based on Manuela M. Veloso lectures on

PLANNING, EXECUTION AND LEARNING

POLITECNICO
MILANO 1863

Cognitive Robotics

2016/2017

Planning: State, Actions and Goal Representation

Matteo Matteucci
matteo.matteucci@polimi.it

Artificial Intelligence and Robotics Lab - Politecnico di Milano

Recall «Think hard, act later»?

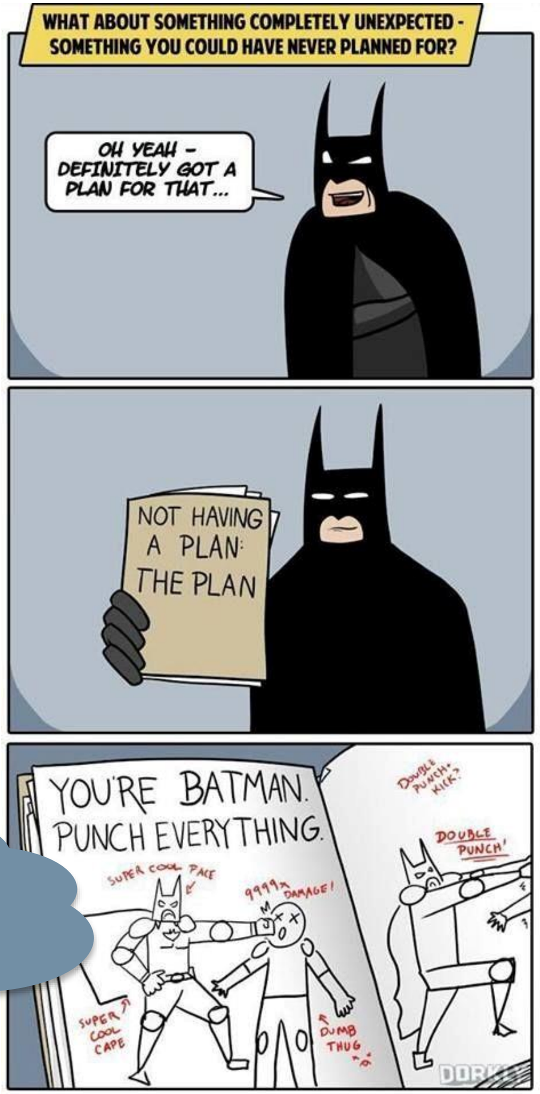
Planning is about «thinking»

- Given the ***actions*** available in a task domain.
- Given a problem specified as:
  - an ***initial state*** of the world
  - a goal statement (***set of goals***) to be achieved
- Find a ***solution*** to the problem

Plan: a way, in terms of a <u>*sequence of actions*</u>, to transform the <u>*initial state*</u> into a <u>*new state*</u> of the world where the <u>*goal statement is true*</u>.

It's all about <u>states</u>, <u>actions</u>, and <u>plans</u>!

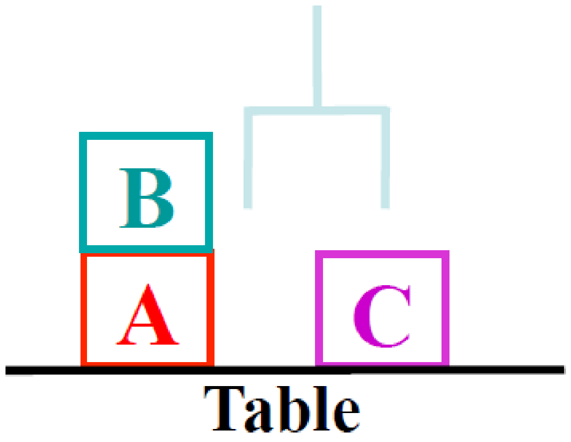
Newell and Simon 1956

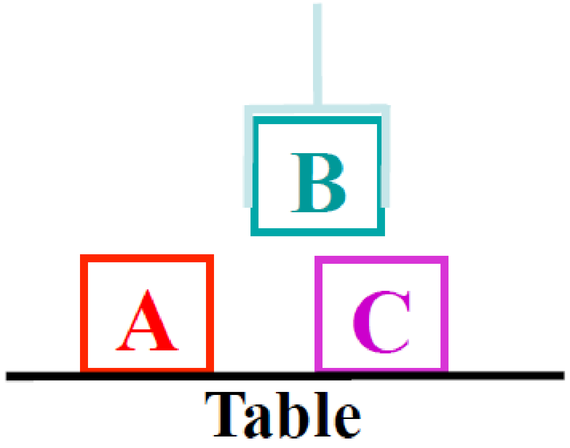


The Block World

The Block World is a useful abstraction to introduce *States, Actions and Plans*

- Blocks are on the Table, or on top of each other.
- There is an Arm – the Arm can be empty or holding one block.
- The table is always clear.





The Block World: States

Objects

- Blocks: *A, B, C*
- Table: *Table*

Some predicates might be redundant

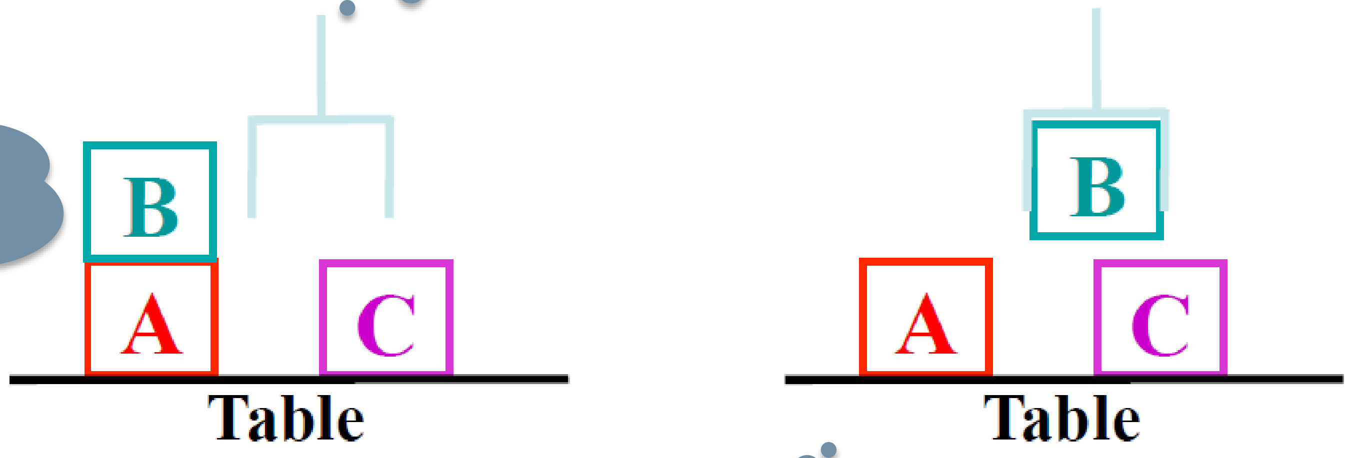
Predicates

- *On(A, B), On(C, Table)*
- *Clear(B), Handempty, Holding(C)*
- *On-table(A), On(A,B), Top(B),…*

States – Conjunctive

- On(A,B) and On(B,C) and Clear(A) and Handempty
- …

On-Table(A), On-Table(C), On(A,B), Clear(C), Clear(B), Handempty



On-Table(A), On-Table(C), Clear(A), Clear(C), Holding(B)

The Block World: Assumptions/Limitations

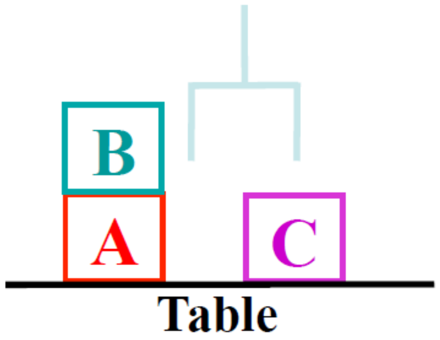
The Block World models Classical Deterministic Planning ...

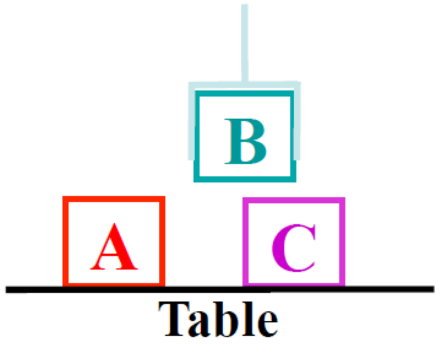
- There is a single initial state
- The description is complete
- The plan is deterministic
- What is not true in the state is false

CWA: Closed World Assumption

The basic operators perform queries on states

- On(A,B) → returns true or false
- On(A,x) → returns x=Table or x=B
- On-table (x) → returns x=A and x=C
- …





The Block World: State Description

A-on-B	*¬A-on-B ∧ ¬A-on-Table*
A-on-Table	*¬B-on-A ∧ ¬B-on-Table*
B-on-A	*¬Holding-A ∧ ¬Holding-B*
B-on-Table	*¬B-on-A*
Holding-A	*¬A-on-B*
Holding-B	
Handempty	
Clear-A	
Clear-B	

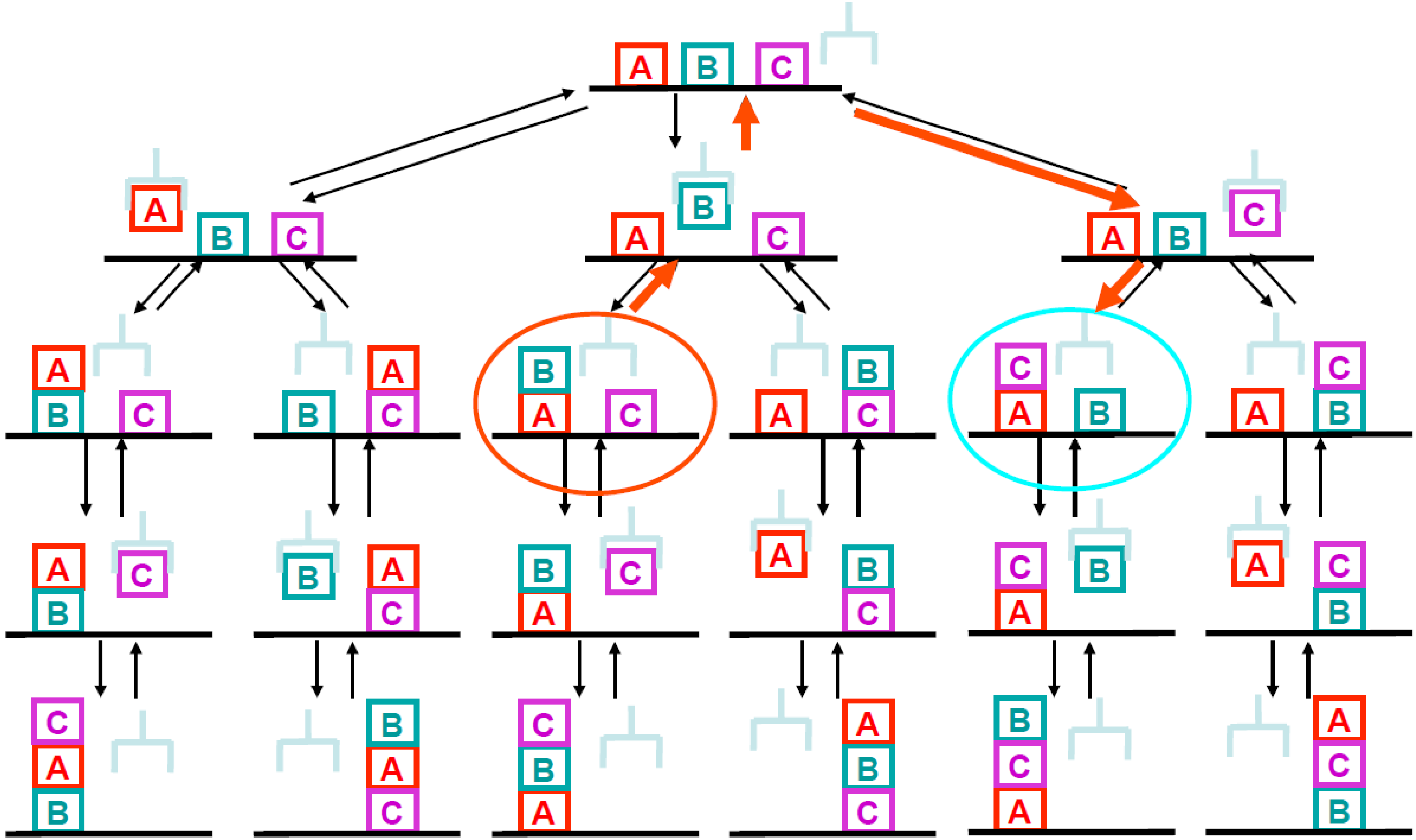
2^4 Possible states

All these define the State Space

A-on-x {∅, table, B}
B-on-x {∅, table, A}

3^2 Possible States

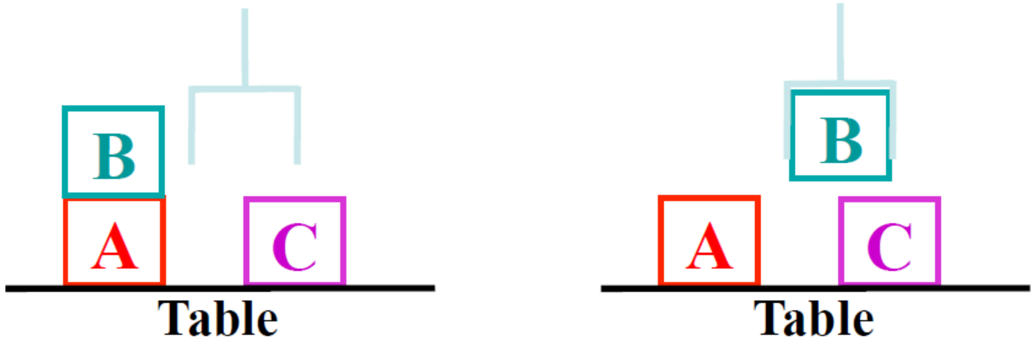
The Block World: Planning as State-Space Search



Models for State Spaces

Different models for states exist ...

- Atomic identification of states (s1, s2,...)
- Symbolic feature based states
- Symbolic predicate based states
- …



… together with different ways of combining them

- Conjunctive → observable
- Probabilistic → approximate
- Incremental → on-demand
- Temporal → dynamic

Predicates, conjunctive, complete, correct, deterministic

Goal Specification

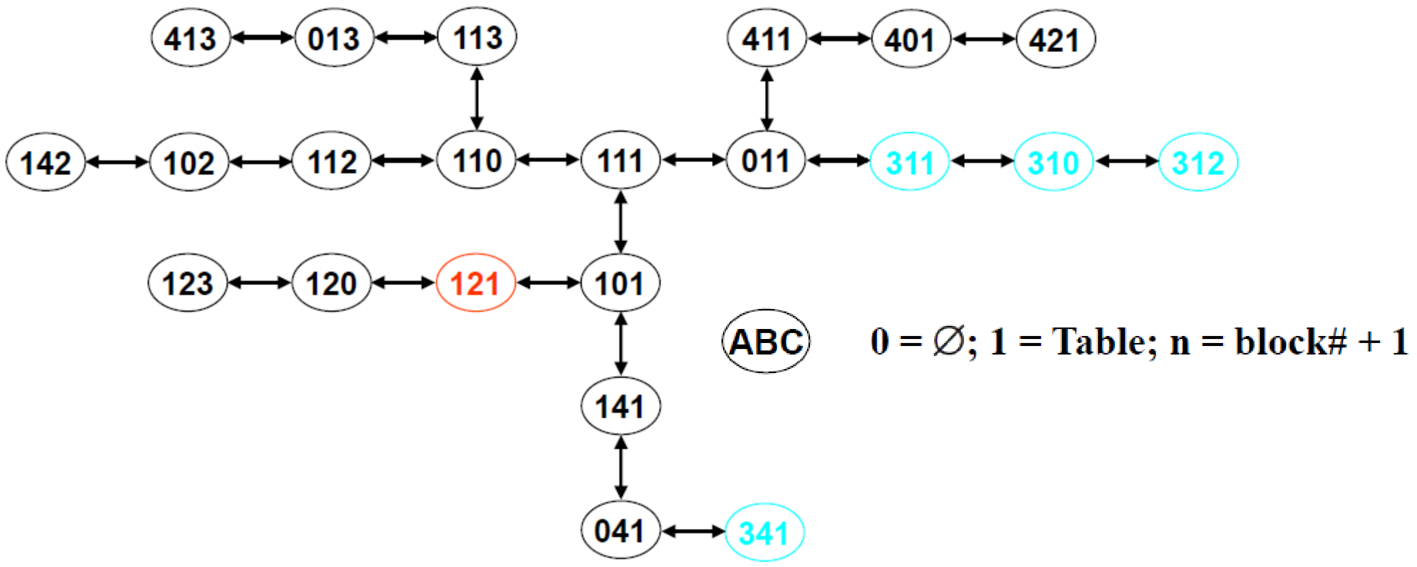
We can specify a Goal according to different levels of generality:

- Goal State → Completely specified state
- Goal Statement → Partially specified state
- Objective function → Defines "good" or "optimal" plan

Increased Generality

Goal Statement example:

- Initial: A-on-x = Table;
  B-on-x = A;
  C-on-x = Table
- Goal: A-on-x = B



What is an Action?

Plan: a way, in terms of a *sequence of actions*, to transform the *initial state* into a *new state* of the world where the *goal statement is true*.

Newell and Simon 1956

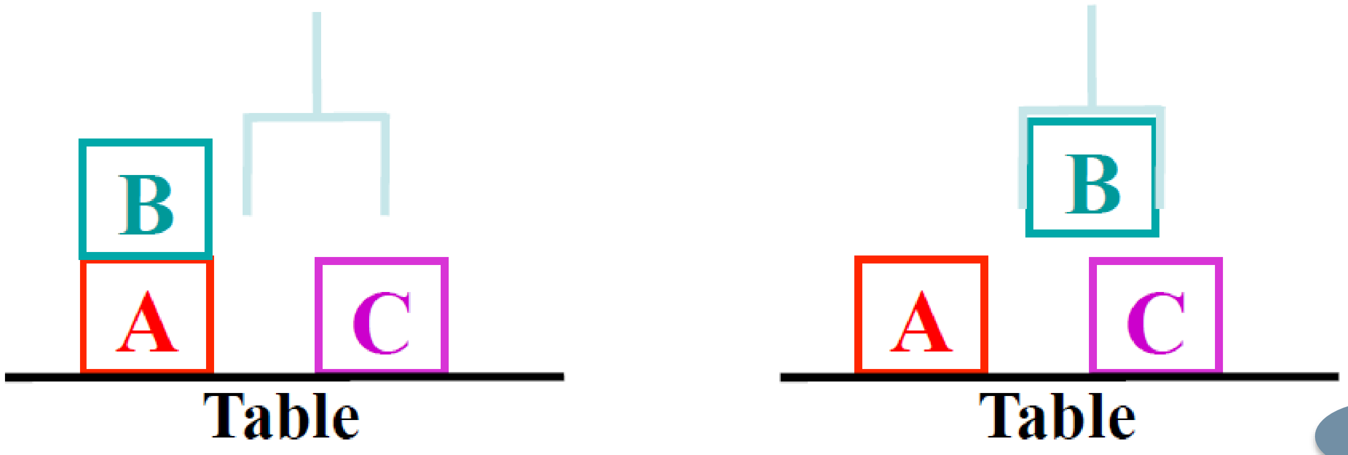
Action: a transition from one (partial) state to another

- May be applicable only in particular states
- Generates new state
  - Deterministic: t_{det}: S x A → S
  - Non-deterministic: $t_{non\text{-}det}$: S x A → 2^S
  - Probabilistic: t_{prob}: S x A → <2^S, r>

	011	013	041	101	102	110	111	112	113	120	121	123	141	142	310	311	312	341	401	411	413	421
011							1									1				1		
013									1												1	
041													1					1				
101							1				1		1									
102								1						1								
110							1	1	1													
111	1			1		1																
112					1	1																
113		1				1																
120											1	1										
121				1						1												
123										1												
141			1	1																		
142					1																	
310																1	1					
311	1														1							
312															1							
341			1																			
401																				1		1
411	1																		1			
413		1																				
421																			1			

Explicit Action Representation

The Block World Dynamics: Actions



How do these transform a state into another?

- Blocks are on the Table, or on top of each other
- Blocks are picked up and put down by the arm
- A block can be picked up only if it is clear, i.e., without a 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 pick up only one block at a time
- The arm can put down blocks on blocks or on the table
- The table is always clear

STRIPS Action Representation

STRIPS (Stanford Research Institute Problem Solver) was the planner used by Shakley, it was developed at SRI International by Richard Fikes and Nils Nilsson in 1971.

Explicit action a representation

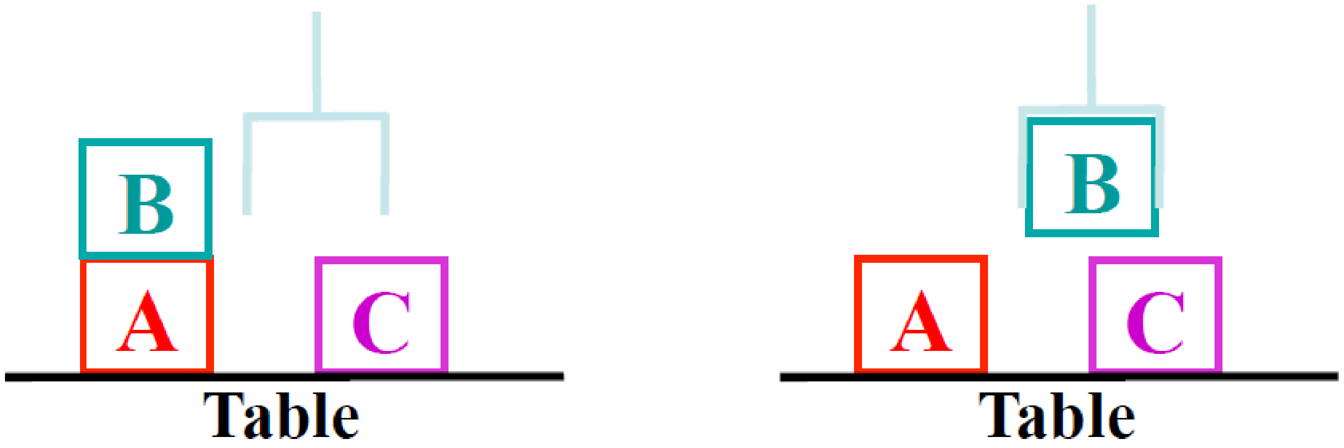
- $\{\text{preconds}(a), \text{effects}^{-}(a), \text{effects}^{+}(a)\}$
- $\text{effects}^{-}(a) \cap \text{effects}^{+}(a) = \emptyset$
- $\tau(\mathcal{S}, a) = \{\mathcal{S} - \text{effects}^{-}(a) \cup \text{effects}^{+}(a)\}$,
  where $\mathcal{S} \in 2^{S}$

Example in the Block World

- Pickup_from_table(?b)
  Pre: ...
  Add: ...
  Delete: ...

Let's try this out together!

Actions in the Block World



In the Block World:

- An action *a* is **applicable** in *s* if all its preconditions *are satisfied by s*.
- RESULT(*s,a)* = (*s* – Del (*a*)) U Add (*a*)
- No explicit mention of *time*
    - The precondition always refers to time *t*
    - The effect always refers to time *t+1*

The Block World: Actions

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

Pickup_from_block(b1, b2)
- Pre: Block(b1),Block(b2), Handempty
  - Clear(b1), On(b1,b2)
- Add: Holding(b1), Clear(b2)
- Delete: Handempty, On(b1,b2)
  - Clear (b1)

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

Putdown_on_block(b1, b2)
- Pre: Block(b1), Holding(b1)
  - Block(b2), Clear(b2), $b1 \neq b2$
- Add: Handempty, On(b1, b2)
- Delete: Holding(b1), Clear(b2)

More Realistic Actions Representations

Conditional Effects

- ```
  Pickup (b)
      Pre: Block(b), Handempty, Clear(b), On(b, x)
      Add: Holding(b)
               if (Block(x)) then Clear(x)
      Delete: Handempty, On(b, x)
  ```

Quantified Effects

- ```
  Move (o, x)
      Pre: At(o, y), At(Robot, y)
      Add: At(o, x), At(Robot, x)
             forall (Object(u)) [ if (In(u, o)) then At(u, y)]
      Delete: At(o, y), At(Robot, y),
                forall (Object(u)) [ if (In(u, o)) then At(u, y)]
  ```

Disjunctive and Negated Preconditions

- `Holding(x) Or Not[Lighter_Than_Air(x)]`

All these extensions can be emulated adding actions!

More Realistic Actions Representations

These extensions make the planning problem significantly harder

Inference Operators / Axioms

- `Clear(x) iff forall(Block(y))[ Not[On(y, x)]]`

Functional effects

- Move (o, x)

```
    Pre: At(o, y), At(Robot, y), Fuel(f), f ≥ Fuel_Needed(y, x)
    Add: At(o, x), At(robot, x), Fuel(f - Fuel_Needed(y, x)),
              forall (Object(u)) [ if (In(u, o)) then At(u, y)]
    Delete: At(o, y), At(Robot, y), Fuel(f),
              forall (Object(u)) [ if (In(u, o)) then At(u, y)]
```

Disjunctive Effects

- Pickup_from_block(b)

```
    Pre: Block(b), Handempty, Clear(b), On(b, c), Block(c)
C1: Add: Clear(c), Holding(b); Delete: On(b, c), Handempty
C2: Add: Clear(c), On(b, Table); Delete: On(b, c)
C3: Add: ; Delete:
```

Much harder and you can add probability!!!

Cognitive Robotics

Planning: Plan Generation

Matteo Matteucci
matteo.matteucci@polimi.it

Artificial Intelligence and Robotics Lab - Politecnico di Milano

Different Plans ...

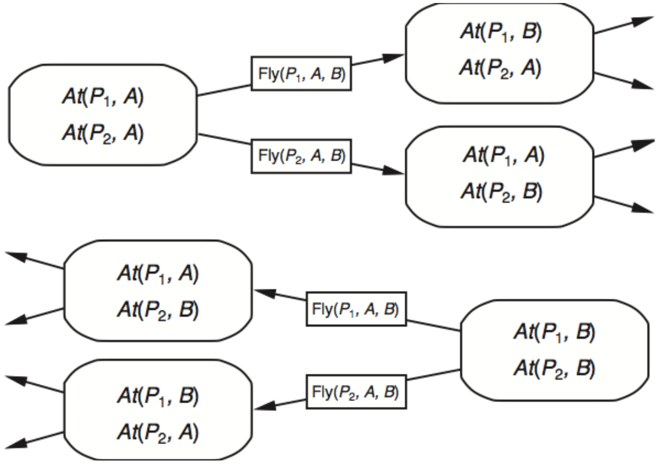
A plan can have different degrees of generality …

- Sequence of Instantiated Actions
- Partial Order of Instantiated Actions
- Set of Instantiated Actions
- Policy (a direct mapping from states to actions)

Increased Generality

… and adopt different search stratiegies:

- Progression, a.k.a. forward state space search, a.k.a. forward chaining
- Regression, a.k.a. backward state-space search, a.k.a. backward chaining



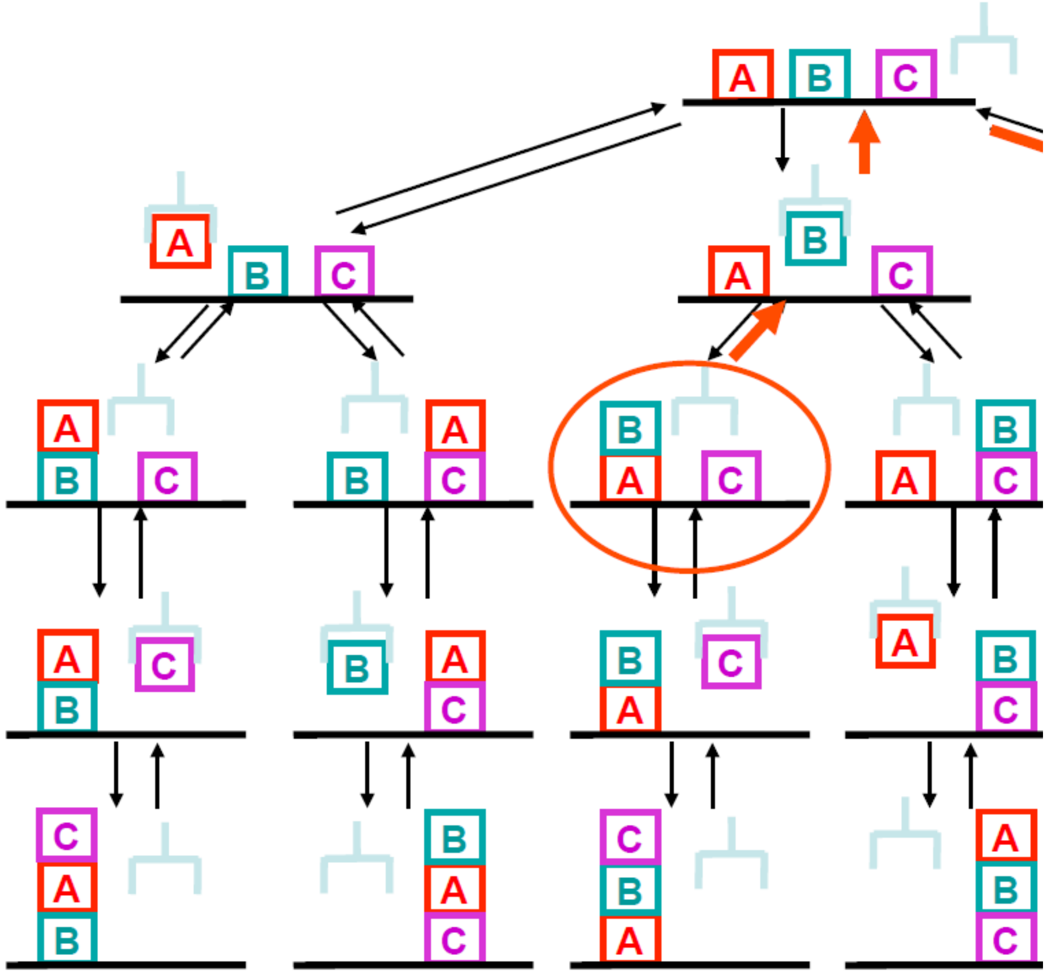
Plan Generation

Backtracking Search Through a Search Space

- How to conduct the search
- How to represent the search space
- How to evaluate the solutions

Non-Deterministic Choices Determine Backtracking

- Choice of actions
- Choice of variable bindings
- Choice of temporal orderings
- Choice of subgoals to work on



Properties of Planning Algorithms

Soundness

- A planning algorithm is ***sound*** if all solutions are legal plans, i.e., all preconditions, goals, and any additional constraints are satisfied

Completeness

- A planning algorithm is ***complete*** if a solution can be found whenever one actually exists
- A planning algorithm is ***strictly complete*** if all solutions are included in the search space

Optimality

- A planning algorithm is ***optimal*** if it maximizes a predefined measure of plan quality

Linear Planning and Means-ends Analysis

Linear Planning

- Uses a Goal stac and work on one goal until completely solved before moving on to the next goal

Mean-ends Analysis

- Search by reducing the difference between the state and the goals, i.e., What means (operators) are available to achieve the desired ends (goal)?

Newell and Simon 60s

```
GPS Algorithm (state, goals, plan)
    If goals ⊆ state, then return (state,plan)
    Choose a difference d ∈ goals between state and goals
    Choose an operator o to reduce the difference d
    If no applicable operators, then return False
    (state,plan) = GPS (state, preconditions(o), plan)
    If state, then return GPS (apply (o, state), goals, [plan,o])

Initial call: GPS (initial-state, initial-goals, [])
```

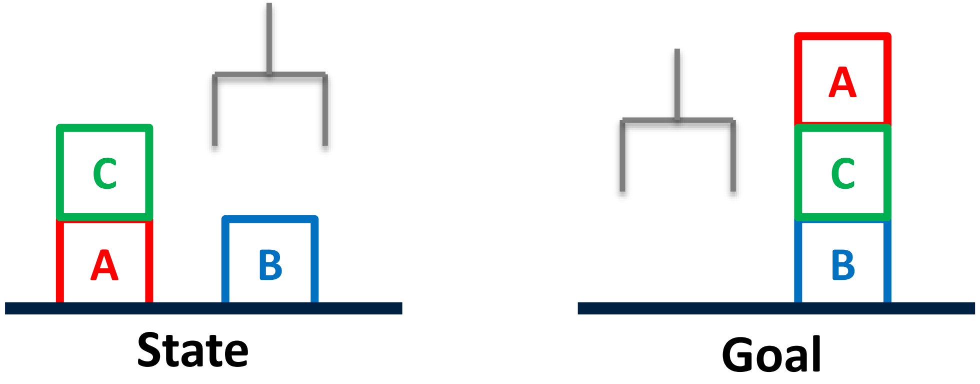
The Block World: GPS at Work

1. Search Stack

- On(A, C) On(C, B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



2. Search Stack

- On(A, C) On(C, B)
- On(A, C)
- On(C, B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

3. Search Stack

- On(A, C) On(C, B)
- On(A, C)
- Put_Block(C, B)
- Holding(C) Clear(B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

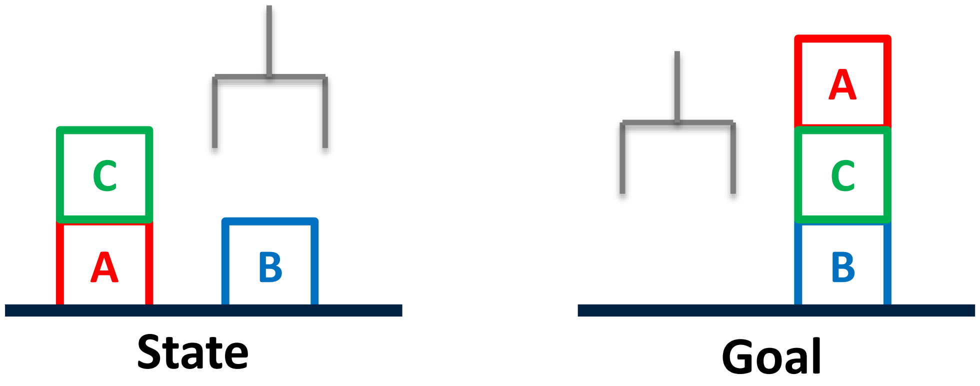
The Block World: GPS at Work

4. Search Stack

- **On(A, C) On(C, B)**
- **On(A, C)**
- **Put_Block(C, B)**
- **Holding(C) Clear(B)**
- **Holding(C)**
- **Clear(B)**

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



5. Search Stack

- **On(A, C) On(C, B)**
- **On(A, C)**
- **Put_Block(C, B)**
- **Holding(C) Clear(B)**
- **Holding(C)**

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

6. Search Stack

- **On(A, C) On(C, B)**
- **On(A, C)**
- **Put_Block(C, B)**
- **Holding(C) Clear(B)**
- **Pick_Block(C)**
- **Handempty Clear(C) On(C, ?b)**

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

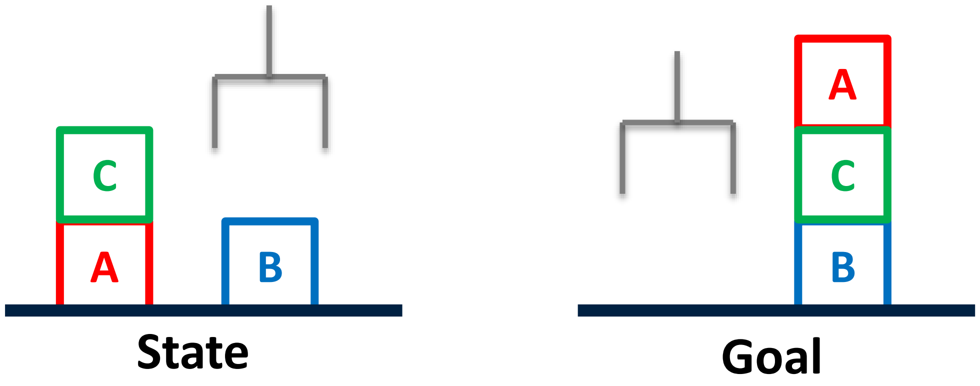
The Block World: GPS at Work

7. Search Stack

- **On(A, C) On(C, B)**
- **On(A, C)**
- **Put_Block(C, B)**
- **Holding(C) Clear(B)**
- **Pick_Block(C)**

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



8. Search Stack

- **On(A, C) On(C, B)**
- **On(A, C)**
- **Put_Block(C, B)**
- **Holding(C) Clear(B)**

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)
Clear(A)

[Pick_Block(C)]

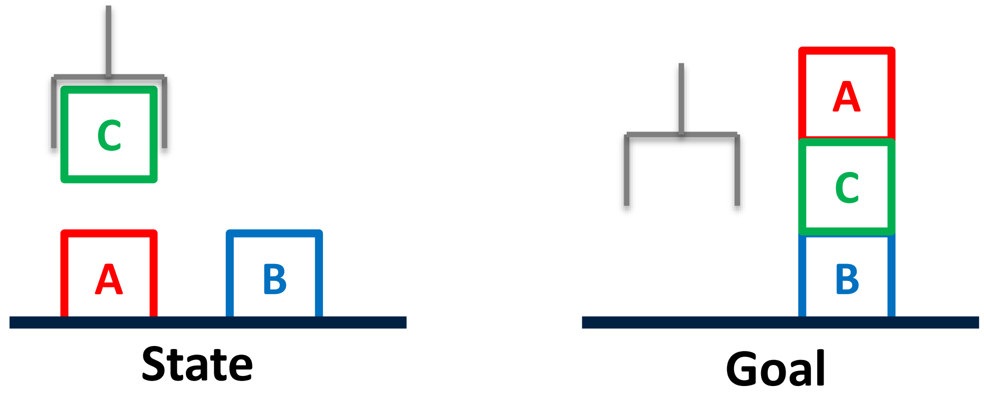
The Block World: GPS at Work

7. Search Stack

- On(A, C) On(C, B)
- On(A, C)
- Put_Block(C, B)
- Holding(C) Clear(B)
- Pick_Block(C)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



8. Search Stack

- On(A, C) On(C, B)
- On(A, C)
- Put_Block(C, B)
- Holding(C) Clear(B)

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)
Clear(A)

[Pick_Block(C)]

9. Search Stack

- On(A, C) On(C, B)
- On(A, C)
- Put_Block(C, B)

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)
Clear(A)

[Pick_Block(C)]

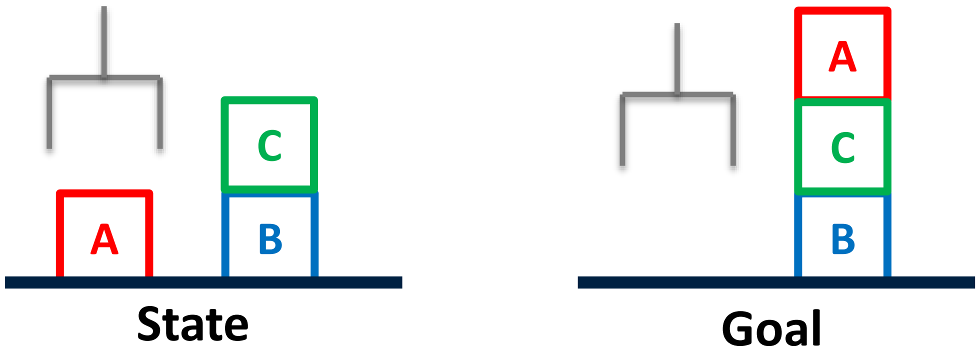
The Block World: GPS at Work

10. Search Stack | **State**

- On(A, C) On(C, B)
- On(A, C)

State: Clear(C), On(A, Table), On(B, Table), Clear(A), Handempty, On(C, B)

[Pick_Block(C); Put_Block(C, B)]



11. Search Stack | **State**

- On(A, C) On(C, B)
- Put_Block(A, C)
- Holding(A) Clear(C)

State: Clear(C), On(A, Table), On(B, Table), Clear(A), Handempty, On(C, B)

[Pick_Block(C)
Put_Block(C, B)]

12. Search Stack | **State**

- On(A, C) On(C, B)
- Put_Block(A, C)
- Holding(A) Clear(C)
- Holding(A)
- Clear(C)

State: *Clear(C)*, On(A, Table), On(B, Table), Clear(A), Handempty, On(C, B)

[Pick_Block(C)
Put_Block(C, B)]

The Block World: GPS at Work

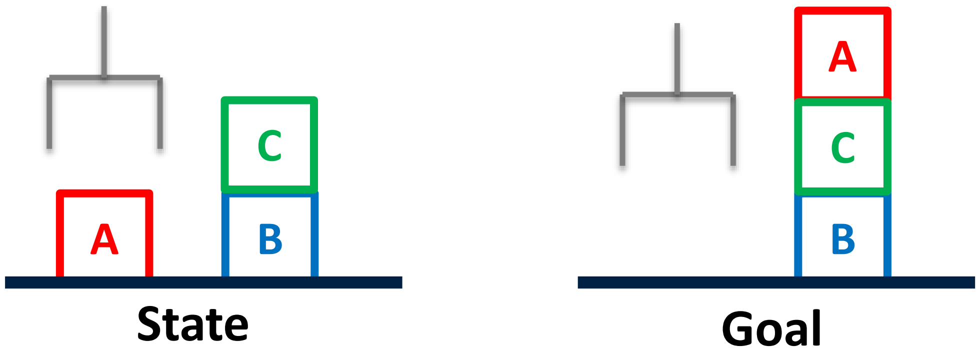
13. Search Stack

- **On(A, C) On(C, B)**
- **Put_Block(A, C)**
- **Holding(A) Clear(C)**
- **Holding(A)**

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick_Block(C); Put_Block(C, B)]



14. Search Stack

- **On(A, C) On(C, B)**
- **Put_Block(A, C)**
- **Holding(A) Clear(C)**
- **Pick_Table(A)**
- **Handempty Clear(A) On(A, Table)**

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick_Block(C); Put_Block(C, B)]

15. Search Stack

- **On(A, C) On(C, B)**
- **Put_Block(A, C)**
- **Holding(A) Clear(C)**
- **Pick_Table(A)**

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick_Block(C); Put_Block(C, B)]

The Block World: GPS at Work

16. Search Stack

On(A, C) On(C, B)
Put_Block(A, C)
Holding(A) Clear(C)

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A)]

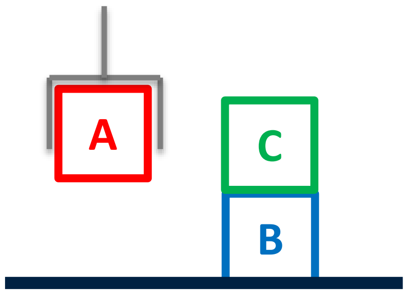
17. Search Stack

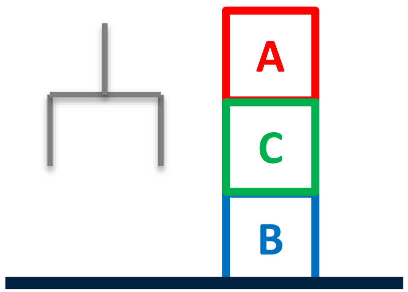
On(A, C) On(C, B)
Put_Block(A, C)

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A)]





The Block World: GPS at Work

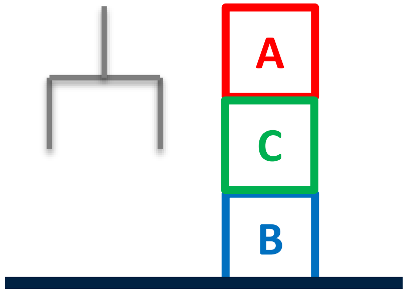
16. Search Stack

- **On(A, C) On(C, B)**
- **Put_Block(A, C)**
- **Holding(A) Clear(C)**

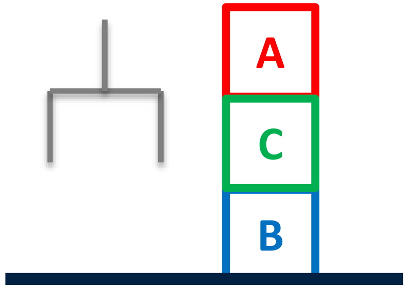
State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A)]



State



Goal

17. Search Stack

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

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A)]

18. Search Stack

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

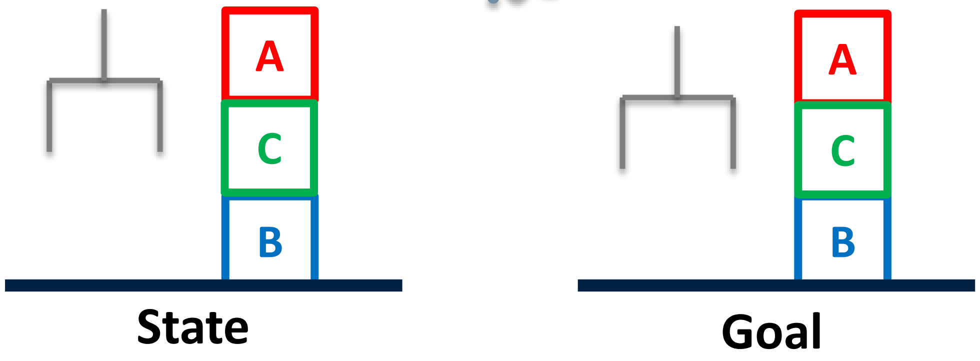
State

On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A);
Put_Block(A, C)]

The Block World: GPS at Work

Sound? Optimal? Complete?



16. Search Stack

On(A, C) On(C, B)
Put_Block(A, C)
Holding(A) Clear(C)

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick_Block(C); Put_Block(C, B); Pick_Table(A)]

17. Search Stack

On(A, C) On(C, B)
Put_Block(A, C)

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick_Block(C); Put_Block(C, B); Pick_Table(A)]

18. Search Stack

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

State

On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

[Pick_Block(C); Put_Block(C, B); Pick_Table(A); Put_Block(A, C)]

19. Search Stack

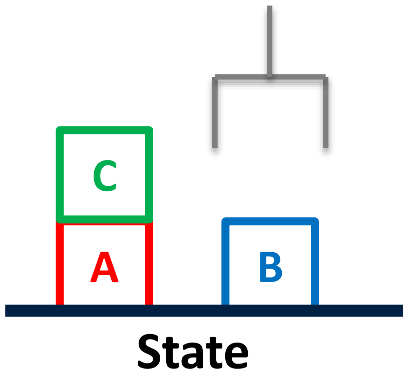
State

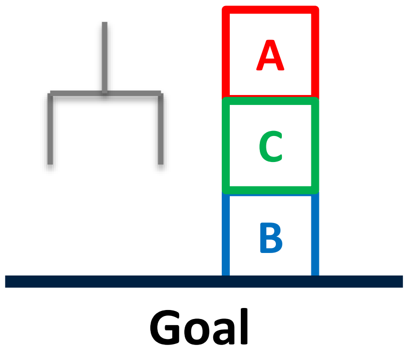
On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

[Pick_Block(C); Put_Block(C, B); Pick_Table(A); Put_Block(A, C)]

The Sussman Anomaly

```
Pickup(?b)
Pre: (handempty)
     (clear ?b)
     (on-table ?b)
Add: (holding ?b)
Delete: (handempty)
        (on-table ?b)
        (clear ?b)
```






```
Putdown(?b)
Pre: (holding ?b)
Add: (handempty)
     (on-table ?b)
Delete: (holding ?b)
```

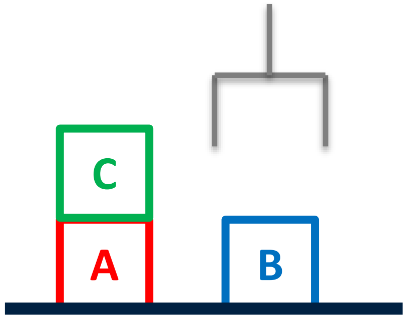
```
Unstack(?a, ?b)
Pre: (handempty)
     (clear ?a)(on ?a ?b)
Add: (holding ?a)(clear ?b)
Delete: (handempty)
        (on ?a ?b)(clear ?a)
```

```
Stack(?a, ?b)
Pre: (holding ?a)
     (clear ?b)
Add: (handempty)(on ?a ?b)
Delete: (holding ?a)
        (clear ?b)
```

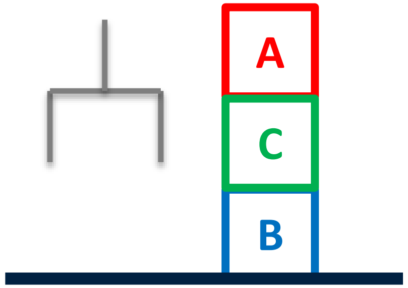
The Sussmann Anomaly – Linear Solution 1

```
(on B C)
```

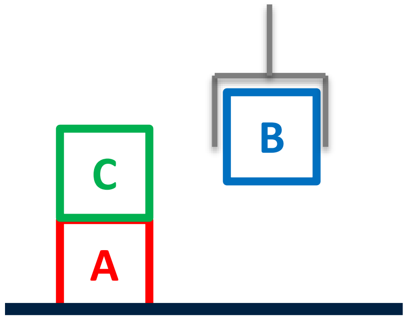
- `Pickup (B)`
- `Stack (B, C)`



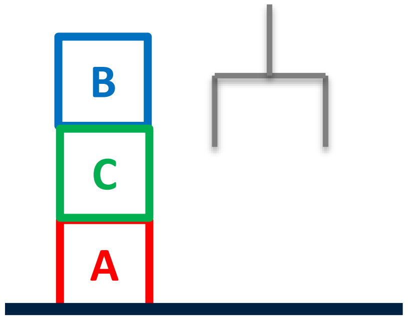
State



Goal



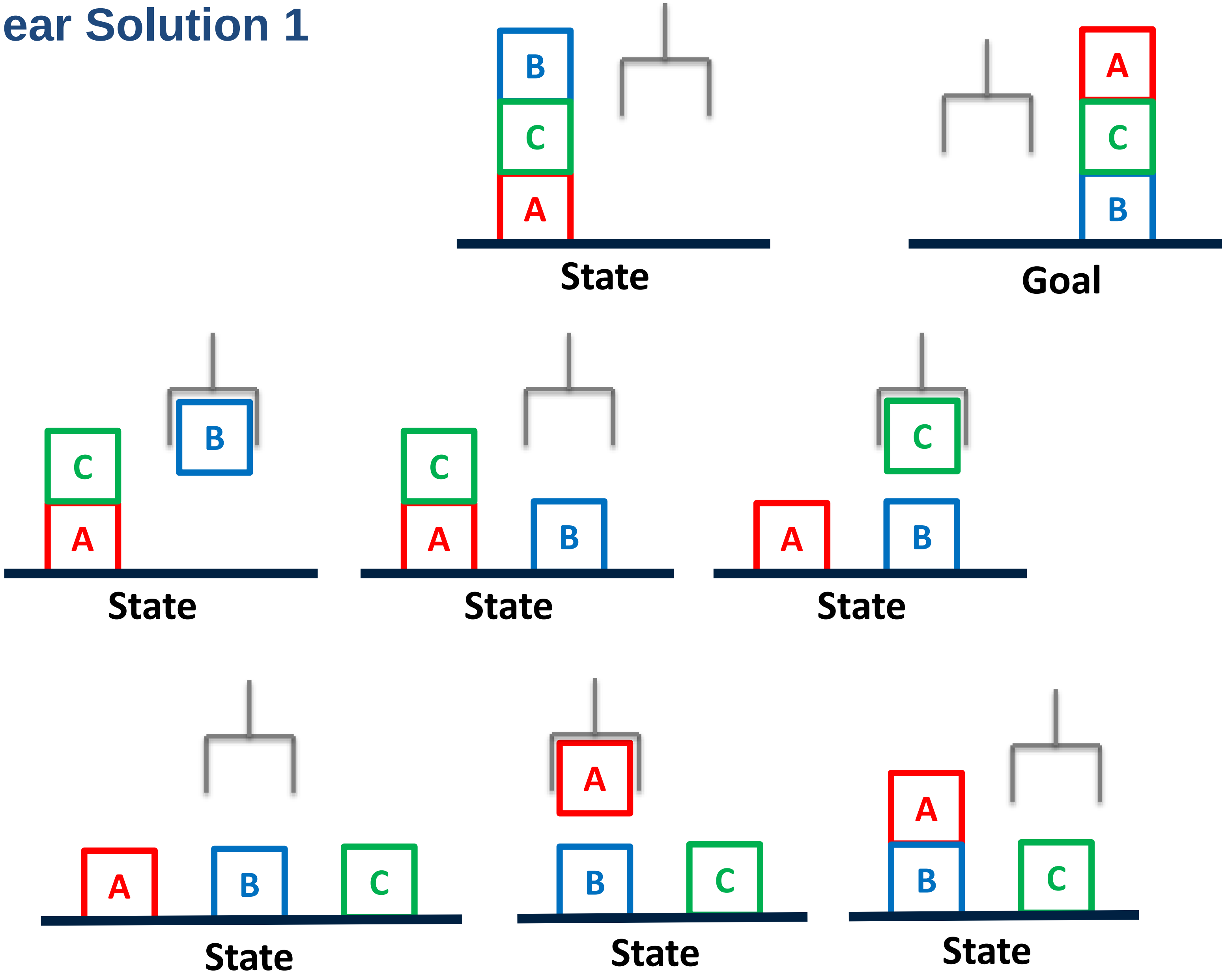
State



State

The Sussmann Anomaly – Linear Solution 1

```
(on B C)
    • Pickup (B)
    • Stack (B, C)
(on A B)
    • Unstack (B, C)
    • Putdown (B)
    • Unstack (C, A)
    • Putdown (C)
    • Pickup(A)
    • Stack (A, B)
```



The Sussmann Anomaly – Linear Solution 1

(on B C)

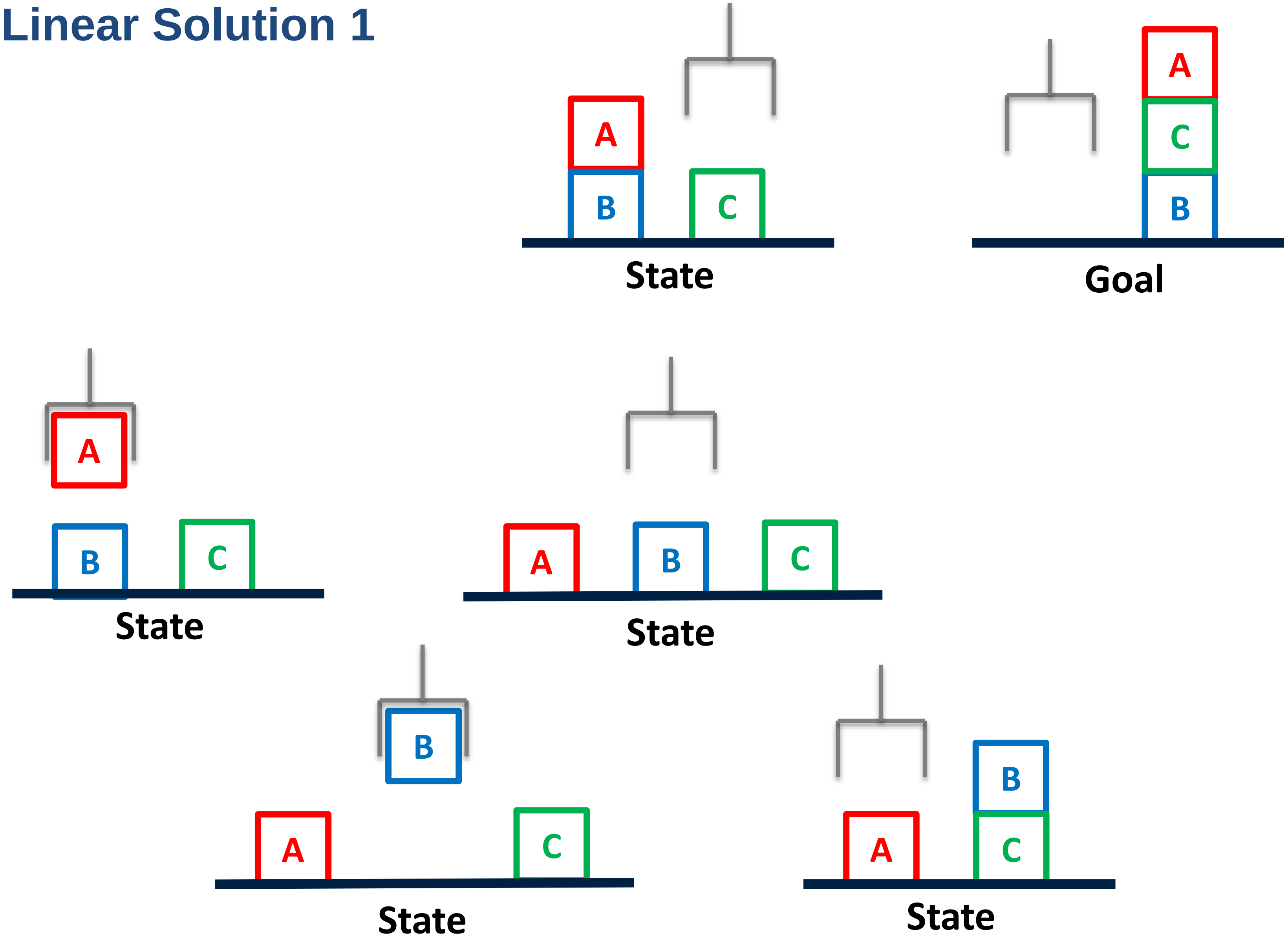
- Pickup (B)
- Stack (B, C)

(on A B)

- Unstack (B, C)
- Putdown (B)
- Unstack (C, A)
- Putdown (C)
- Pickup(A)
- Stack (A, B)

(on B C)

- Unstack (A, B)
- Putdown (A)
- Pickup (B)
- Stack (B, C)



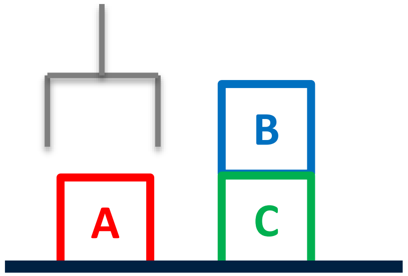
The Sussmann Anomaly – Linear Solution 1

(on B C)
- Pickup (B)
- Stack (B, C)

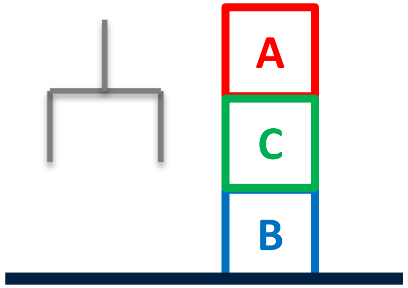
(on A B)
- Unstack (B, C)
- Putdown (B)
- Unstack (C, A)
- Putdown (C)
- Pickup(A)
- Stack (A, B)

(on B C)
- Unstack (A, B)
- Putdown (A)
- Pickup (B)
- Stack (B, C)

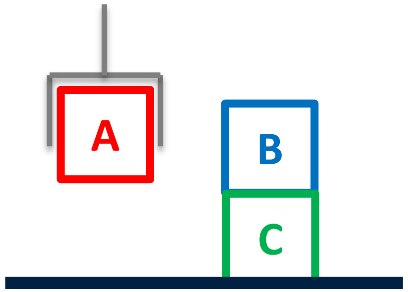
(on A B)
- Pickup (A)
- Stack (A,B)



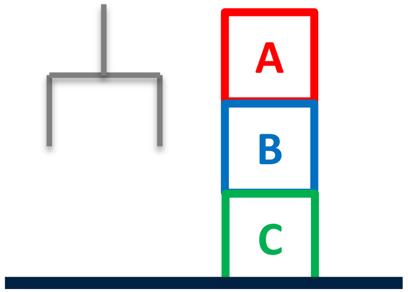
State



Goal



State

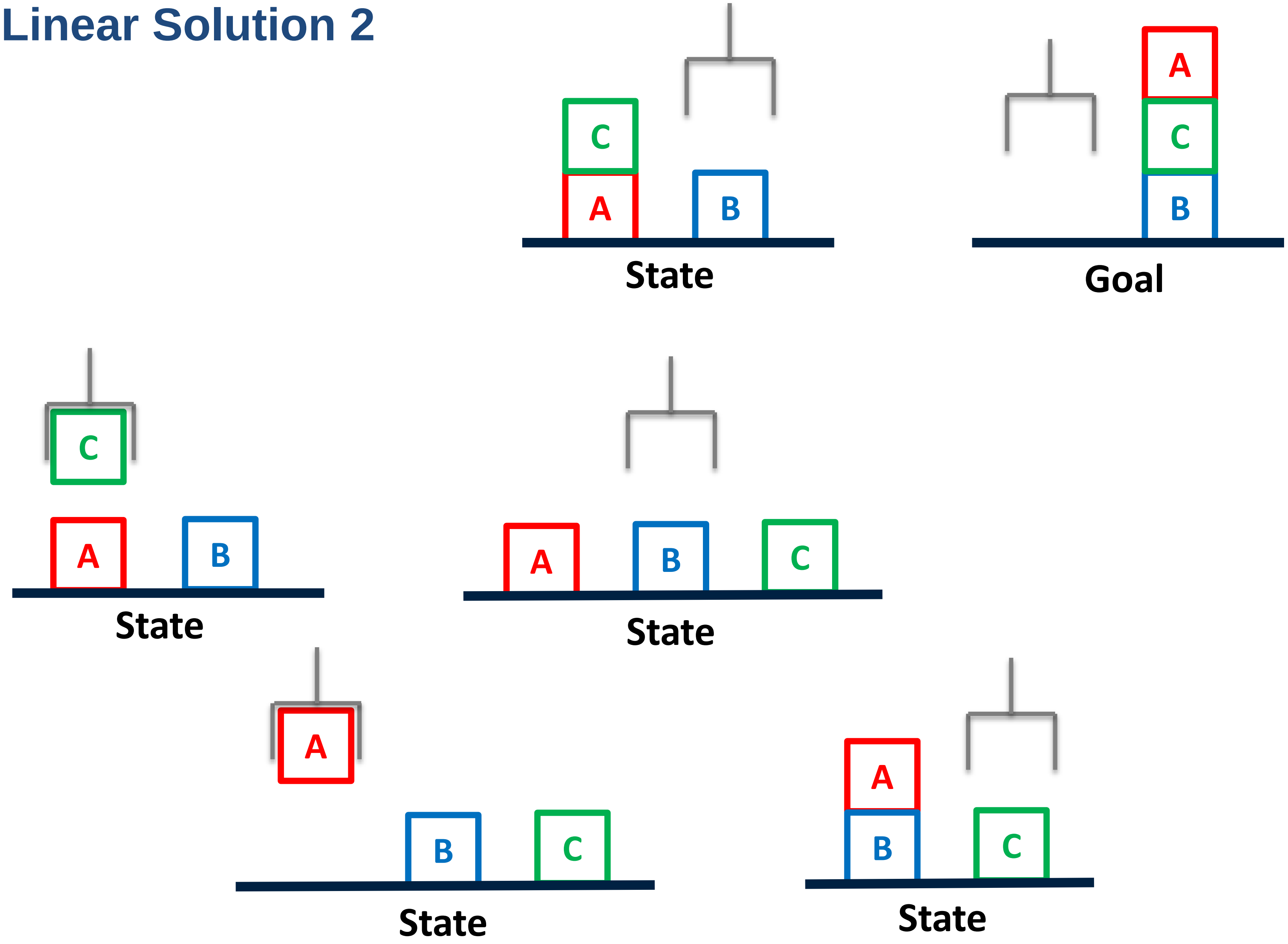


State

The Sussmann Anomaly – Linear Solution 2

(on A B)

- Unstack (C, A)
- Putdown (C)
- Pickup(A)
- Stack (A, B)



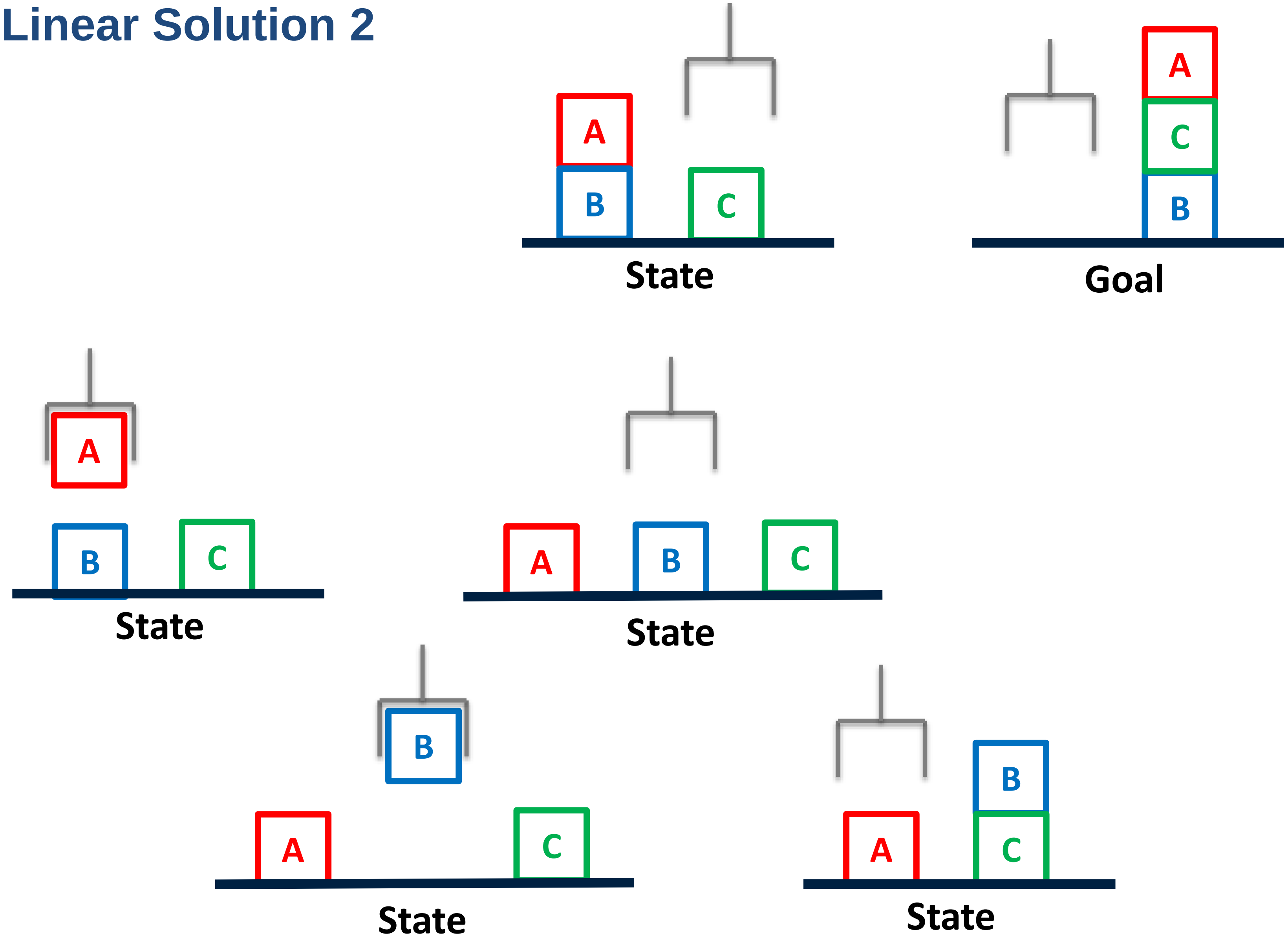
The Sussmann Anomaly – Linear Solution 2

(on A B)

- Unstack (C, A)
- Putdown (C)
- Pickup(A)
- Stack (A, B)

(on B C)

- Unstack (A, B)
- Putdown (A)
- Pickup (B)
- Stack (B, C)

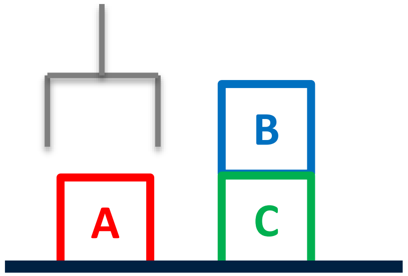


The Sussmann Anomaly – Linear Solution 2

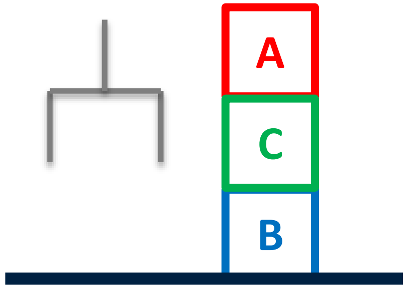
(on A B)
- Unstack (C, A)
- Putdown (C)
- Pickup(A)
- Stack (A, B)

(on B C)
- Unstack (A, B)
- Putdown (A)
- Pickup (B)
- Stack (B, C)

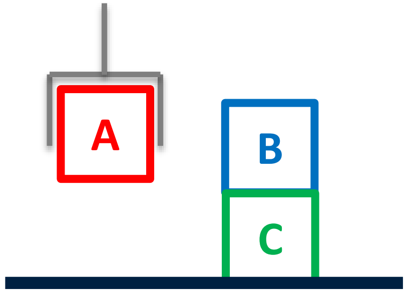
(on A B)
- Pickup (A)
- Stack (A,B)



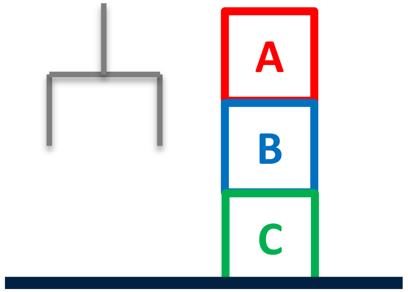
State



Goal



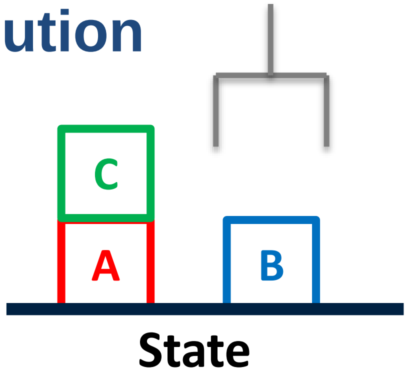
State

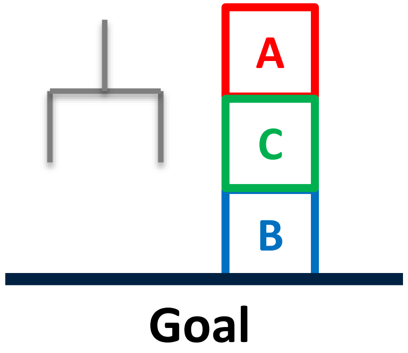


State

Is it Optimal? Can we do it with less actions?

The Sussmann Anomaly: Non Linear (Optimal) Solution





(on A B)

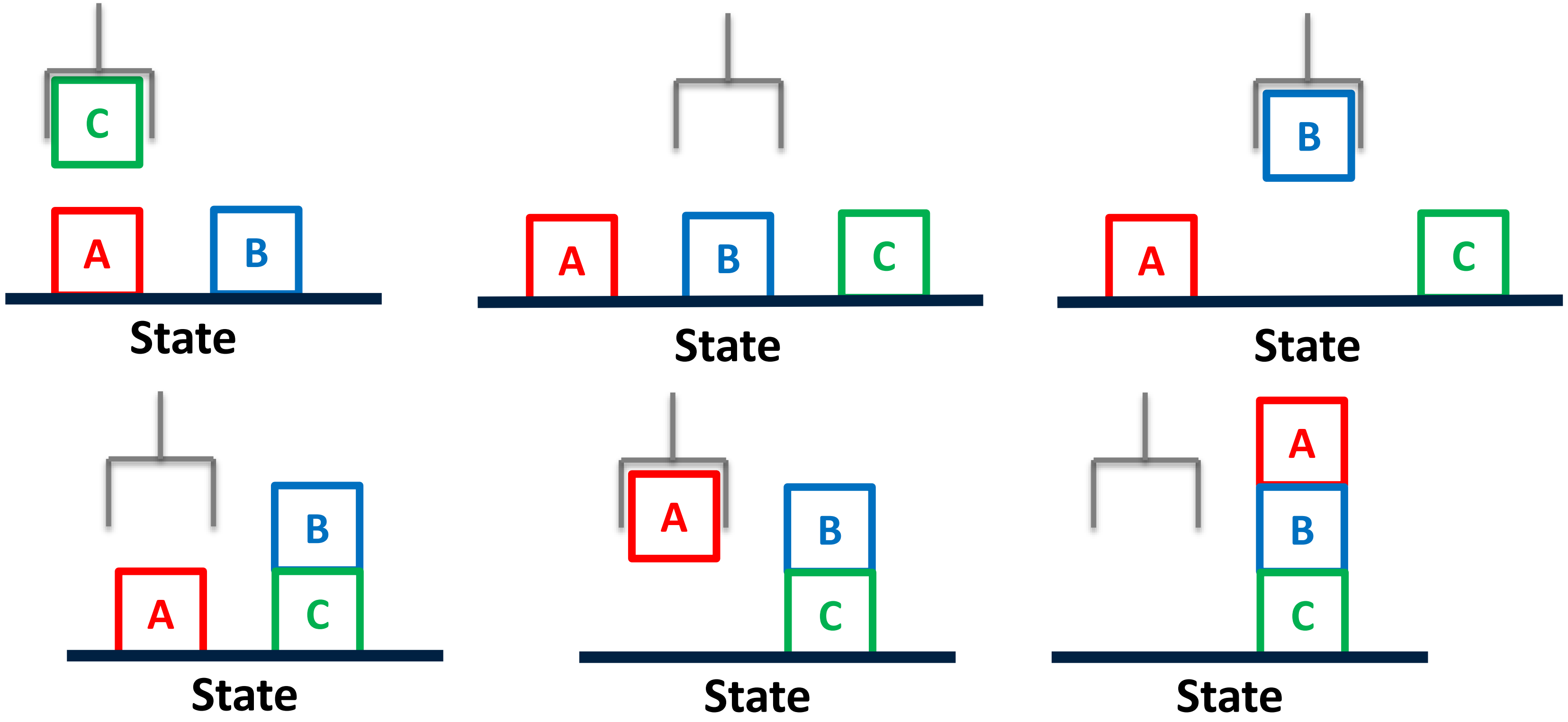
- Unstack (C, A)
- Putdown (C)

(on B C)

- Pickup (B)
- Stack (B, C)

(on A B)

- Pickup(A)
- Stack(A, B)



Linear Planning and the Goal Stack

Advantages

- Reduced search space, since goals are solved one at a time, and not all possible goal orderings are considered
- Advantageous if goals are (mainly) independent
- Linear planning is sound

What about completeness?

Disadvantages

- Linear planning may produce suboptimal solutions (based on the number of operators in the plan)
- Planner's efficiency is sensitive to goal orderings
    - Control knowledge for the “right” ordering
    - Random restarts
    - Iterative deepening

One Way Rocket (Veloso '89)

```
(OPERATOR LOAD-ROCKET
 :preconds
  ?roc ROCKET
  ?obj OBJECT
  ?loc LOCATION
 (and (at ?obj ?loc)
      (at ?roc ?loc))
 :effects
  add (inside ?obj ?roc)
  del (at ?obj ?loc))
```

```
(OPERATOR UNLOAD-ROCKET
 :preconds
  ?roc ROCKET
  ?obj OBJECT
  ?loc LOCATION
 (and (inside ?obj ?roc)
      (at ?roc ?loc))
 :effects
  add (at ?obj ?loc)
  del (inside ?obj ?roc))
```

```
(OPERATOR MOVE-ROCKET
 :preconds
  ?roc ROCKET
  ?from-l LOCATION
  ?to-l LOCATION
 (and (at ?roc ?from-l)
      (has-fuel ?roc))
 :effects
  add (at ?roc ?to-l)
  del (at ?roc ?from-l)
  del (has-fuel ?roc))
```

```
Initial state:
        (at obj1 locA)
        (at obj2 locA)
        (at ROCKET locA)
        (has-fuel ROCKET)
```

```
Goal statement:
        (and
          (at obj1 locB)
          (at obj2 locB))
```

Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA) (MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	*failure*

State Space Non Linear Planning

Extend linear planning:

- From stack to set of goals
- Include in the search space all possible interleaving of goals

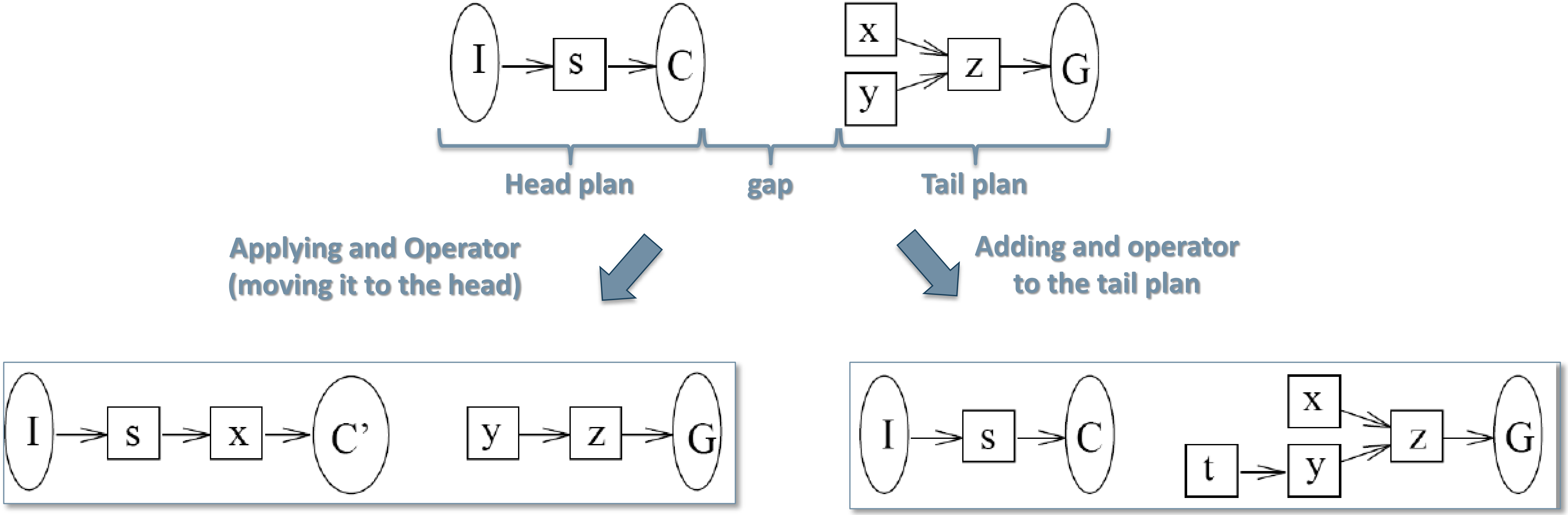
State-space nonlinear planning is complete

Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA)
(at obj2 locB)	(LOAD-ROCKET obj2 locA)
(at obj1 locB)	(MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	(UNLOAD-ROCKET obj1 locB)

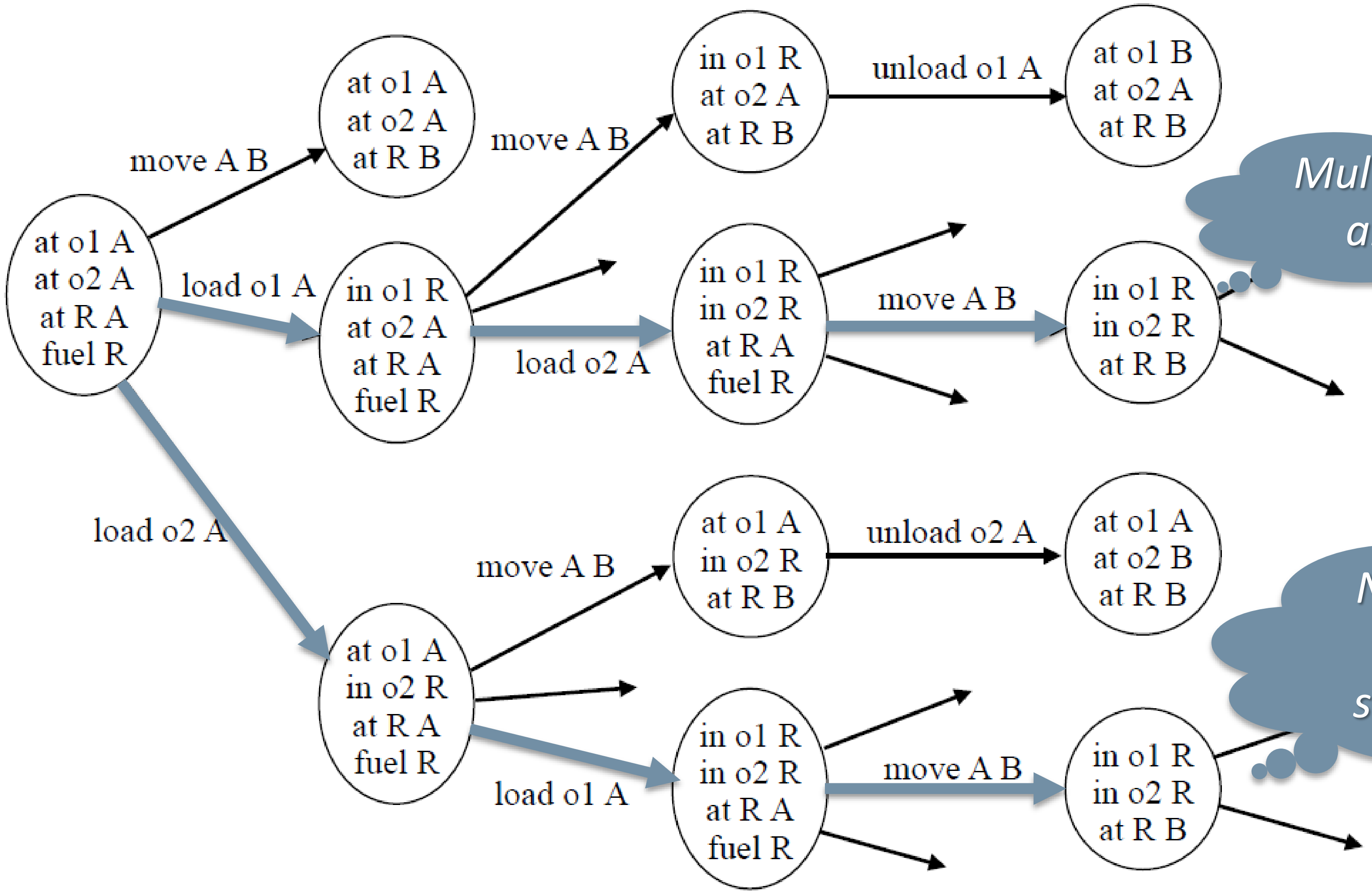
Prodigy4.0 (Veloso et al. 90)

1. Terminate if the goal statement is satisfied in the current state. Initially the set of applicable relevant operators is empty.
2. Compute the SET of pending goals G, and the SET of applicable relevant operators A.
   - A goal is pending if it is a precondition, not satisfied in the current state, of a relevant operator already in the plan.
   - A relevant operator is applicable when all its preconditions are satisfied in the state.
3. Choose a pending goal G in G or choose a relevant applicable operator A in A.
4. If the pending goal G has been chosen, then
   - Expand goal G, i.e., get the set O of relevant instantiated operators that could achieve G,
   - Choose an operator O from O, as a relevant operator for goal G.
   - Go to step 1.
5. If a relevant operator A has been selected as directly applicable, then
   - Apply A,
   - Go to step 1.

Prodigy4.0 Search Representation



After all, it is all about graph exploration



Multiple solutions are possible.

No need to explore the whole graph, but you should be able to do it!

Planning issues

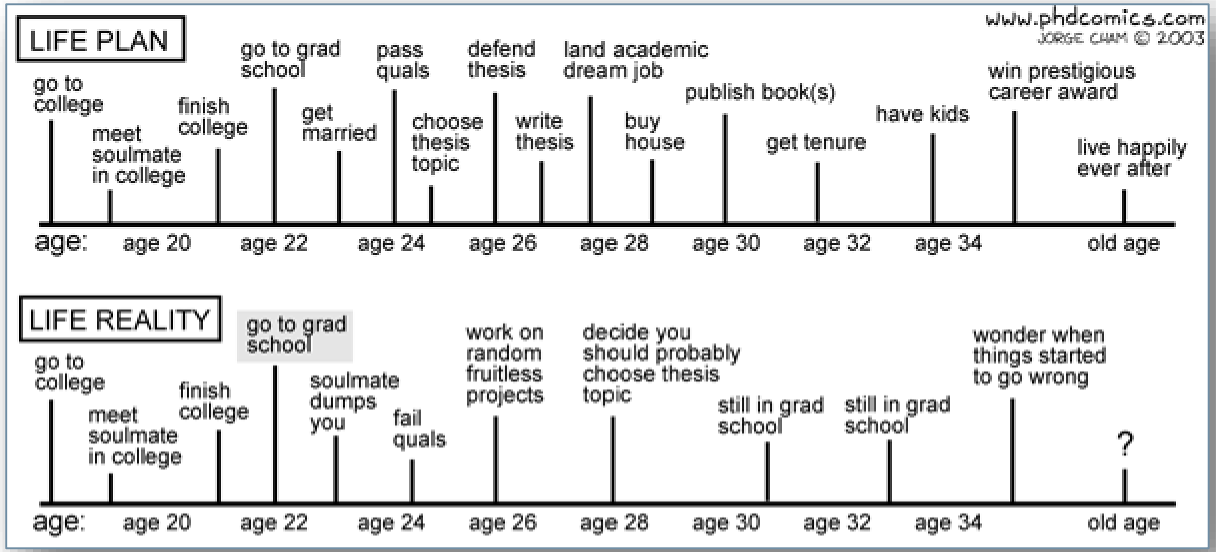
State representation

- The frame problem
- The "choice" of predicates
  (e.g., On-table (x), On (x, table), On-table-A, On-table-B,…)

Action representation

- Many alternative definitions
- Reduce to "needed" definition
- Conditional effects
- Uncertainty
- Quantification
- Functions

Generation – planning algorithm(S)



Wrap-up slide on “Planning and Plan Generation”

What should remain from this lecture?

- Planning: selecting one sequence of actions (operators) that transform (apply to) an initial state to a final state where the goal statement is true.
- Means-ends analysis: identify and reduce, as soon as possible, differences between state and goals.
- Linear planning: backward chaining with means-ends analysis using a stack of goals, potentially efficient, possibly unoptimal, incomplete; GPS
- Nonlinear planning with means-ends analysis: backward chaining using a set of goals; reason about when “to reduce the differences;” Prodigy4.0.

References

- S. Russell, P. Norvig. «Artificial Intelligence: A Modern Approach». Chapter 11: Planning, pages 375-416.Pearson, 2010.