, no effect

Partial-Order Plans

Complete plan

A plan is complete iff every precondition is achieved

A precondition c of a step S_j is achieved (by S_i) if

- $S_i \prec S_j$
- $c \in effect(S_i)$
- there is no S_k with $S_i \prec S_k \prec S_j$ and $\neg c \in effect(S_k)$
  (otherwise S_k is called a clobberer or threat)

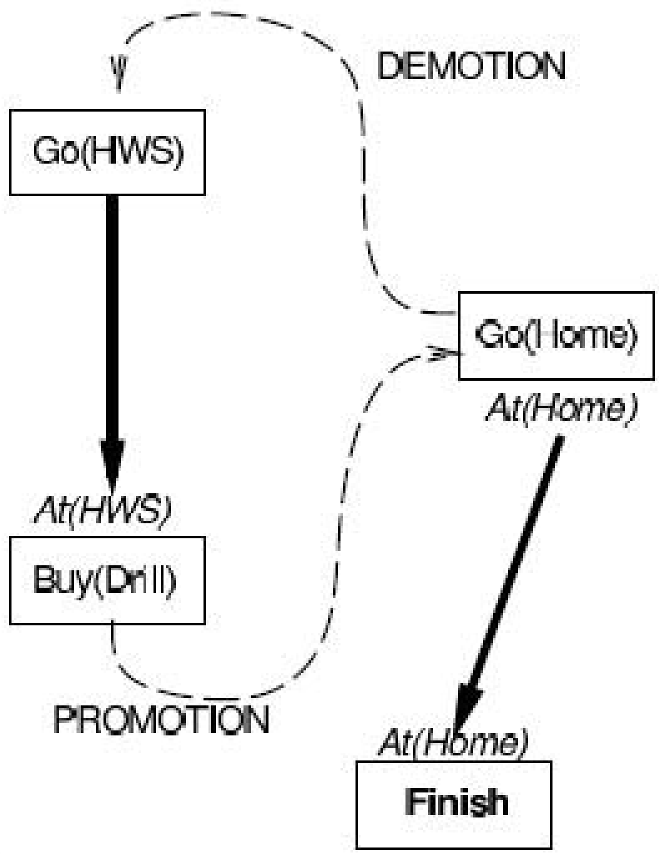
Clobberer / threat

A potentially intervening step that destroys the condition achieved by a causal link

Partial-Order Plans

Example

$Go(Home)$ **clobbers** $At(HWS)$



Demotion

Put before $Go(HWS)$

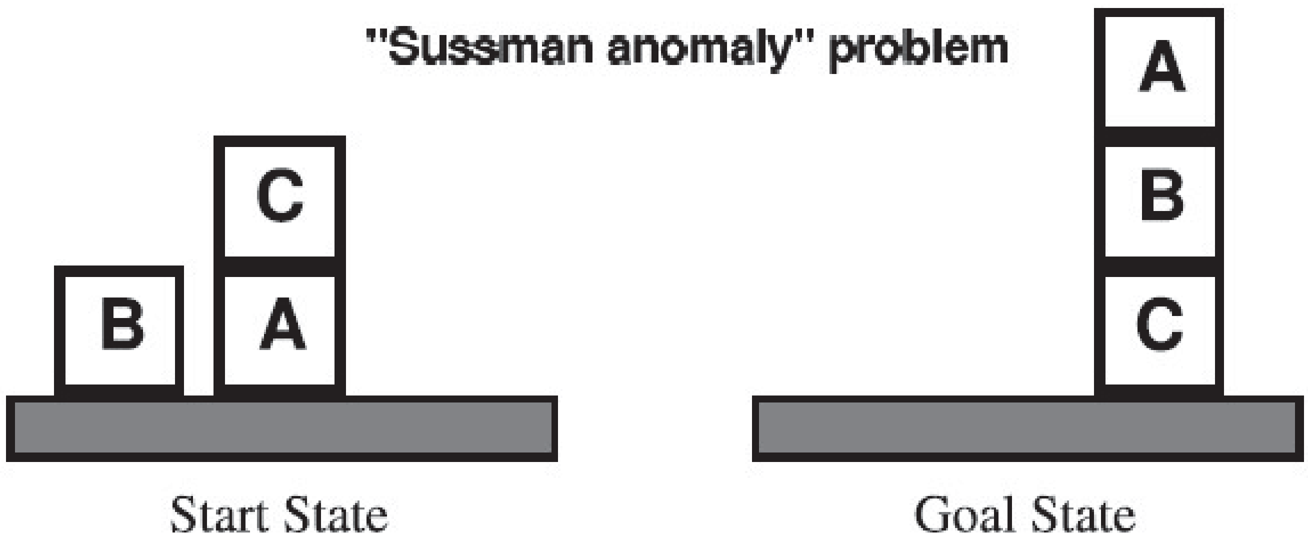
Promotion

Put after $Buy(Drill)$

General Approach

- General Approach
  - Find unachieved precondition
    - Add new action *or* link to existing action
  - Determine if conflicts occur
    - Previously achieved precondition is "clobbered"
    - Fix conflicts (reorder, bind, …)
- Partial-order planning can easily (and optimally) solve blocks world problems that involve goal interactions (e.g., the "Sussman Anomaly" problem)

Blocks World



Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

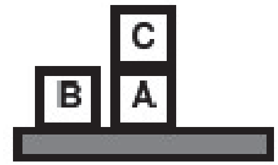
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Blocks World

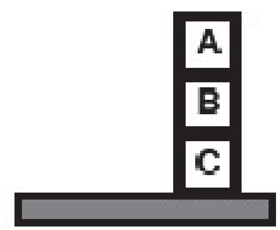
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

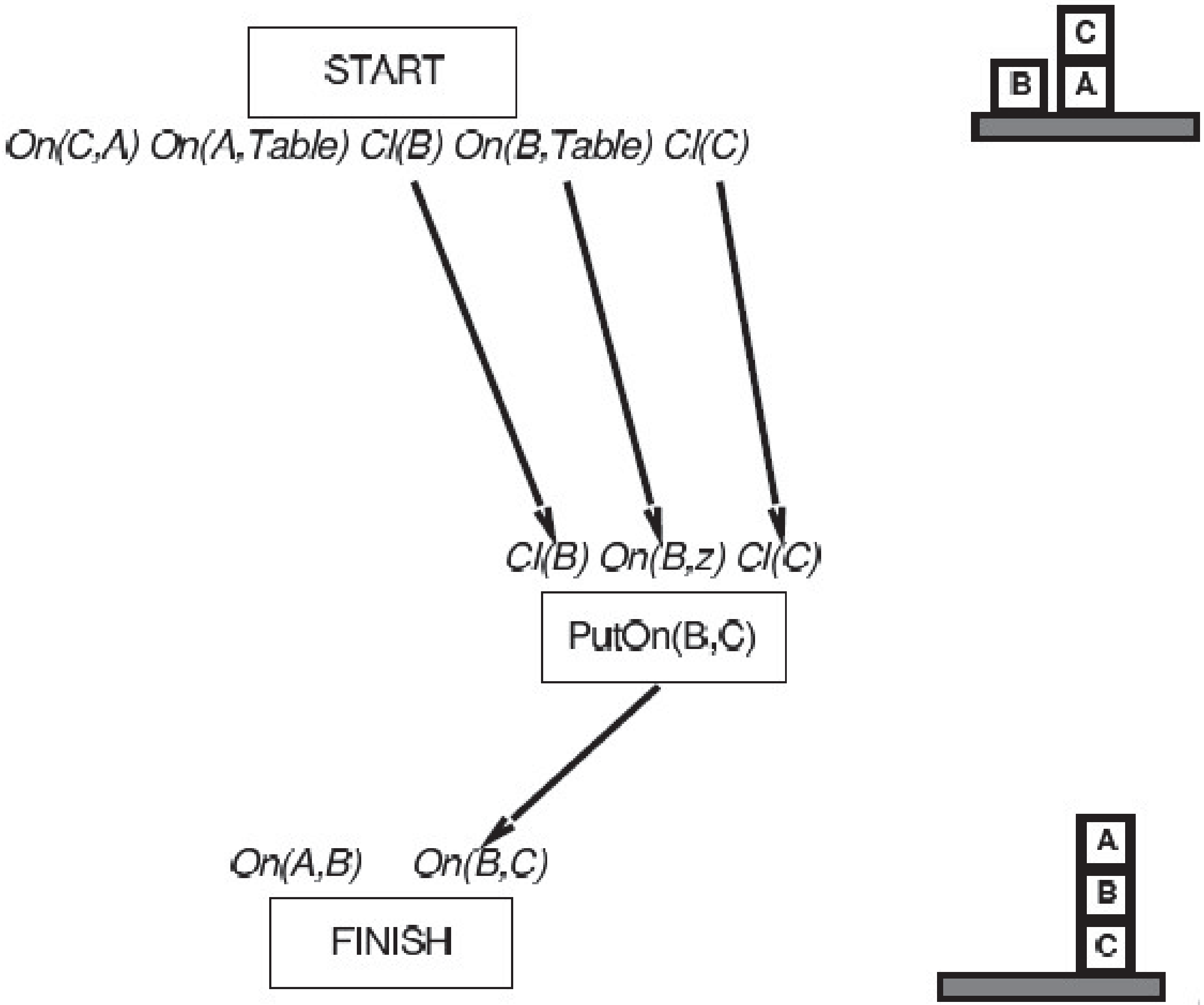


On(A,B) *On(B,C)*

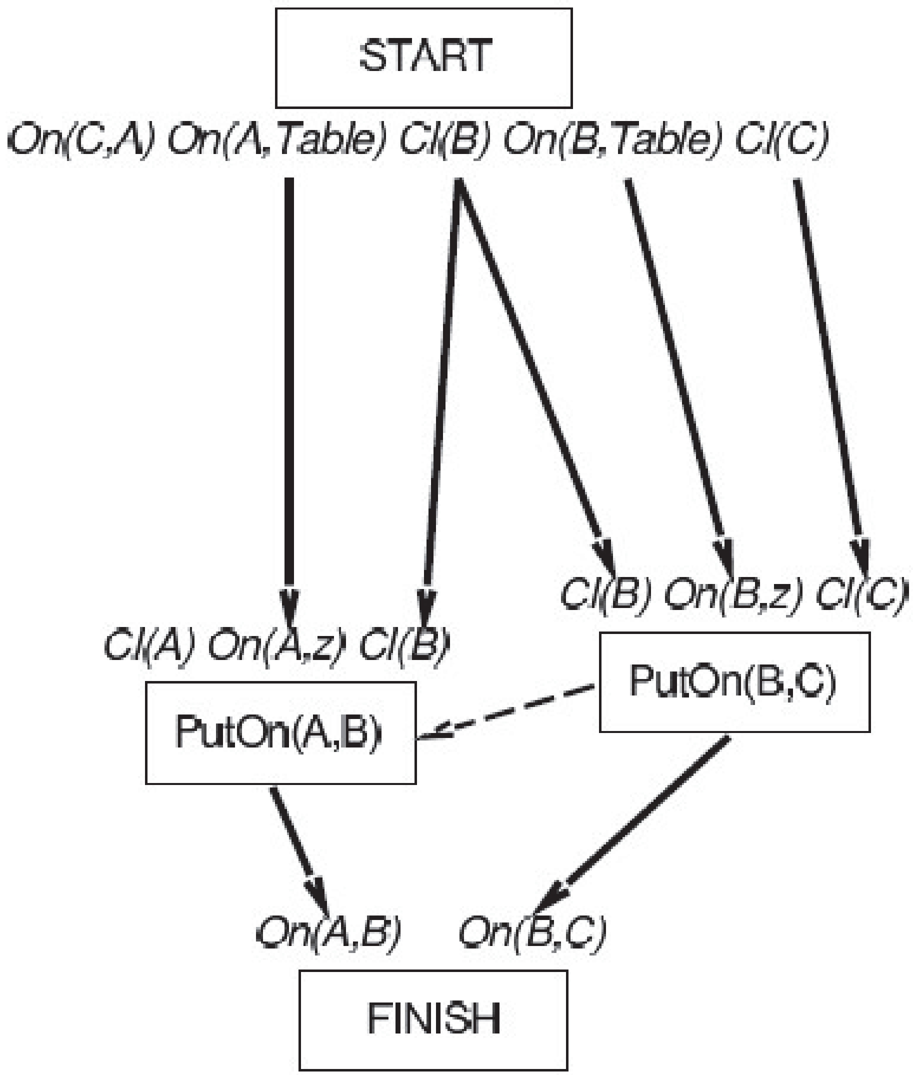
FINISH

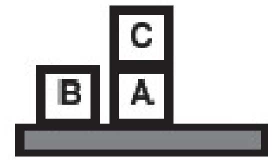


Blocks World

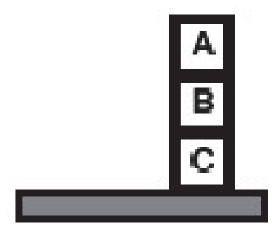


Blocks World

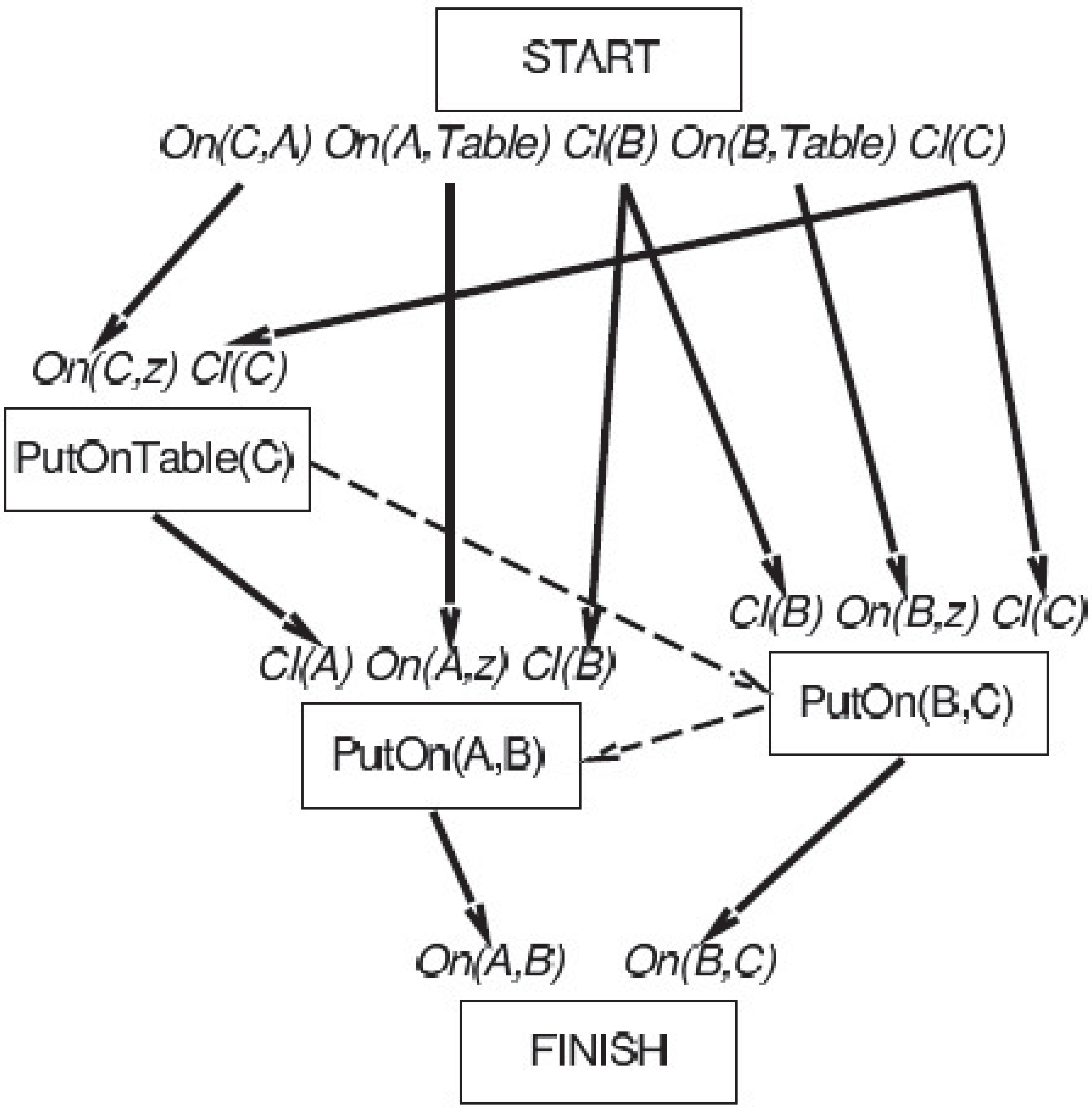


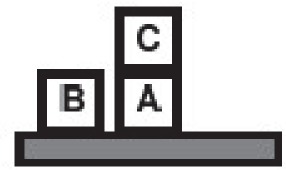


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



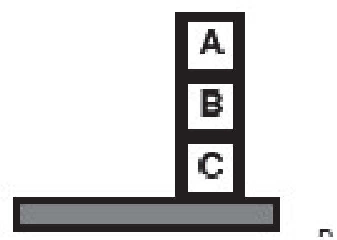
Blocks World



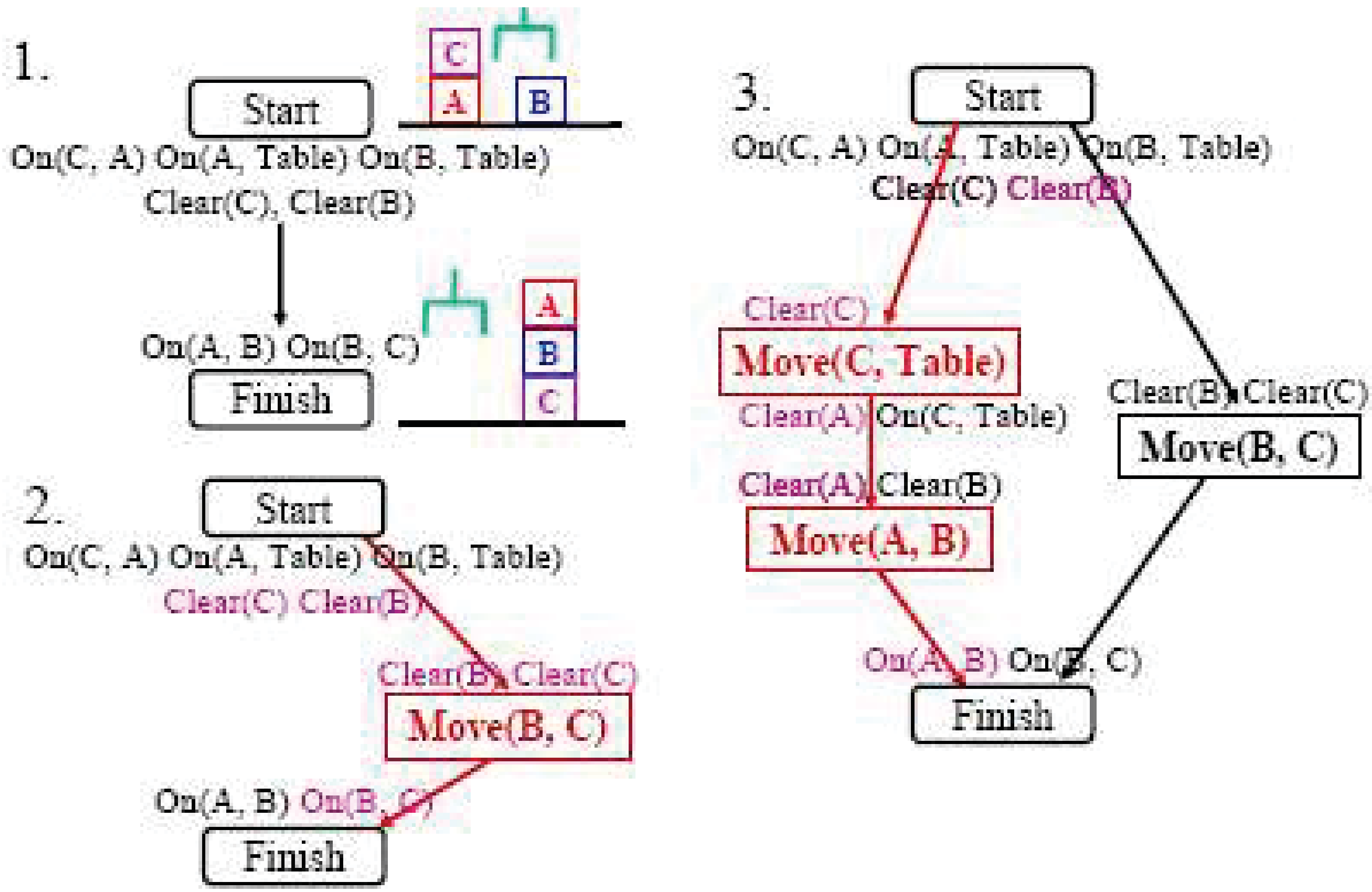


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)

PutOn(B,C)
clobbers Cl(C)
=> order after
PutOnTable(C)

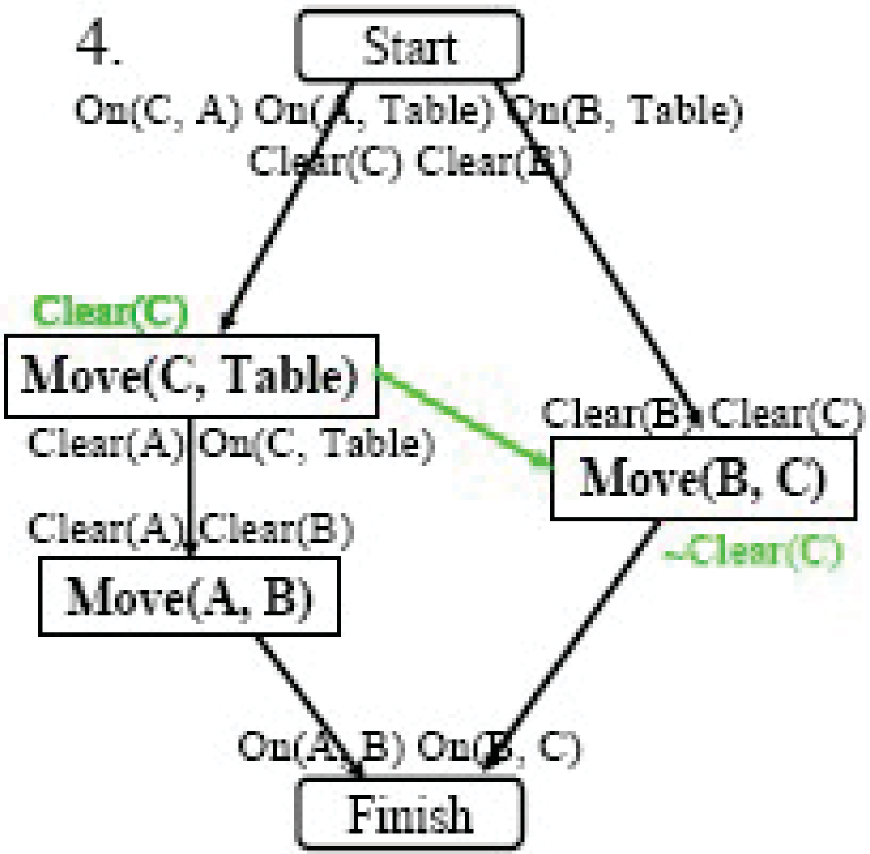


Blocks World

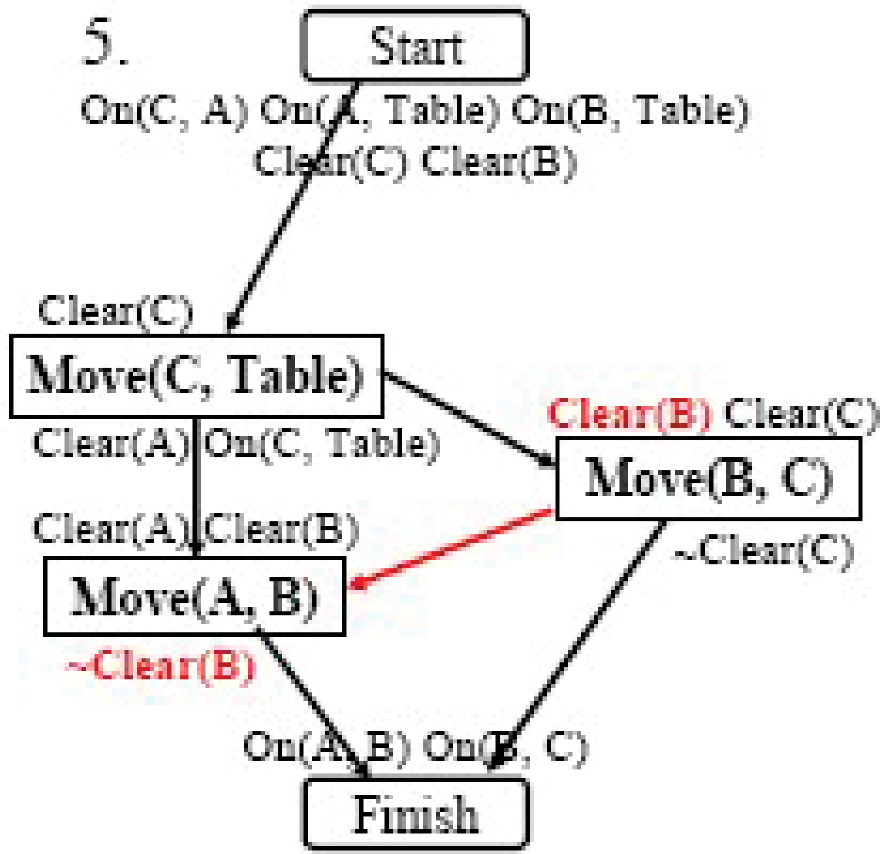


Blocks World

4.



5.



Least Commitment

- Basic Idea
  - *Make choices that are only relevant to solving the current part of the problem*

- Least Commitment Choices
  - **Orderings**: Leave actions unordered, unless they must be sequential
  - **Bindings**: Leave variables unbound, unless needed to unify with conditions being achieved
  - **Actions**: Usually not subject to "least commitment"
- Refinement
  - Only *add* information to the current plan
  - *Transformational* planning can remove choices

Terminology

- ***Totally Ordered*** Plan
  - There exists sufficient orderings O such that all actions in A are ordered with respect to each other
- ***Fully Instantiated*** Plan
  - There exists sufficient constraints in B such that all variables are constrained to be equal to some constant
- ***Consistent*** Plan
  - There are no contradictions in O or B
- ***Complete*** Plan
  - Every precondition p of every action a_i in A is ***achieved***: There exists an effect of an action a_j that comes before a_i and unifies with p, and no action a_k that deletes p comes between a_j and a_i

POP-Algorithm

```
function POP(initial, goal, operators) returns plan

   plan ← MAKE-MINIMAL-PLAN(initial, goal)
   loop do
       if SOLUTION?(plan) then return plan        % complete and consistent
       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
```

POP-Algorithm

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 $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
add S_{add} to STEPS(*plan*)
add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

POP-Algorithm

procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**

choose either

Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)

Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)

if not CONSISTENT(*plan*) **then fail**

end

POP-Algorithm

- Non-deterministic search for plan,
  backtracks over choicepoints on failure:
  - choice of S_{add} to achieve S_{need}
  - choice of promotion or demotion for clobberer
- Sound and complete
- There are extensions for:
  disjunction, universal quantification, negation, conditionals
- Efficient with good heuristics from problem description
  But: very sensitive to subgoal ordering
- Good for problems with loosely related subgoals

POP-Algorithm

- **Advantages**
  - Partial order planning is *sound* and *complete*
  - Typically produces *optimal* solutions (plan length)
  - Least commitment may lead to shorter search times

- **Disadvantages**
  - Significantly more complex algorithms (higher *per-node* cost)
  - Hard to determine what is true in a state
  - Larger search space (**infinite**!)

Plan Monitoring

Execution monitoring

Failure: Preconditions of remaining plan not met

Action monitoring

Failure: Preconditions of next action not met

(or action itself fails, e.g., robot bump sensor)

Consequence of failure

Need to replan

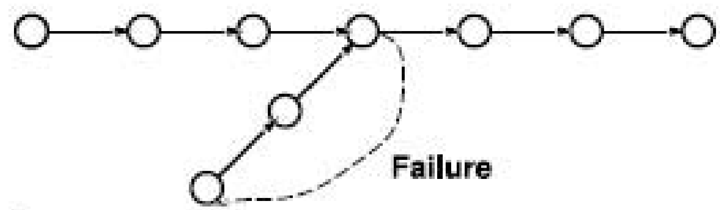
Replanning

Simplest

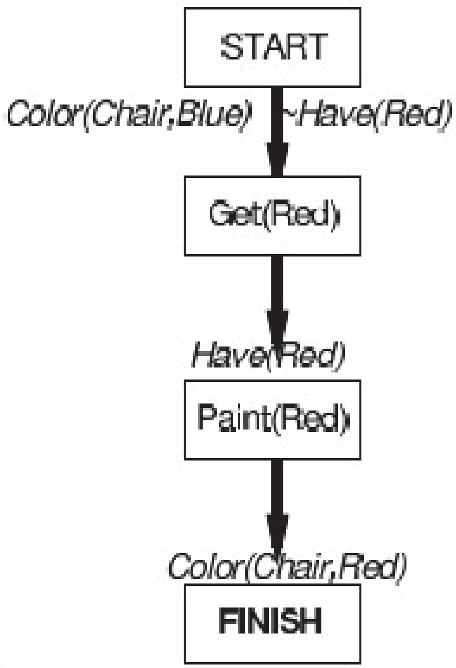
On failure, replan from scratch

Better

Plan to get back on track by reconnecting to best continuation



Replanning



PRECONDITIONS	FAILURE RESPONSE
none	N/A
Have(Red)	Fetch more red
Color(Chair,Red)	Repaint