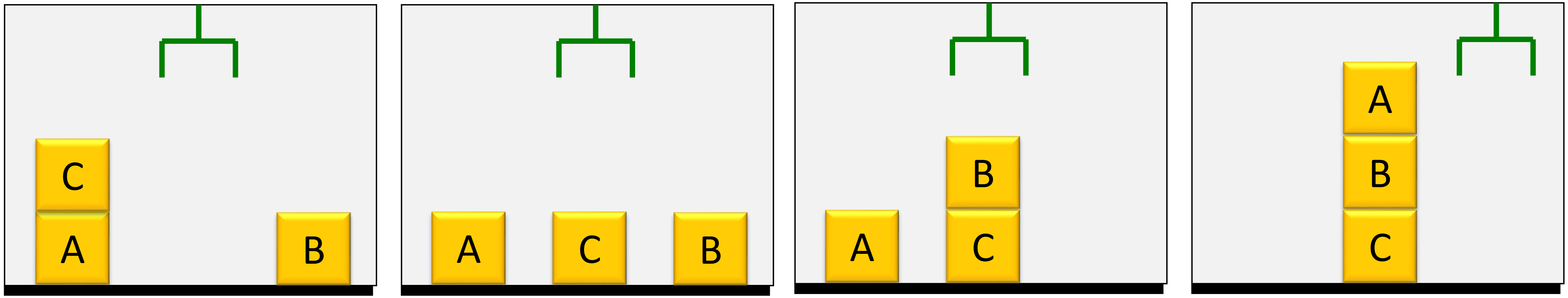


Artificial Intelligence

Planning

Where logic-based representation of knowledge makes search problems more interesting

-Anish Raj, Dept. CE and IT, COEP

Outline

Introduction to Planning

Block World Environment
State Representation
Action Representation

Situation Calculus
State Representation
Action Representation
Goal Representation
Situation Planning

STRIPS
STRIP Representation
Forward Planning
Backward Planning

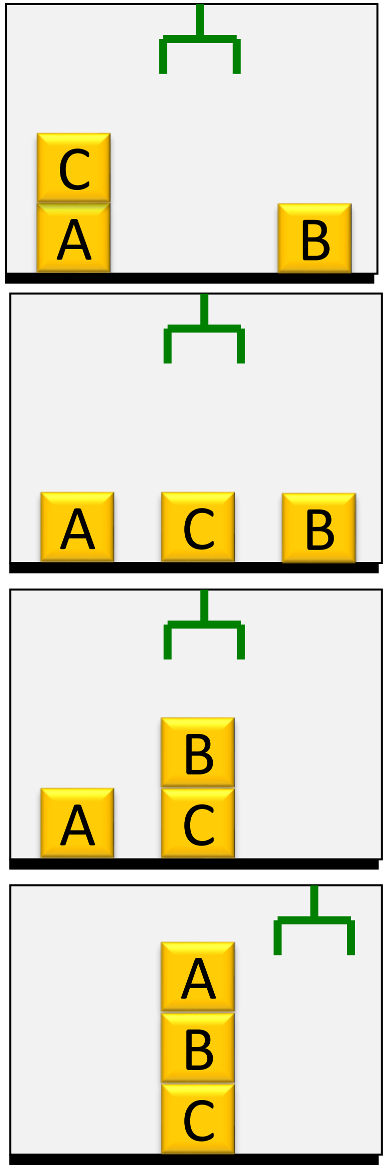
Goal Stack Planning
Sussman Anomaly

Nonlinear Planning
TWEAK Planner

Hierarchical Planning

Planning

- The world is dynamic:
  - What is true now may not be true tomorrow
  - Changes in the world may be triggered by our activities
- Problems:
  - Logic (FOL) as we had it referred to a static world. How to represent the change in the FOL ?
  - How to represent actions we can use to change the world?
- Planning problem:
  - Find a sequence of actions that achieves some goal in this complex world



Components for Planning

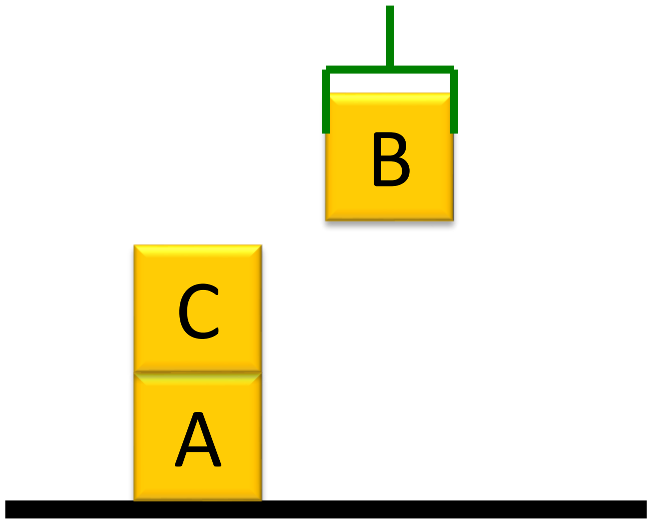
- Classical planning model is a tuple S = $\langle S, s_0, S_G, A, f, c \rangle$, where
- Finite and discrete state space S
- A known initial state $s_0 \in S$
- A set $S_G \subseteq S$ of goal states
- Actions $A(s) \subseteq A$ applicable in each $s \in S$
- A deterministic transition function

  $s' = f(a, s)$ for $a \in A(s)$
- Non-negative action costs $c(a, s)$

- Solution: sequence of applicable actions that maps s_0 into S_G

An Example Domain: The Block World

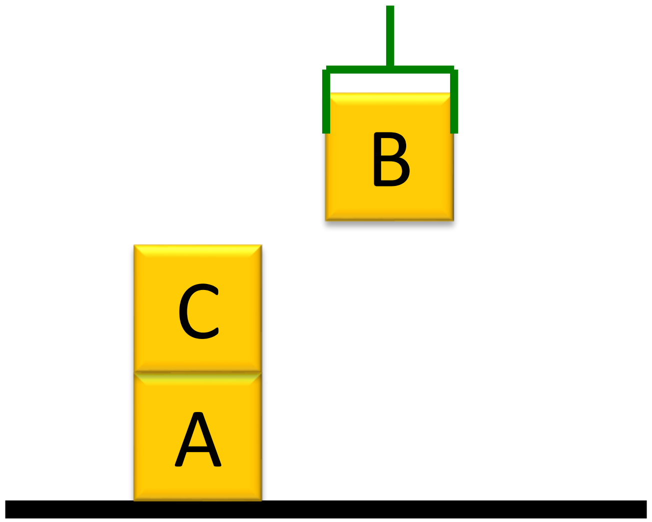
- There is flat surface and square blocks (all are same size)
- Blocks can be placed over the flat surface.
- Blocks can be stacked one upon another.
- There is a robot arm that can manipulate the blocks.



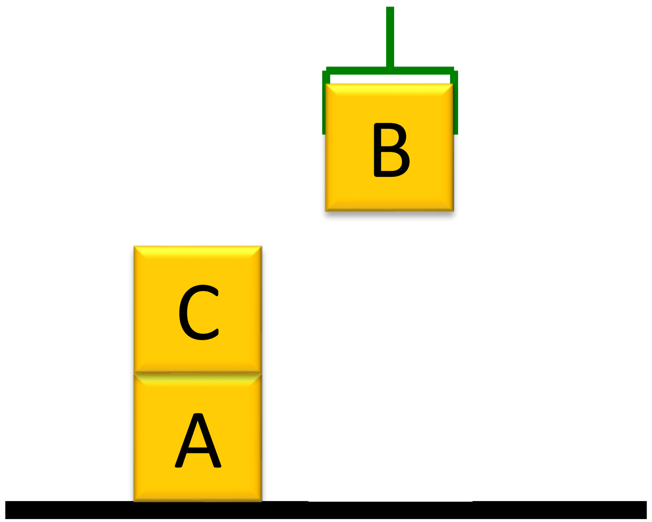
- Represent the environment by predicates:
  - ON(C, A): Block C is on Block A
  - ONTABLE(A): Block A is on the table
  - CLEAR(C): There is nothing on top of Block C
  - HOLDING(B): The arm is holding Block B
  - HANDEMPTY: The arm is holding nothing

An Example Domain: The Block World

- There is flat surface and square blocks (all are same size)
- Blocks can be placed over the flat surface.
- Blocks can be stacked one upon another.
- There is a robot arm that can manipulate the blocks.

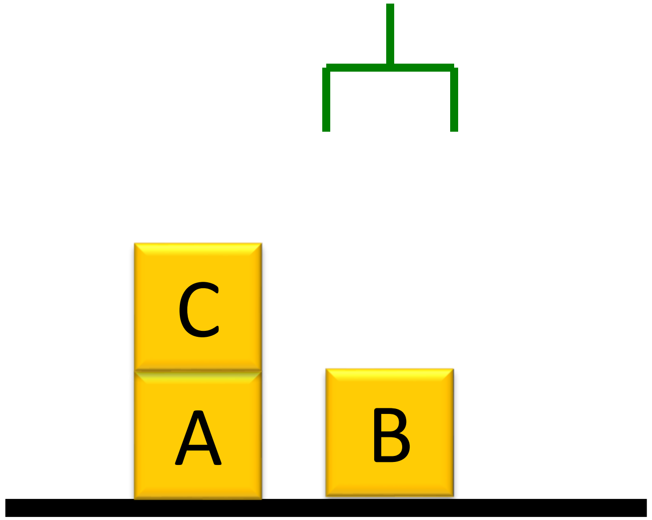


- Action represented as:
  - PUTDOWN(B): Put block B down on the table
  - PICKUP(B): Pickup block B from the table and hold it
  - UNSTACK(C, A): Pickup block C from its current position on block A
  - STACK(C, A): Place block C on block A

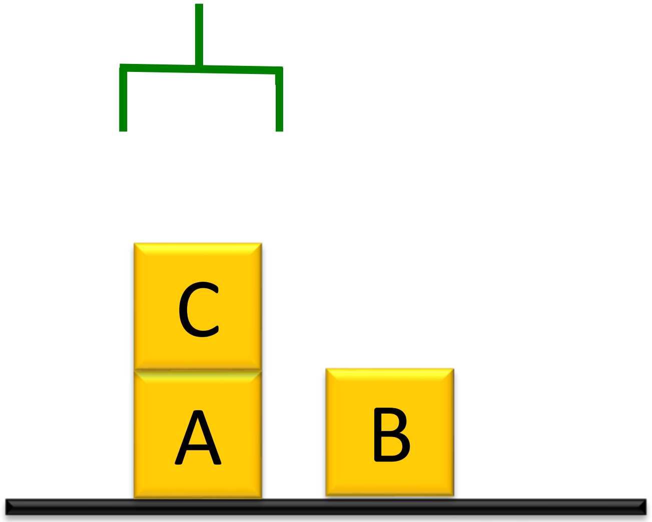


An Example Domain: The Block World

- There is flat surface and square blocks (all are same size)
- Blocks can be placed over the flat surface.
- Blocks can be stacked one upon another.
- There is a robot arm that can manipulate the blocks.



- Action represented as:
  - PUTDOWN(B): Put block B down on the table
  - PICKUP(B): Pickup block B from the table and hold it
  - UNSTACK(C, A): Pickup block C from its current position on block A
  - STACK(C, A): Place block C on block A



Planning Methods

Situation calculus
(extends FOL)

State space search
(STRIPS - restricted FOL)

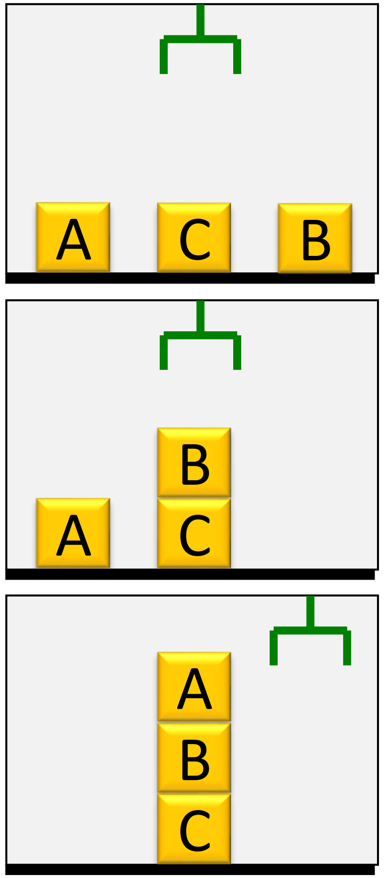
Methods for modeling and solving a planning problem

Plan-based search
(for STRIPS)

GRAPHPLAN
propositional languages

Situation Calculus

- Provides a framework for representing change, actions and reasoning about them
- Situation calculus
  - Based on first-order logic
  - A situation variable models new states of the world
  - Action objects model activities
  - Uses inference methods developed for FOL to do the reasoning
- The world is described by
  - Sequences of **situations** of the current state
  - Changes from one situation to another are **caused by actions**
- The situation calculus allows us to
  - Describe the initial state and a goal state
  - Build the KB that describes the effect of actions (operators)
  - Prove that the KB and the initial state lead to a goal state
    - extracts a plan as side-effect of the proof

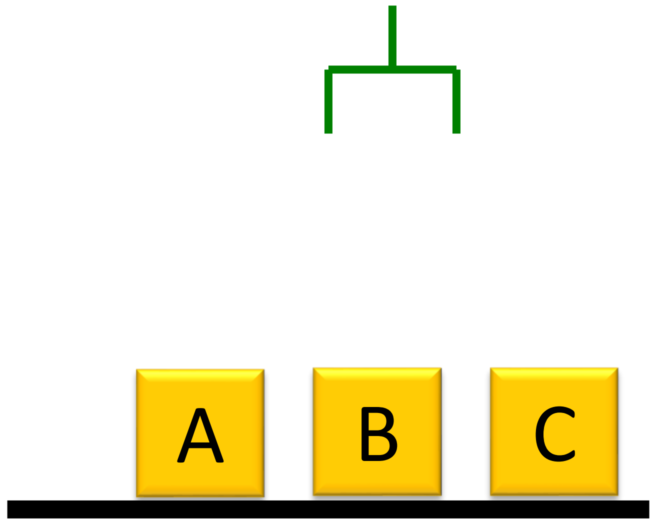


The Language of Planning Problems

Representation of states

- Decompose the world into logical conditions and represent a state as a **list of literals**
- Must be ground and function-free

- Special variables:
  - **s**: objects of type situation



ON(A, TABLE, s0)
ON(B, TABLE, s0)
ON(C, TABLE, s0)
CLEAR(A, s0)
CLEAR(B, s0)
CLEAR(C, s0)
CLEAR(TABLE, s0)

The Language of Planning Problems

Representation of actions

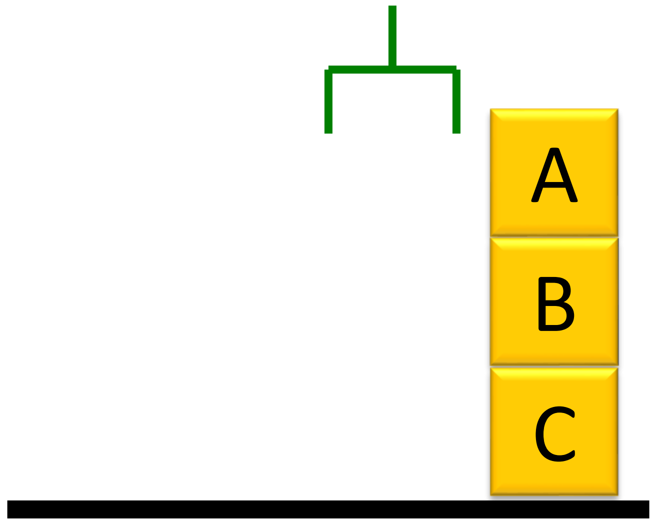
- Action functions: return actions.
  - Move(A, TABLE, B) represents a move action
  - Move(x, y ,z) represents an action schema
- Special variables:
  - **a**: objects of type action
- Two special function symbols of type situation
  - **DO(a, s)**: denotes the situation obtained after performing an action a in situation s
- Situation-dependent functions and relations (also called **fluents**)
  - Relation: On(x, y, s) – object **x** is on object **y** in situation **s**
  - Function: Above(x, s) – object that is above **x** in situation **s**

The Language of Planning Problems

Representation of goals

- Goal is a partially specified state
- Represented as a list of ground literals
- State ***s*** satisfies goal ***g*** if ***s*** contains all the atoms in ***g*** (and possibly others)

- **Note:** It is not necessary that the goal describes all relations
  - Clear(A, g)

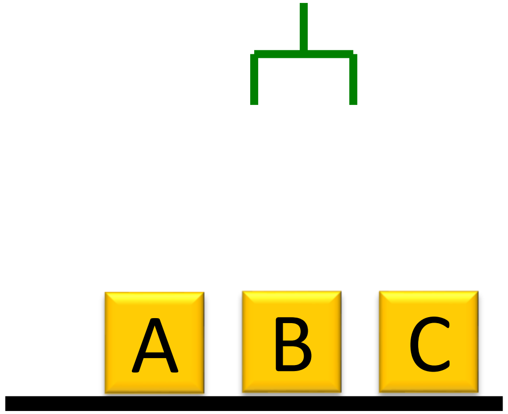


ON(A, B, g)
ON(B, C, g)
ON(C, TABLE, g)

The Language of Planning Problems

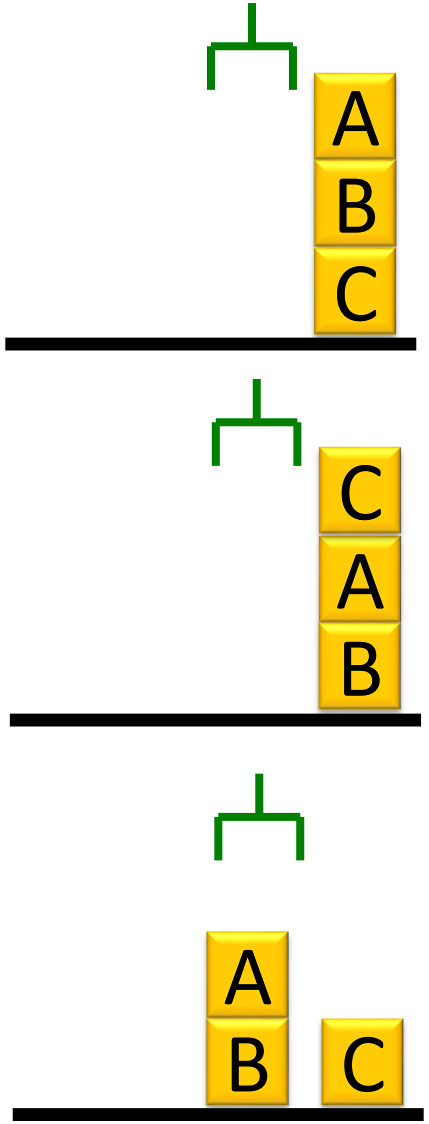
Representation of goals

- Assume a simpler goal ON(A,B, g)



ON(A, TABLE, s0)
ON(B, TABLE, s0)
ON(C, TABLE, s0)
CLEAR(A, s0)
CLEAR(B, s0)
CLEAR(C, s0)

3 Possible Goal Configuration:
ON(A, B, g)



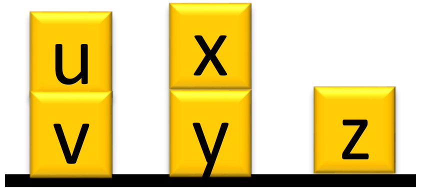
Knowledge base: Axioms

- Knowledge base needed to support the reasoning:
  - Must represent changes in the world due to actions.

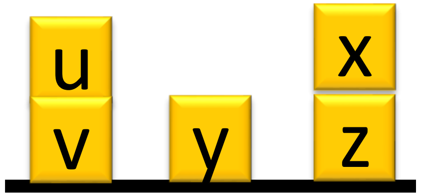
- Two types of axioms:
  - Effect axioms
    - Changes in situations that result from actions
  - Frame axioms
    - Things preserved from the previous situation

Blocks World: Effect axioms

- Moving x from y to z.   MOVE (x, y, z)



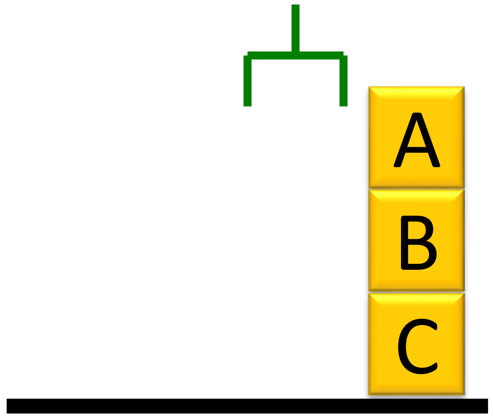
- Effect axioms:
- Effect of move changes on ON relations
  - $ON(x, y, s) \land CLEAR(x, s) \land CLEAR(z, s) \rightarrow ON(x, z, DO\ (MOVE\ (x, y, z), s))$
  - $ON(x, y, s) \land CLEAR(x, s) \land CLEAR(z, s) \rightarrow \neg ON(x, y, DO(MOVE\ (x, y, z), s))$



- Frame axioms:
- ON relation:
  - $ON(u, v, s) \land (u \neq x) \land (v \neq y) \rightarrow ON(u, v, DO\ (MOVE\ (x, y, z), s))$

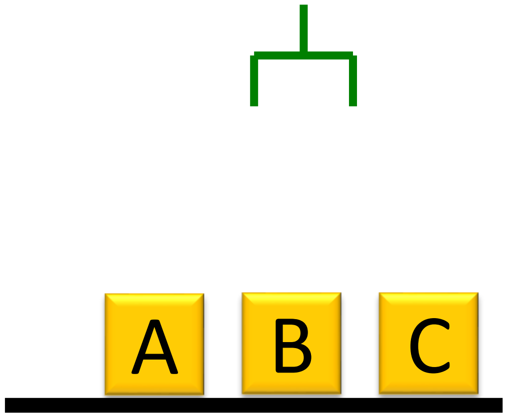
Planning in Situation Calculus

- Planning problem:
  - Find a sequence of actions that lead to a goal
- Planning in situation calculus is **converted to the theorem proving problem**
  - Goal state: $\exists s\ ON(A, B, s) \land ON(B, C, s) \land On(C, TABLE, s)$
- Possible inference approaches:
  - Inference rule approach
  - Resolution refutation
- Plan (solution) is a byproduct of theorem proving.
  - Example: blocks world



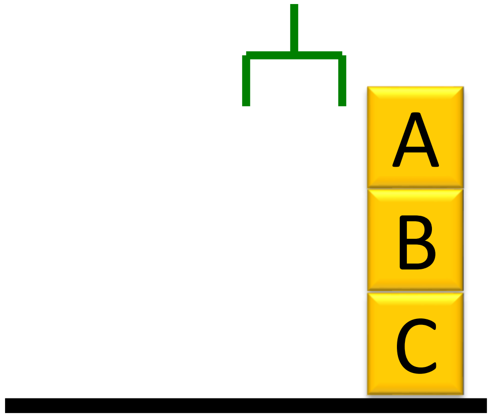
Planning in Block World

Initial State



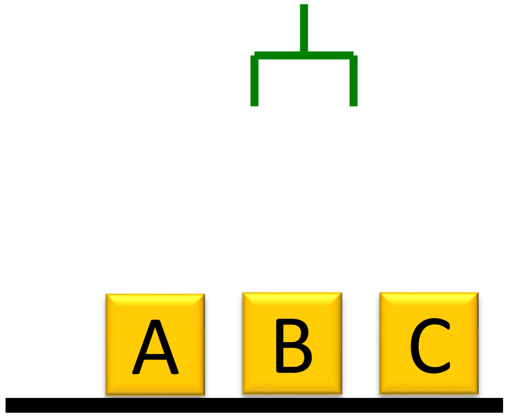
ON(A, TABLE, s0)
ON(B, TABLE, s0)
ON(C, TABLE, s0)
CLEAR(A, s0)
CLEAR(B, s0)
CLEAR(C, s0)

Goal State



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

Planning in Block World

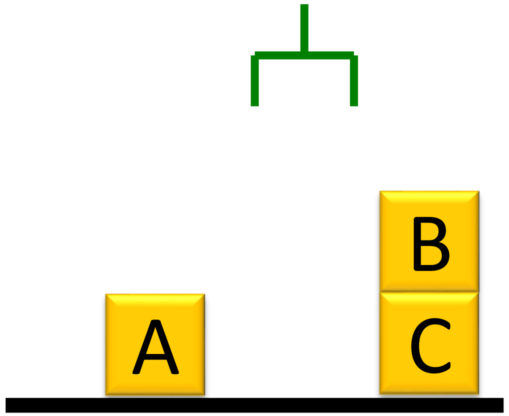


s0

ON(A, TABLE, s0)
ON(B, TABLE, s0)
ON(C, TABLE, s0)
CLEAR(A, s0)
CLEAR(B, s0)
CLEAR(C, s0)

Action

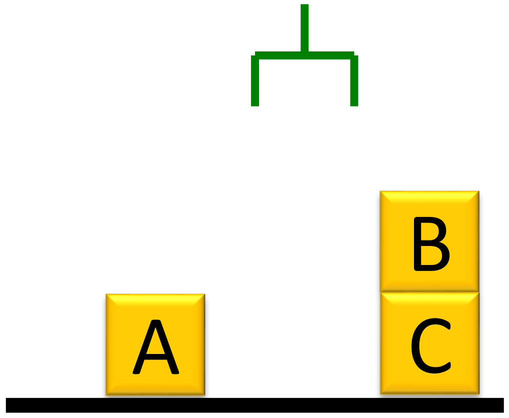
MOVE(B, TABLE, C)
S1 = DO(MOVE(B, TABLE, C), s0)



s1

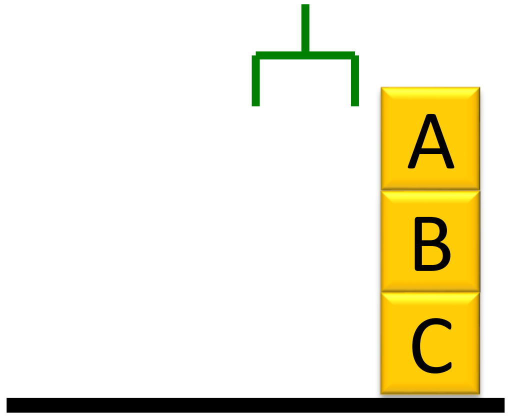
ON(A, TABLE, s1)
ON(B, C, s1)
¬ ON(B, TABLE, s1)
ON(C, TABLE, s1)
CLEAR(A, s1)
CLEAR(B, s1)
¬ CLEAR(C, s1)

Planning in Block World



s1

Action



s2

ON(A, TABLE, s1)
ON(B, C, s1)
¬ ON(B, TABLE, s1)
ON(C, TABLE, s1)
CLEAR(A, s1)
CLEAR(B, s1)
¬ CLEAR(C, s1)

MOVE(A, TABLE, B)
s2 = DO(MOVE(A, TABLE, B), s1)

ON(A, B, s2)
ON(B, C, s2)
ON(C, TABLE, s2)
¬ ON(A, TABLE, s2)
¬ ON(B, TABLE, s2)
CLEAR(A, s2)
¬ CLEAR(B, s1)
¬ CLEAR(C, s1)

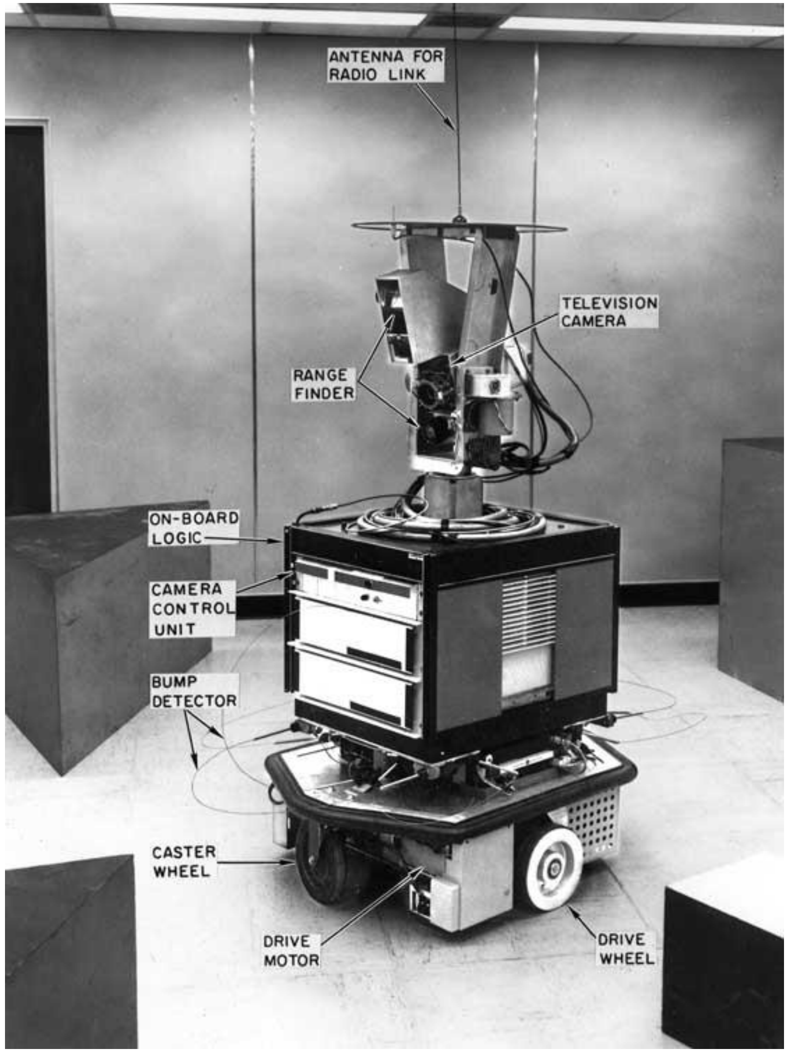
Planning in Situation Calculus

- Planning problem:
  - Find a sequence of actions that lead to a goal
  - Planning in situation calculus is converted to theorem proving.
- Problems with situation calculus:
  - Large search space
  - Large number of axioms to be defined for one action
  - Proof may not lead to the best (shortest) plan.
- Solution →
- Complex state description and local action effects:
  - Avoid the enumeration and inference of every state component, focus on changes
- Many possible actions:
  - Apply actions that make progress towards the goal
  - Understand what the effect of actions is and reason with the consequences
- Sequences of actions in the plan can be too long
  - Many goals consists of independent or nearly independent sub-goals
  - Allow goal decomposition & divide and conquer strategies

STRIPS

- STRIPS: a simple, still reasonably expressive planning language based on logic
  1. Represent planning problems in STRIPS
  2. Planning methods
  3. Extensions of STRIPS
- Like programming
- Knowledge representation is still an art

SHAKEY the Robot



Language of Classical Planning: STRIPS

- A STRIPS Planning task is 5-tuple = $\langle F, O, c, I, G \rangle$:
  - F : finite set of atoms (boolean variables)
  - O: finite set of operators (actions) of form $\langle Add, Del, Pre \rangle$
    (Add/Delete/Preconditions; subsets of atoms)
  - $c : O \rightarrow \mathbb{R}^{0+}$ captures operator cost
  - I: initial state (subset of atoms)
  - G: goal description (subset of atoms)

From Language to Models

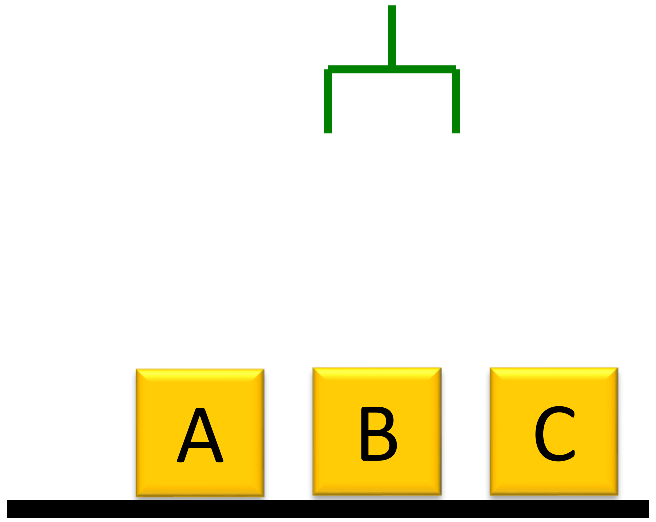
- A STRIPS Planning task $\Pi = \langle F, O, c, I, G \rangle$ determines state model $S(\Pi)$ where
  - the states $s \in S$ are collections of atoms from F
  - the initial state s_0 is I
  - the goal states s are such that $G \subseteq s$
  - the actions a in $A(s)$ are ops in O s.t. $\mathrm{Pre}(a) \subseteq s$
  - the next state is $s' = s - \mathrm{Del}(a) + \mathrm{Add}(a)$
  - action costs $c(a, s) = c(a)$

- Solutions of $S(\Pi)$ are plans of Π

STRIP: State

- Decompose the world into logical conditions and represent a state as a **conjunction of positive literals**
  - No negated proposition, such as $\neg$Clean(A)
- Closed-world assumption:
  - Every proposition that is not listed in a state is false in that state
- No “or” connective
  - Such as On(A,B)$\vee$On(B,C)
  - No variable, e.g., $\exists$x Clean(x)

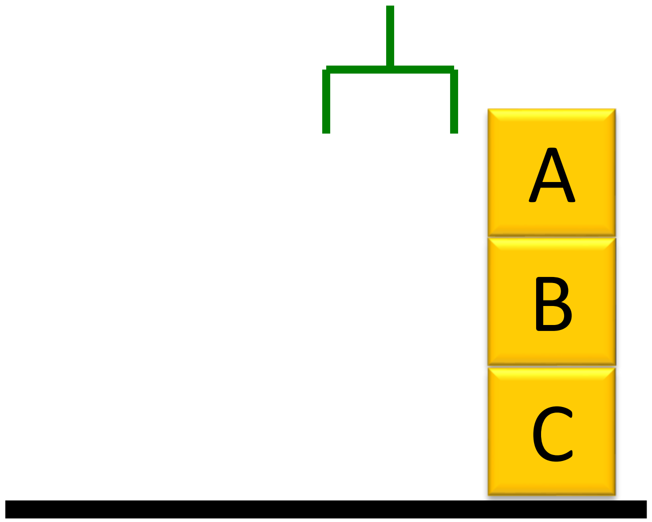
No uncertainty



Block(A) ∧ On(A, TABLE) ∧
Block(B) ∧ On(B, TABLE) ∧
Block(C) ∧ On(C, TABLE) ∧
Clear(A) ∧ Clear(B) ∧
Clear(C) ∧ Handempty

STRIP: Goal State

- **Conjunction of positive literals**
- Goal is a partially specified state
- Represented as a conjunction of ground literals
- A goal G is achieved in a state S if all
  the propositions in G (called sub-goals) are also in S

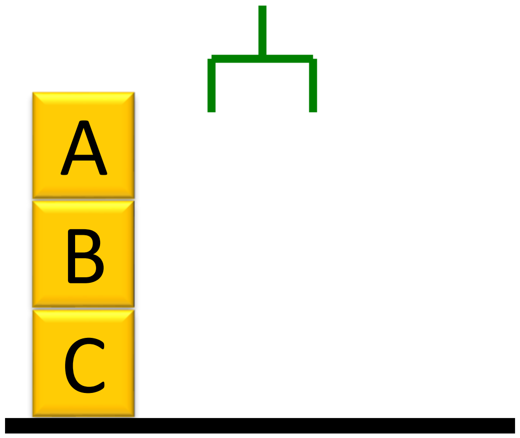


- **Note:** It is not necessary that the goal describes all relations
  - BLOCK(A)

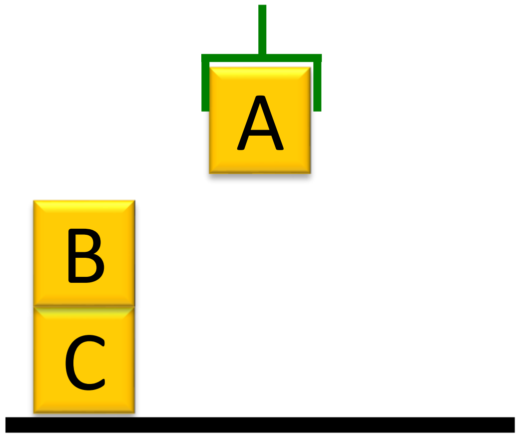
On(A, B)
On(B, C)
On(C, TABLE)
Clear(C)

STRIP: Action

- Specified in terms of the preconditions that must hold before it can be executed and the effects that ensue when it is executed
- Action functions (Operator):
  - Unstack(A, B)

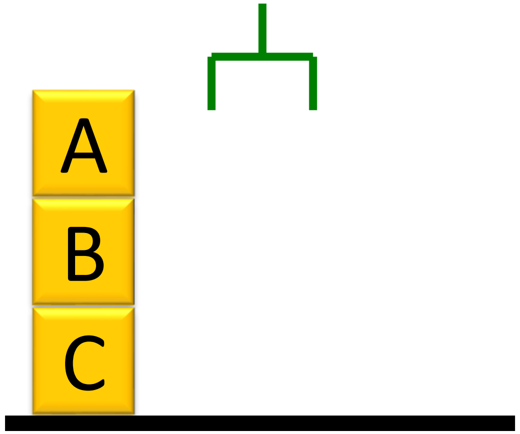


- Preconditions:
  - Block(A) $\wedge$ Block(B) $\wedge$ Clear(A) $\wedge$ On(A, B) $\wedge$ Handempty
- Effects: Two lists
  - **Delete list:** Handempty, Clear(A), On(A, B)
  - **Add list:** Holding(A), Clear(B)
- Everything else remains untouched (is preserved)

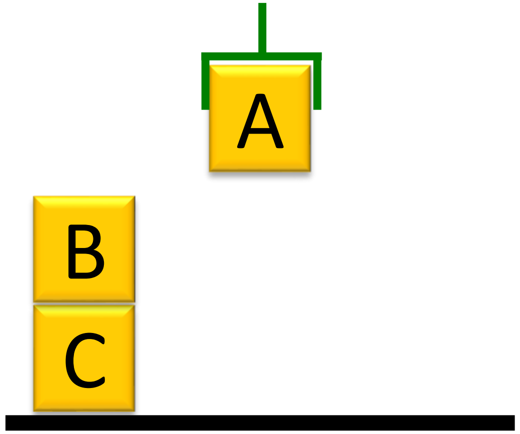


STRIP: Action

- Unstack(A, B)
  - **P**: Block(A) ∧ Block(B) ∧ Clear(A) ∧ On(A, B) ∧ Handempty
  - **D:** On(A, B), Clear(A), Handempty
  - **A:** Clear(B), Holding(A)



Block(A) ∧ Block(B) ∧ Block(C) ∧ On(C, TABLE) ∧ On(B, C) ∧ Clear(B) ∧ Holding(A)



the actions a in $A(s)$ are ops in O s.t. $\mathrm{Pre}(a) \subseteq s$
the next state is $s' = s - \mathrm{Del}(a) + \mathrm{Add}(a)$

STRIP: Actions

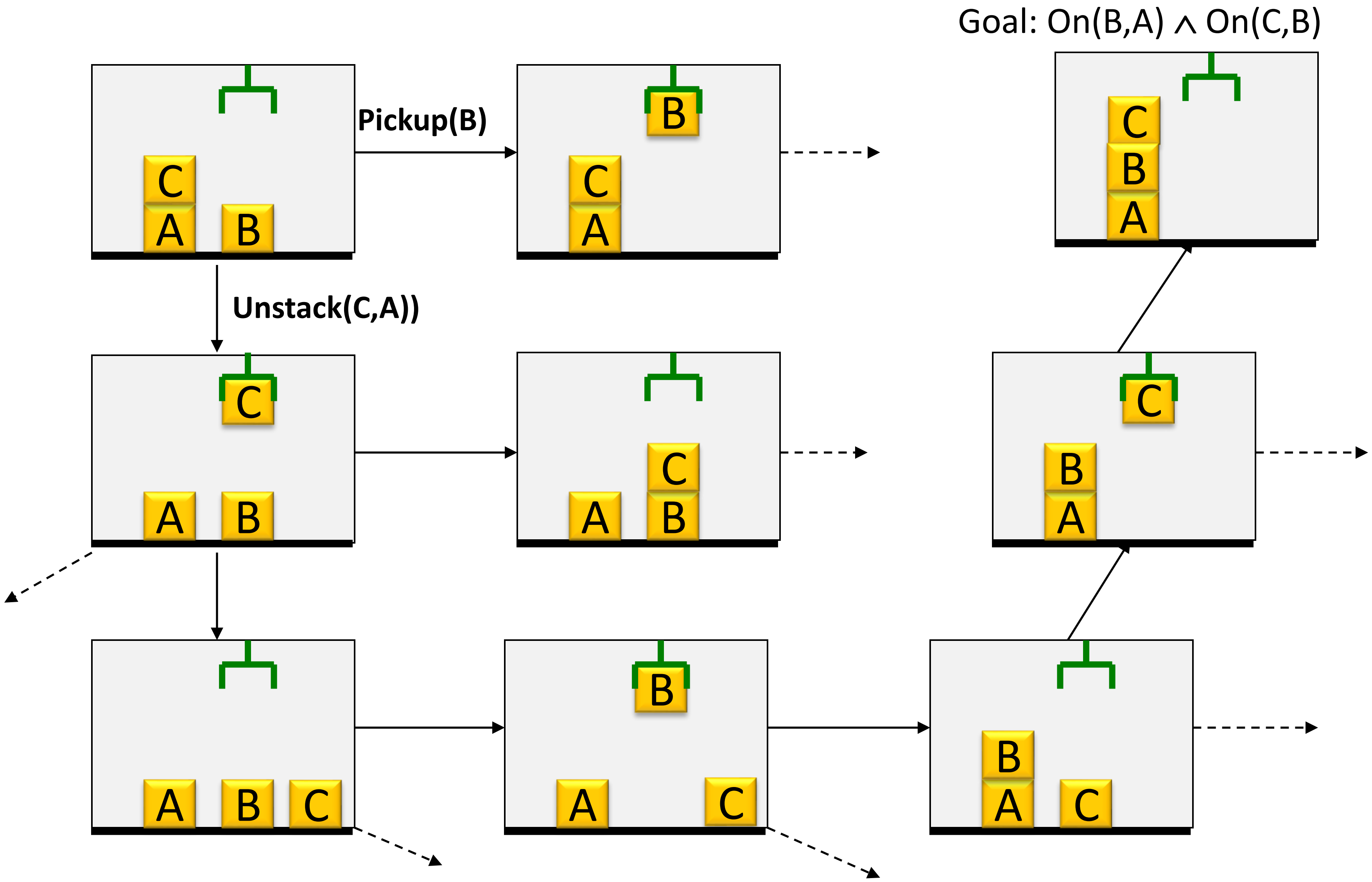
OPERATORS	PRECONDITION	DELETE	ADD
STACK(X,Y)	CLEAR(Y)∧ HOLDING(X)	CLEAR(Y) HOLDING(X)	ARMEMPTY ON(X,Y)
UNSTACK(X,Y)	ARMEMPTY∧ ON(X,Y)∧ CLEAR(X)	ARMEMPTY∧ ON(X,Y)	HOLDING(X) ∧CLEAR(Y)
PICKUP(X)	CLEAR(X)∧ ONTABLE(X)∧ ARMEMPTY	ONTABLE(X)∧ ARMEMPTY	HOLDING(X)
PUTDOWN(X)	HOLDING(X)	HOLDING(X)	ONTABLE(X)∧ ARMEMPTY

STRIPS Planning

- Objective
  - Find a sequence of operators (a plan) from the initial state to the state satisfying the goal

- Two approaches to build a plan
  - Forward state space search (goal progression)
    - Start from what is known in the initial state and apply operators in the order they are applied
  - Backward state space search (goal regression)
    - Start from the description of the goal and identify actions that help to reach the goal

Forward Planning (Goal Progression)



Need for an Accurate Heuristic

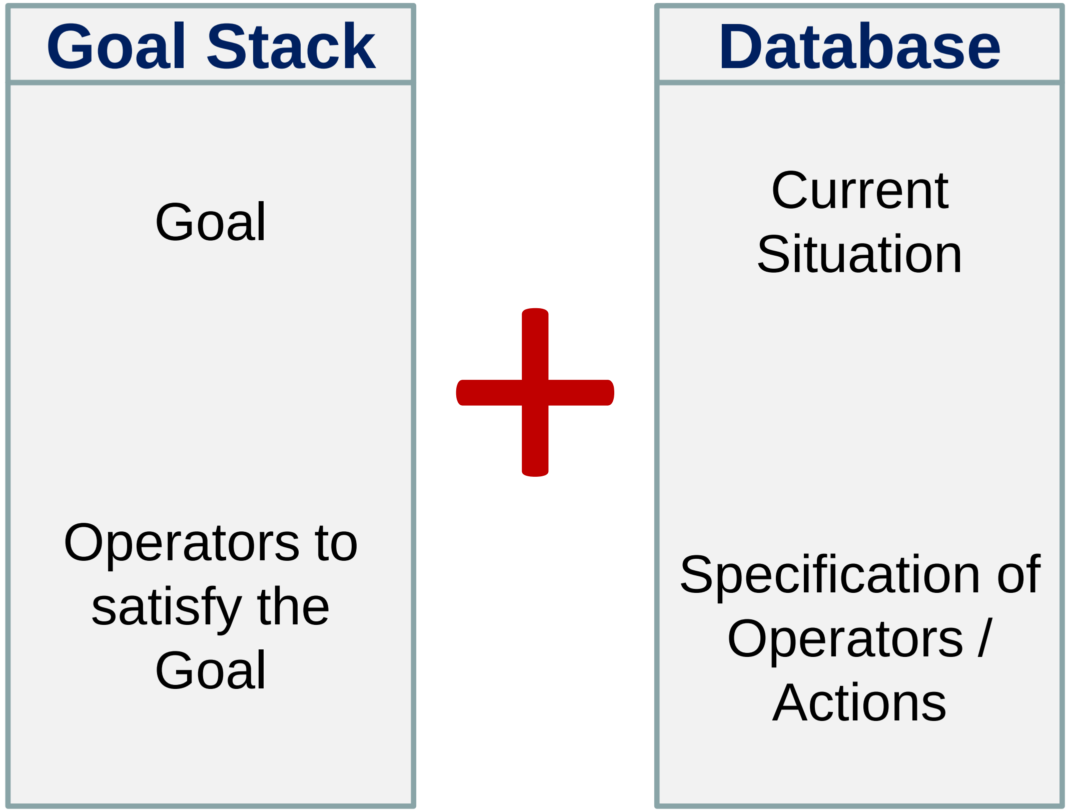
- Forward planning simply searches the space of world states from the initial to the goal state
- Imagine an agent with a large library of actions, whose goal is G, e.g., G = Have(Tea)
- In general, many actions are applicable to any given state, so the branching factor is huge
- In any given state, most applicable actions are irrelevant to reaching the goal Have(Tea)
- Fortunately, an accurate consistent heuristic can be computed using planning graphs
- How to determine which actions are relevant? How to use them?
- → **Backward planning**

STRIPS Goal Stack Planning

The original STRIPS system used a goal stack to control its search.

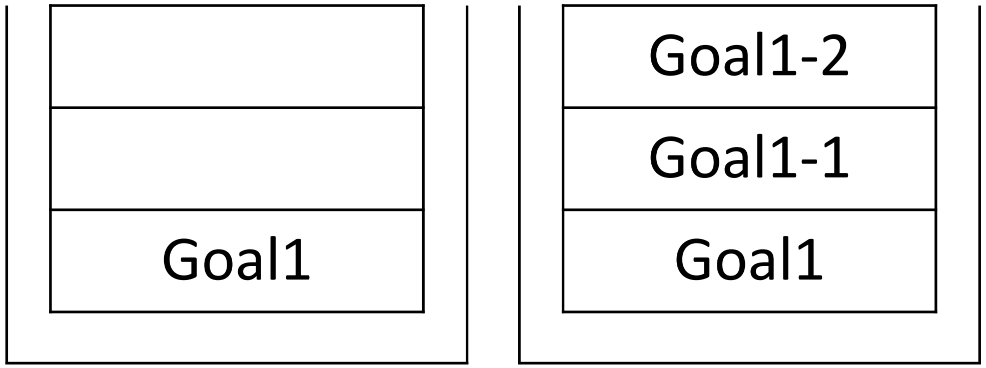
The system has

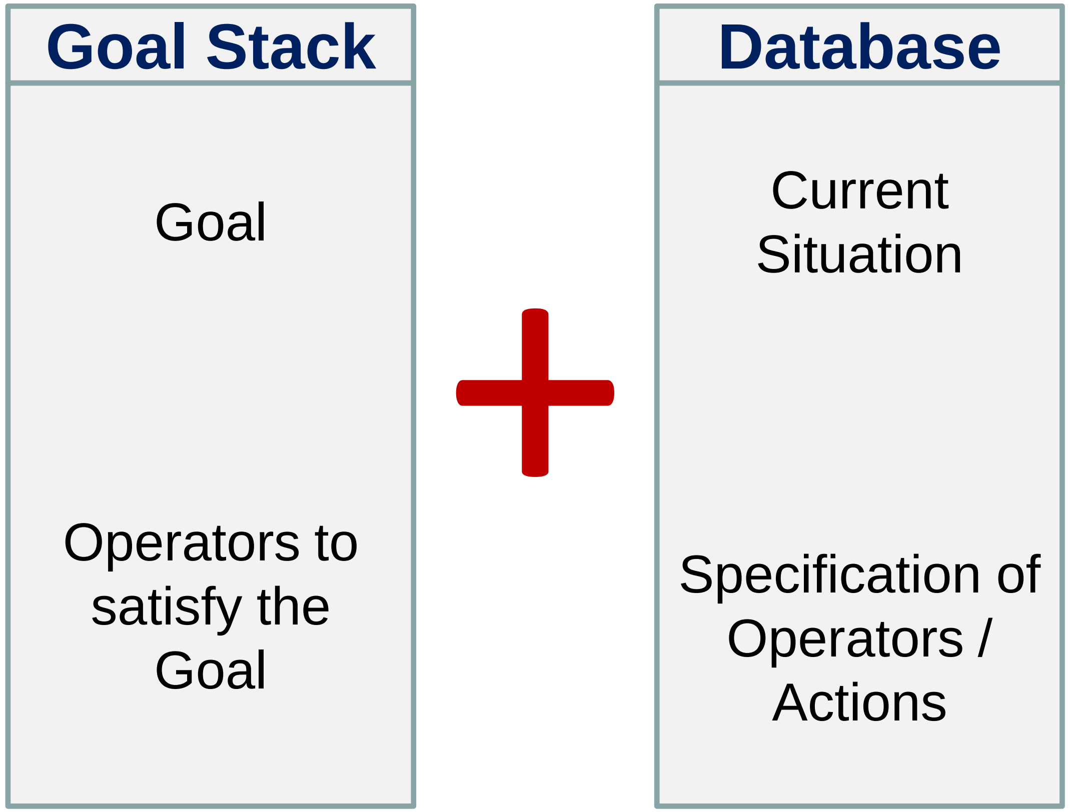
1. Database
2. Goal stack



STRIPS Goal Stack Planning

- Place goal in goal stack
- Considering top Goal1
- Place onto its sub-goals
- Then try to solve sub-goal
- GoalS1-2
- Continue…

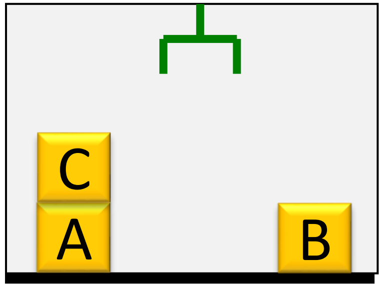


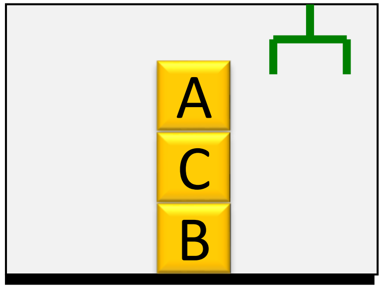


Stack Manipulation Rules

If on TOP of the Stack	Then Do
Compound or Single goal Matching the current state description	Remove it
Compound goal Not matching the current state description	1. Keep original compound goal on stack 2. List the unsatisfied component goals on the stack in some new order
Single-literal goal Not matching the current state description	1. Find new action who's instantiated add-list includes the goal 2. Replace the goal with the action 3. Place the action’s precondition formula on top of stack
Action	1. Remove action from stack 2. Update database using action 3. Keep track of action (for solution)
Nothing	Stop

Step: 1



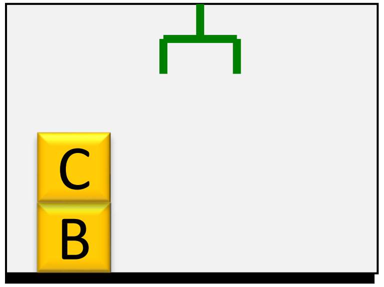


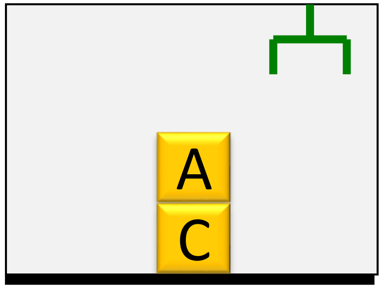
Goal Stack
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution	

Step: 2





Goal Stack

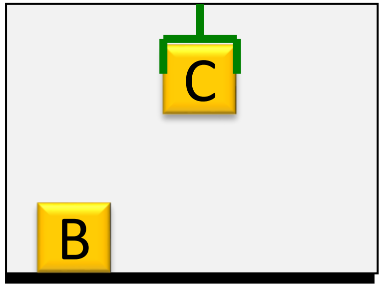
On(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

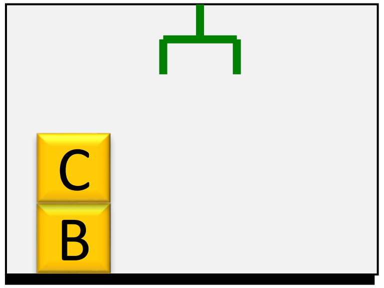
Database

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

Solution

Step: 2



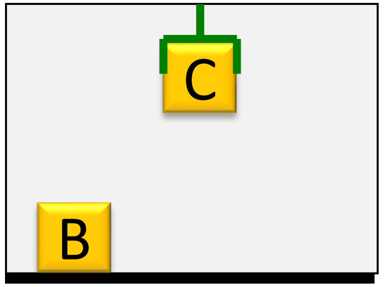


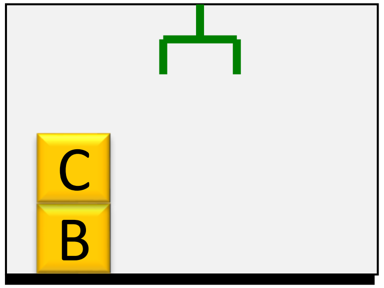
Goal Stack
On(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution	

Step: 3



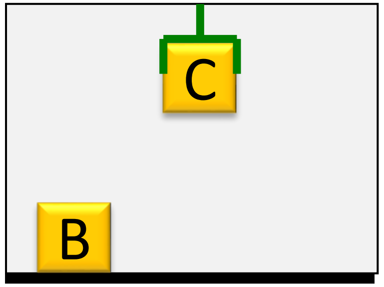


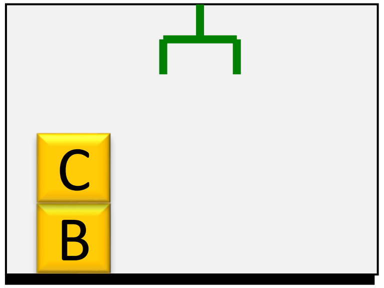
Goal Stack
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution	

Step: 4



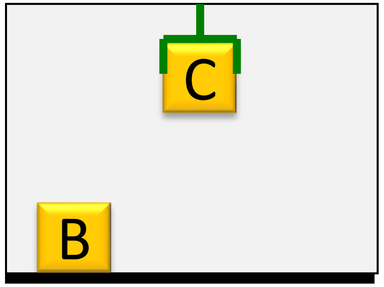


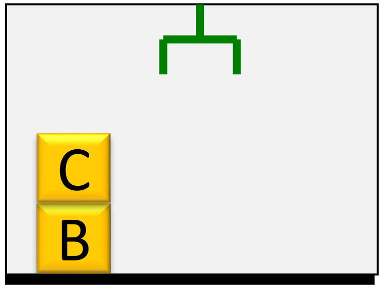
Goal Stack
Clear(B)
Holding(C)
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution

Step: 5



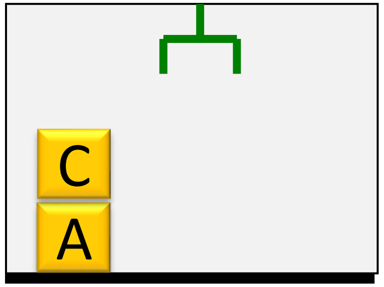


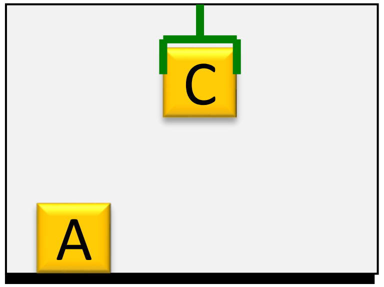
Goal Stack
Clear(B)
Holding(C)
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution

Step: 5



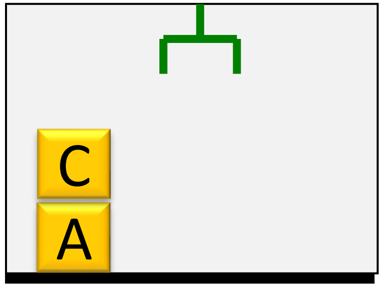


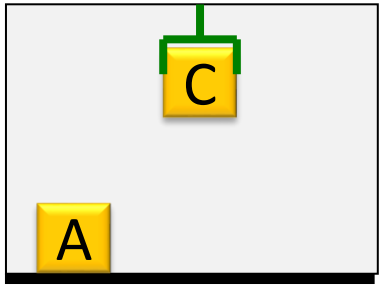
Goal Stack
Holding(C)
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution	

Step: 6



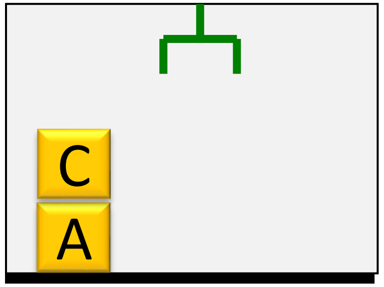


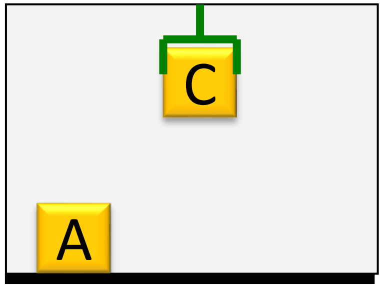
Goal Stack
On(C, A) ∧ Clear(C) ∧ Handempty
Unstack(C, A)
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution	

Step: 7



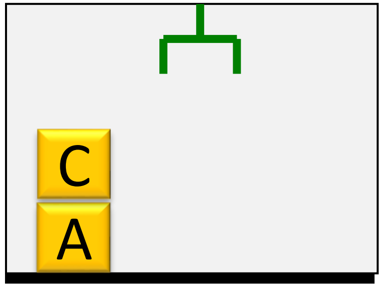


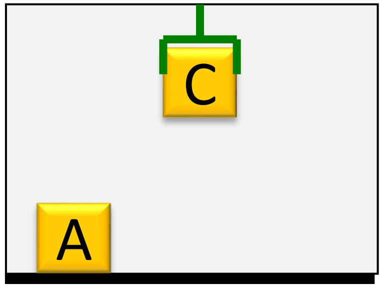
Goal Stack
On(C, A) ∧ Clear(C) ∧ Handempty
Unstack(C, A)
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution	

Step: 8





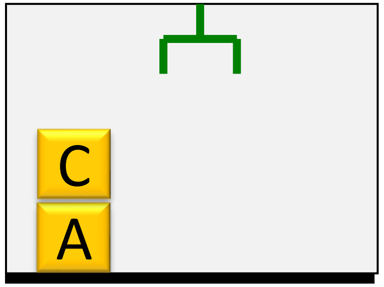
Unstack(C,A):
Add - [Holding(C), Clear(A)]
Delete -[Handempty, Clear(C), On(C,A)]

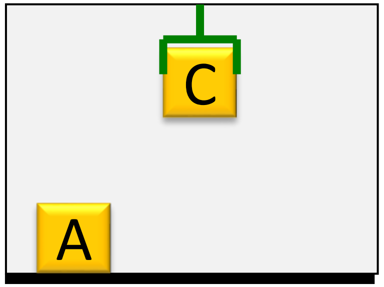
Goal Stack
Unstack(C, A)
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

Solution	

Step: 8





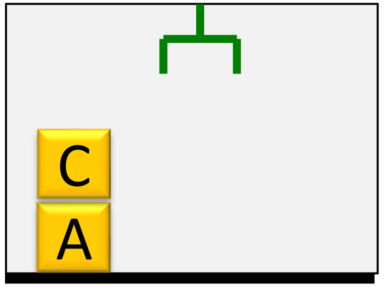
Unstack(C,A):
Add - [Holding(C), Clear(A)]
Delete -[Handempty, Clear(C), On(C,A)]

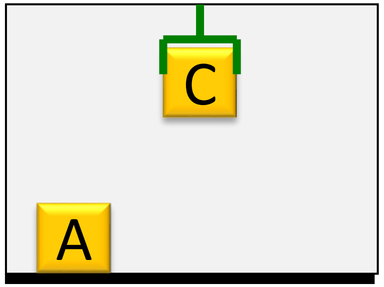
Goal Stack
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(A, TABLE)
On(B, TABLE)
Clear(A)
Holding(C)

Solution	Unstack(C, A)

Step: 9



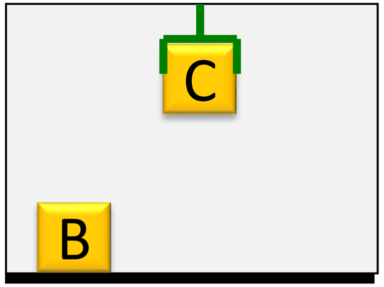


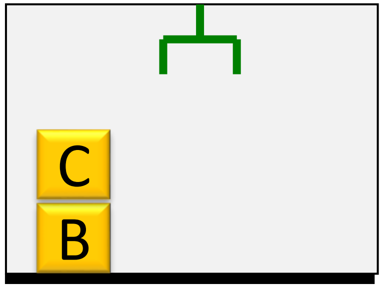
Goal Stack
Holding(C) ∧ Clear(B)
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(A, TABLE)
On(B, TABLE)
Clear(A)
Holding(C)

Solution	Unstack(C, A)

Step: 10



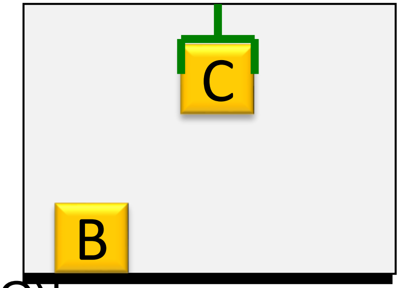


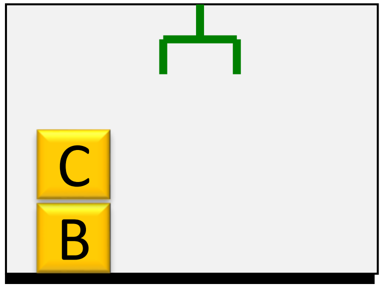
Goal Stack
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(A, TABLE)
On(B, TABLE)
Clear(A)
Holding(C)

Solution	Unstack(C, A)

Step: 10





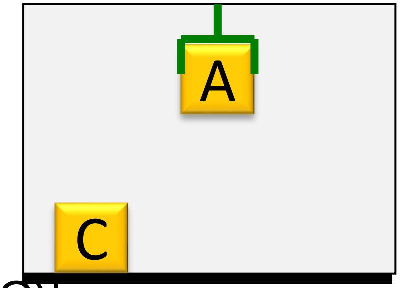
Stack(C,B):
Add - [Handempty, On(C,B), Clear(C)]
Delete - [Holding(C), Clear(B)]

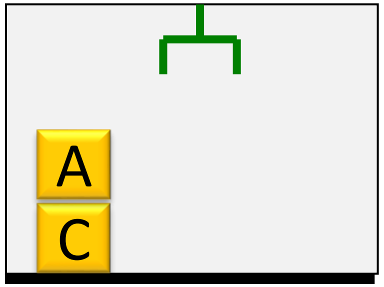
Goal Stack
Stack(C, B)
On(A, C)
On(A, C) ∧ On(C, B)

Database
Clear(B)
On(A, TABLE)
On(B, TABLE)
Clear(A)
Holding(C)

Solution	Unstack(C, A)

Step: 11





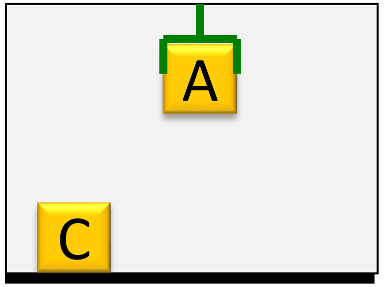
Stack(C,B):
Add - [Handempty, On(C,B), Clear(C)]
Delete - [Holding(C), Clear(B)]

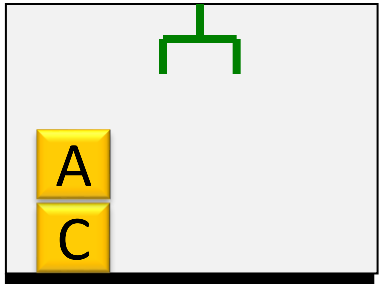
Goal Stack
On(A, C)
On(A, C) ∧ On(C, B)

Database
On(A, TABLE)
On(B, TABLE)
Clear(A)
Clear(C)
On(C, B)
Handempty

Solution	Unstack(C, A), Stack(C, B)

Step: 12

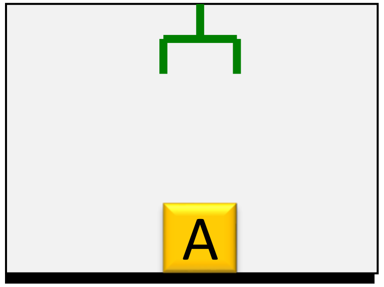


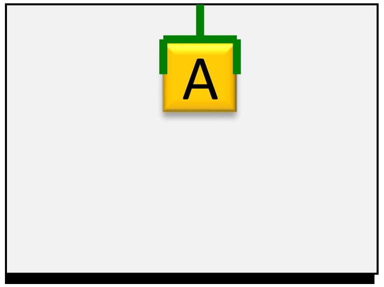


Goal Stack
Holding(A) ∧ Clear(C)
Stack(A, C)
On(A, C) ∧ On(C, B)

Database
On(A, TABLE)
On(B, TABLE)
Clear(A)
Clear(C)
On(C, B)
Handempty

Solution	Unstack(C, A), Stack(C, B)



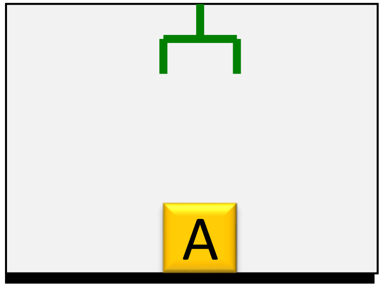


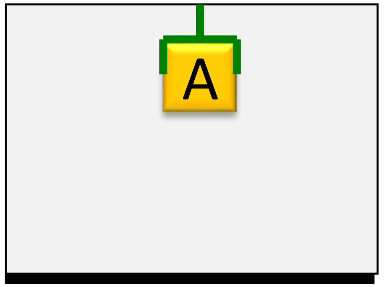
Goal Stack
Holding(A)
Holding(A) ∧ Clear(C)
Stack(A, C)
On(A, C) ∧ On(C, B)

Database
On(A, TABLE)
On(B, TABLE)
Clear(A)
Clear(C)
On(C, B)
Handempty

Solution	Unstack(C, A), Stack(C, B)

Step: 13



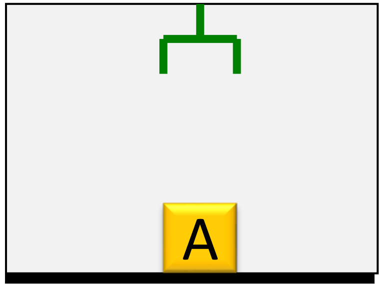


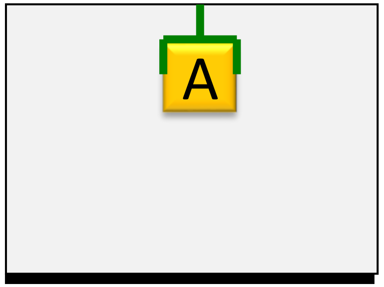
Goal Stack
On(A, TABLE) ∧ Clear(A) ∧ Handempty
Pickup(A)
Holding(A) ∧ Clear(C)
Stack(A, C)
On(A, C) ∧ On(C, B)

Database
On(A, TABLE)
On(B, TABLE)
Clear(A)
Clear(C)
On(C, B)
Handempty

Solution	Unstack(C, A), Stack(C, B)

Step: 14



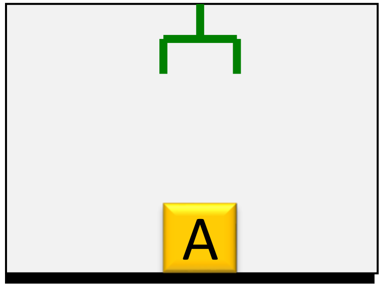


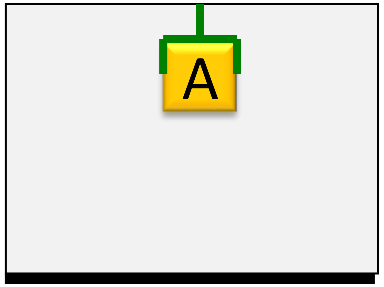
Goal Stack
On(A, TABLE) ∧ Clear(A) ∧ Handempy
Pickup(A)
Holding(A) ∧ Clear(C)
Stack(A, C)
On(A, C) ∧ On(C, B)

Database
On(A, TABLE)
On(B, TABLE)
Clear(A)
Clear(C)
On(C, B)
Handempty

Solution Unstack(C, A), Stack(C, B)

Step: 16





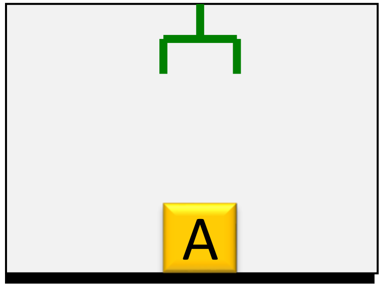
Pickup(A):
Add - [Holding(A)]
Delete - [On(A, TABLE), Clear(A), Handempty]

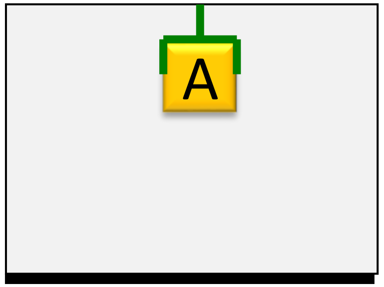
Goal Stack
Pickup(A)
Holding(A) ∧ Clear(C)
Stack(A, C)
On(A, C) ∧ On(C, B)

Database
On(A, TABLE)
On(B, TABLE)
Clear(A)
Clear(C)
On(C, B)
Handempty

Solution	Unstack(C, A), Stack(C, B)

Step: 17





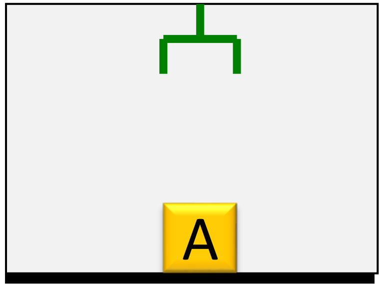
Pickup(A):
Add - [Holding(A)]
Delete - [On(A, TABLE), Clear(A), Handempty]

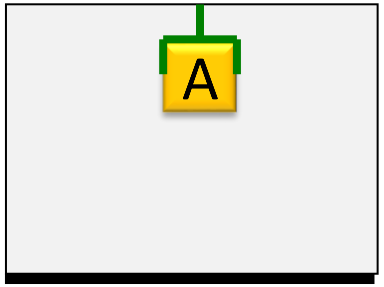
Goal Stack
Holding(A) ∧ Clear(C)
Stack(A, C)
On(A, C) ∧ On(C, B)

Database
On(B, TABLE)
Clear(C)
On(C, B)
Holding(A)

Solution	Unstack(C, A), Stack(C, B), Pickup(A)

Step: 17





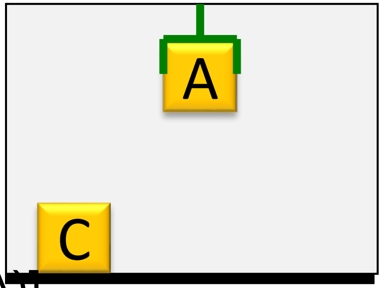
Goal Stack
Holding(A) ∧ Clear(C)
Stack(A, C)
On(A, C) ∧ On(C, B)

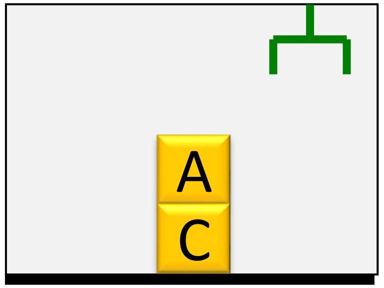
Database
On(B, TABLE)
Clear(C)
On(C, B)
Holding(A)

Solution	Unstack(C, A), Stack(C, B), Pickup(A)

Step: 18

Stack(A,C):
Add - [Handempty, On(A,C), Clear(A)]
Delete - [Holding(A), Clear(C)]



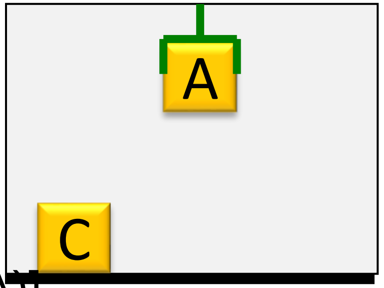


Goal Stack
Stack(A, C)
On(A, C) ∧ On(C, B)

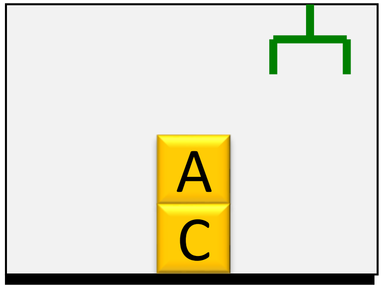
Database
On(B, TABLE)
Clear(C)
On(C, B)
Holding(A)

Solution	Unstack(C, A), Stack(C, B), Pickup(A)

Step: 18



Stack(A,C):
Add - [Handempty, On(A,C), Clear(A)]
Delete - [Holding(A), Clear(C)]

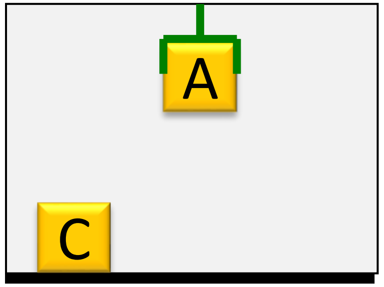


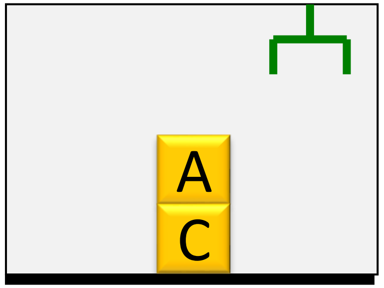
Goal Stack
On(A, C) ∧ On(C, B)

Database
On(B, TABLE)
On(C, B)
Clear(A)
On(A, C)
Handempty

Solution	Unstack(C, A), Stack(C, B), Pickup(A), Stack(A, C)

Step: 19



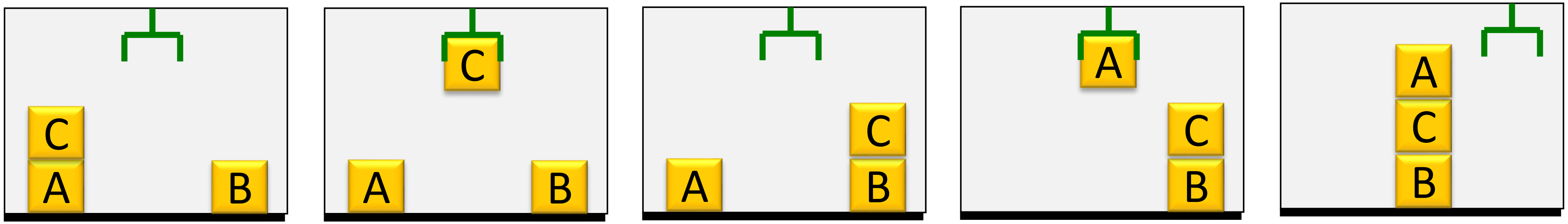


Goal Stack
On(A, C) ∧ On(C, B)

Database
On(B, TABLE)
On(C, B)
Clear(A)
On(A, C)
Handempty

Solution	Unstack(C, A), Stack(C, B), Pickup(A), Stack(A, C)

Step: 20



Goal Stack

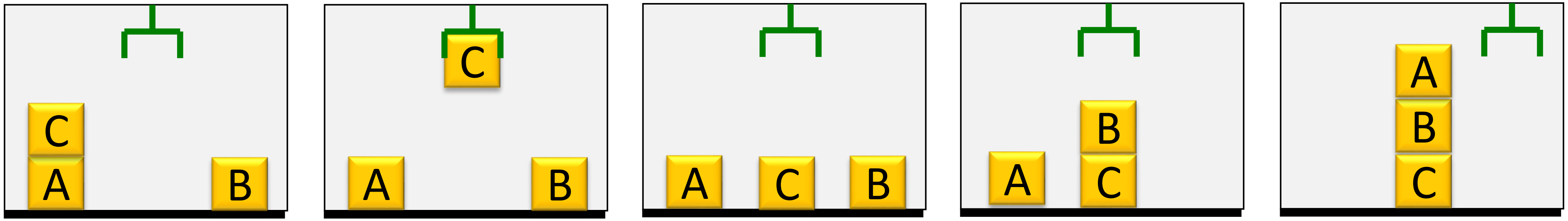
Database
On(B, TABLE)
On(C, B)
Clear(A)
On(A, C)
Handempty

Solution	Unstack(C, A), Stack(C, B), Pickup(A), Stack(A, C)

Stack Manipulation Rules

If on TOP of the Stack	Then Do
Compound or Single goal Matching the current state description	Remove it
Compound goal Not matching the current state description	1. Keep original compound goal on stack 2. List the unsatisfied component goals on the stack in some new order
Single-literal goal Not matching the current state description	1. Find new action who's instantiated add-list includes the goal 2. Replace the goal with the action 3. Place the action's precondition formula on top of stack
Action	1. Remove action from stack 2. Update database using action 3. Keep track of action (for solution)
Nothing	Stop

Goal Stack



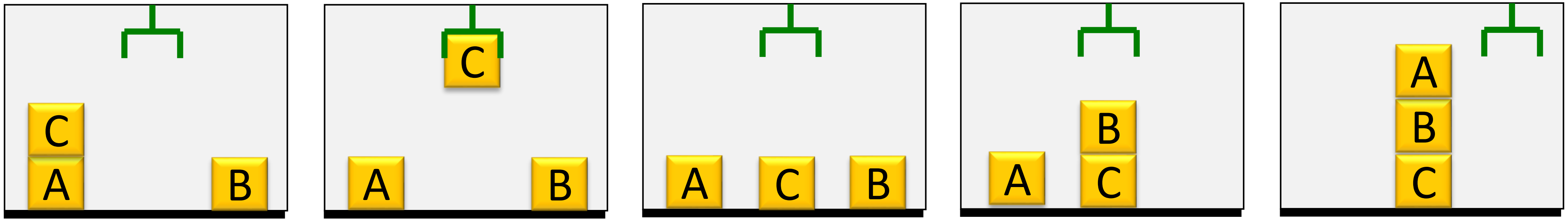
- **This method may fail to find the good solution**
- **There are two way that could begin to solve this problem**

Goal Stack
On(B, C)
On(A, B)
On(A, B) ∧ On(B, C)

Database
Clear(B)
On(C,A)
Clear(C)
On(A, TABLE)
On(B, TABLE)
Handempty

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

Goal Stack



Goal Stack
On(B, C)
On(A, B)
On(A, B) ∧ On(B, C)

1st

Goal Stack
On(A, B)
On(B, C)
On(A, B) ∧ On(B, C)

2nd

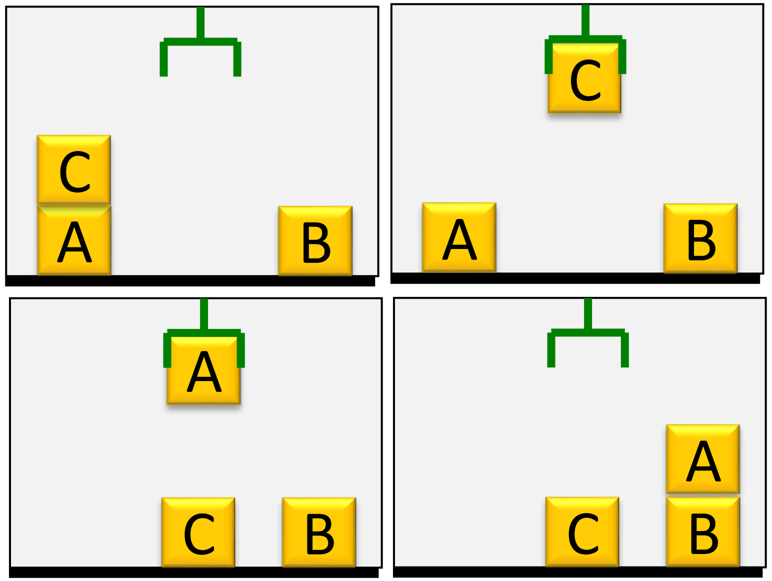
Goal Stack

On(C, A) ∧ Clear(C) ∧ Handempty
Unstack(C, A)
Handempty
Clear(A) ∧ Handempty
Pickup(A)
Clear(B) ∧ Holding(A)
Stack(A, B)
On(B, C)
On(A, B) ∧ On(B, C)

Goal Stack

Holding(C)
Putdown(C)
Handempty
Clear(A) ∧ Handempty
Pickup(A)
Clear(B) ∧ Holding(A)
Stack(A, B)
On(B, C)
On(A, B) ∧ On(B, C)

Goal Stack

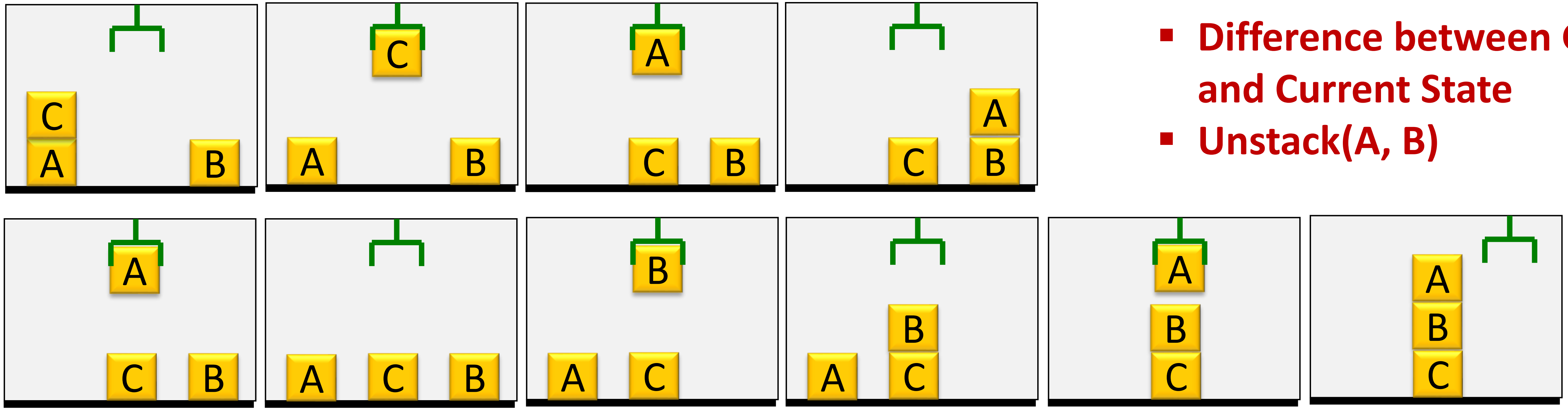


On(B, C)
On(A, B) ∧ On(B, C)

Goal Stack

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

Goal Stack



Pickup(A) Stack(A, B)

- **Difference between Gol and Current State**
- **Unstack(A, B)**

Goal Stack
On(B, C)
On(A, B) ∧ On(B, C)

Goal Stack
On(A, B) ∧ On(B, C)

Sussman Anomaly

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

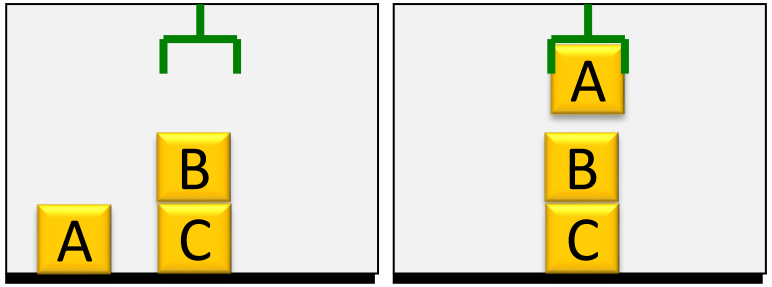
Linear vs Nonlinear Planning

- Goal Stack Planning
  - Problem involving conjoined goals
  - Solve the goal one at time, in order
  - Plan contain a sequence of operators for attending first sub-goal followed by second sub-goal.
- Advantages
  - Reduced search space, since goals are solved one at a time
  - Advantageous if goals are (mainly) independent
  - Linear planning is **sound**
- Difficulty
  - Ordering of sub-goals may leads to **suboptimal** or **incomplete.**
  - Poor **irreversible actions**
  - Operator used to solve the sub-goal may interfere with the solution of previous sub-goal.
- Solution

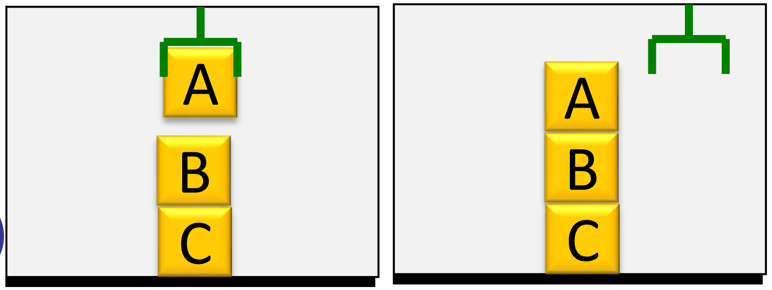
- Nonlinear Planning
  - Multiple sub-goals are solved **simultaneously**.
  - Use the **set** instead of goal stack
  - Include in the search space **all possible subgoal orderings**
  - Handles goal interactions by **interleaving**
- Advantages
  - Non-linear planning is **sound**
  - Non-linear planning is **complete**
  - Non-linear planning **may be optimal** with respect to plan length (depending on search strategy employed)
- Disadvantages
  - **Larger search space**, since all possible goal orderings may have to be considered
  - Somewhat more **complex algorithm**
  - More bookkeeping

TWEAK Representation

- Action — Pickup(A)
- Preconditions — Clear(B) ∧ On(A, TABLE) ∧ Handempty
- Postconditions — Hold(A) ∧ ¬ On(A, TABLE) ∧ ¬ Handempty



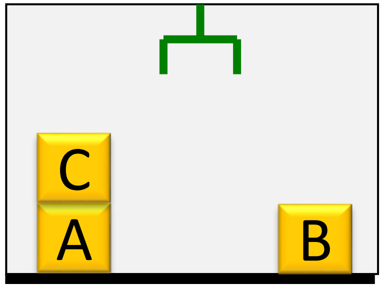
- Action — Stack (A, B)
- Preconditions — Clear(B) ∧ Holding(A)
- Postconditions — Handempty ∧ On(A, B) ∧ ¬ Clear(B) ∧ ¬ Holding(A)

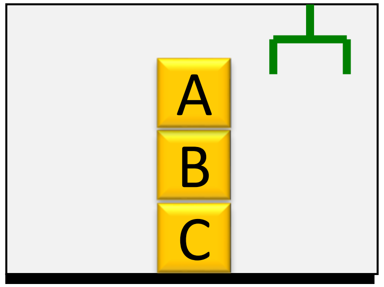


Heuristic Planning using Constraint Posting: TWEAK

- Heuristic Operations for plan modification:
  - Step Addition: Creating new step for plan
  - Promotion: Constraint one step to come before another in a final plan
  - DeClobbeering: Placing one step S2 between two old steps S1 and S3 such that S2 reasserts some precondition of S3 that was negated by S1
  - Simple Binding: Assigning a value to a variable, in order to ensure the precondition of some step
  - Separation: Preventing the assignment of certain values to a variable.

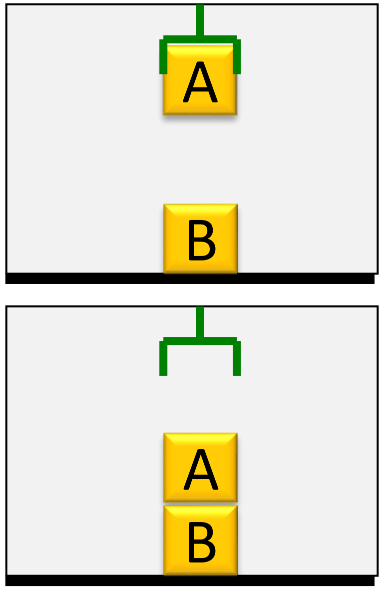
Block World Example

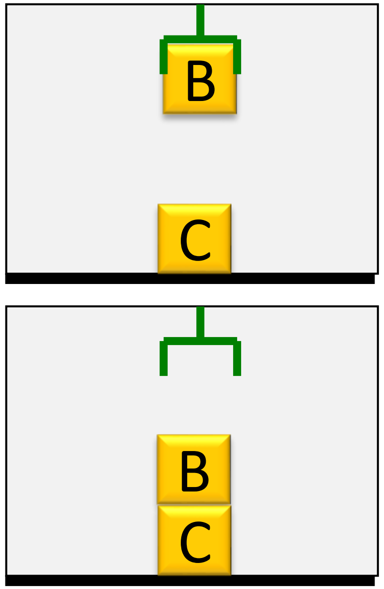


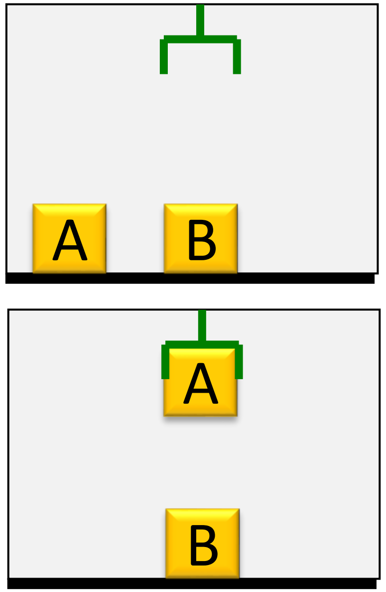


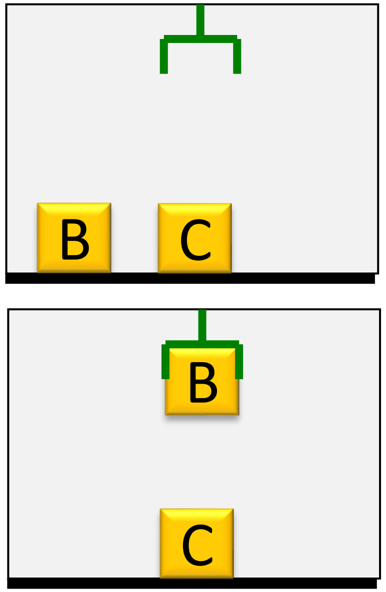
On(A, TABLE) $\wedge$ On(C, A)
On(B, TABLE) $\wedge$ Handempty

On(A, B) $\wedge$ On(B, C)









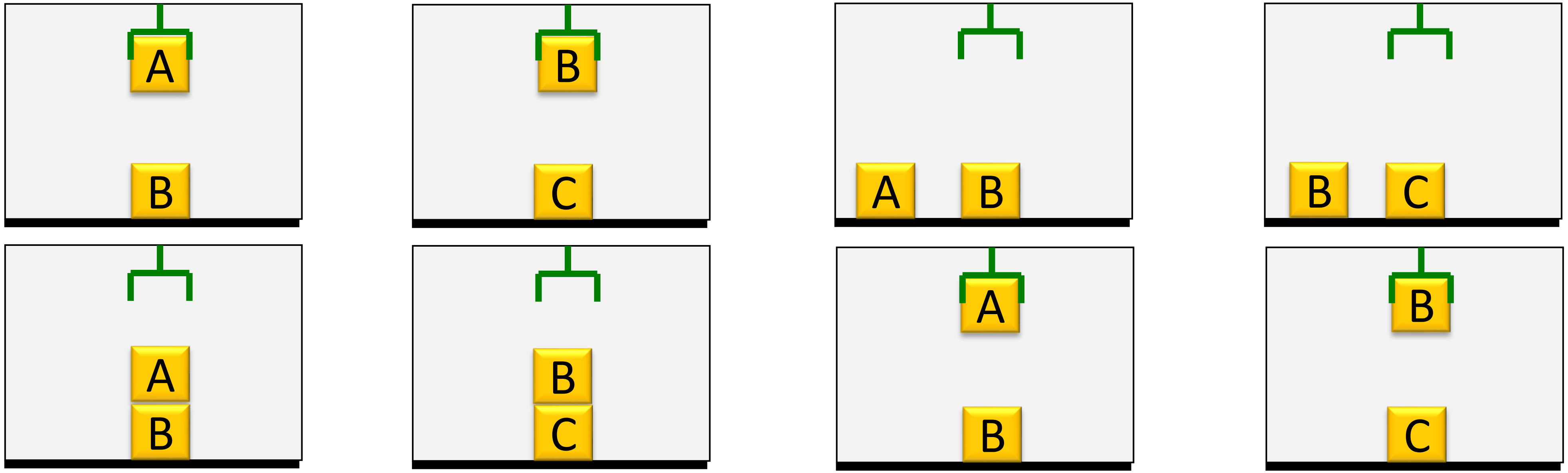
		*Clear(A)	*Clear(B)
Clear(B)	Clear(C)	On(A, TABLE)	On(B, TABLE)
*Holding(A)	*Holding(B)	*Handempty	*Handempty
Stack(A, B)	**Stack(B, C)**	**Pickup(A)**	**Pickup(B)**
Handempty	Handempty	Holding(A)	Holding(B)
On(A, B)	On(B, C)	¬On(A, TABLE)	¬On(B, TABLE)
¬Clear(B)	¬Clear(C)	¬Handempty	¬Handempty
¬Holding(A)	¬Holding(B)		

Plan = {Stack (A, B), Stack(B, C)}

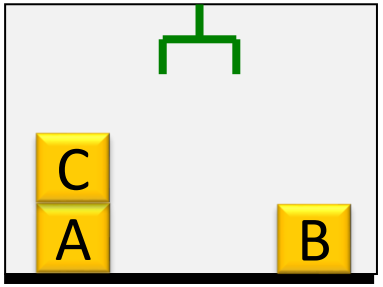
Plan = {Stack A, B), Stack(B, C), Pickup(A), Pickup(B)}

Plan = {Stack A, B), Stack(B, C), Pickup(A), Pickup(B)}

Pickup(A) ← Stack(A, B)
Pickup(B) ← Stack(B, C)



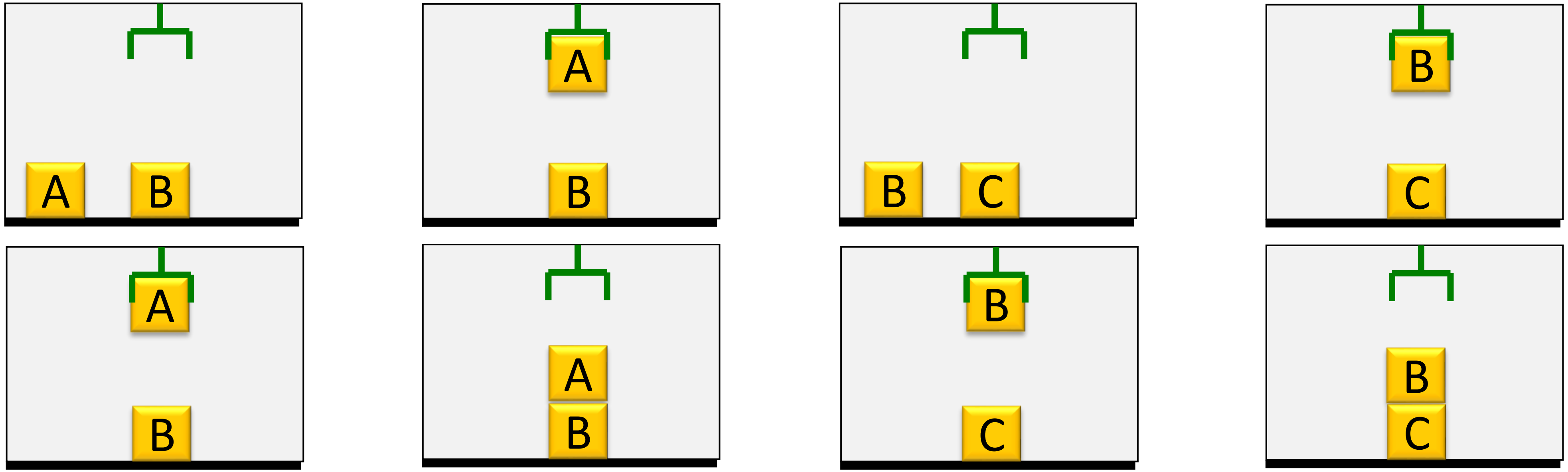
		*Clear(A)	*Clear(B)
Clear(B)	Clear(C)	On(A, TABLE)	On(B, TABLE)
*Holding(A)	*Holding(B)	*Handempty	*Handempty
Stack(A, B)	**Stack(B, C)**	**Pickup(A)**	**Pickup(B)**
Handempty	Handempty	Holding(A)	Holding(B)
On(A, B)	On(B, C)	¬On(A, TABLE)	¬On(B, TABLE)
¬Clear(B)	¬Clear(C)	¬Handempty	¬Handempty
¬Holding(A)	¬Holding(B)		



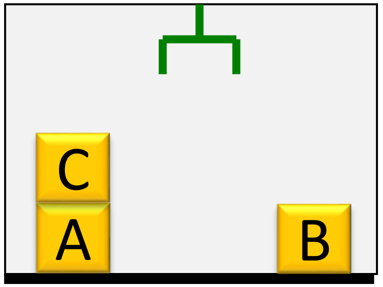
Pickup(A) ← Stack(A, B)

Pickup(B) ← Stack(B, C)

Plan = {Pickup(A), Stack (A, B), Pickup(B), Stack(B, C)}

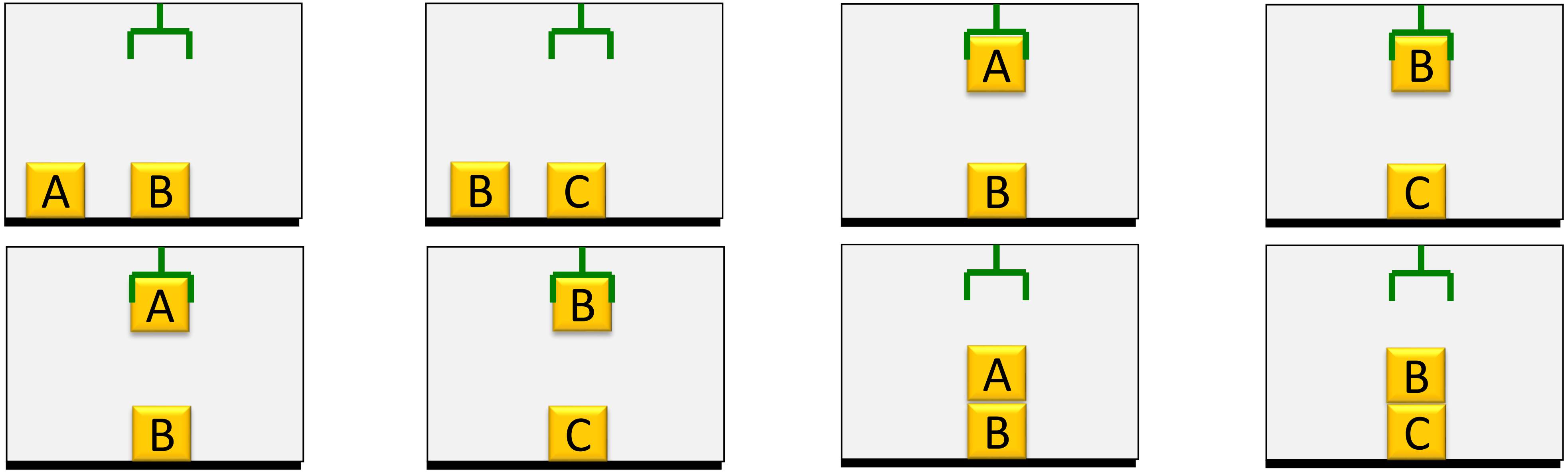


*Clear(A)		*Clear(B)	
On(A, TABLE)	Clear(B)	On(B, TABLE)	Clear(C)
*Handempty	*Holding(A)	*Handempty	*Holding(B)
Pickup(A)	**Stack(A, B)**	**Pickup(B)**	**Stack(B, C)**
Holding(A)	Handempty	Holding(B)	Handempty
¬On(A, TABLE)	On(A, B)	¬On(B, TABLE)	On(B, C)
¬Handempty	¬Clear(B)	¬Handempty	¬Clear(C)
	¬Holding(A)		

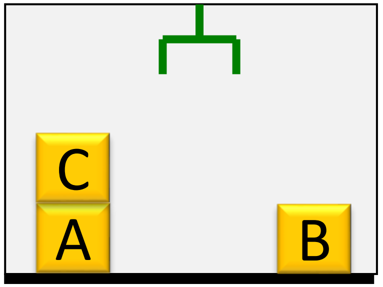


Pickup(B) ← Stack(A, B)

Plan = {Pickup(A), Pickup(B), Stack (A, B), Stack(B, C)}

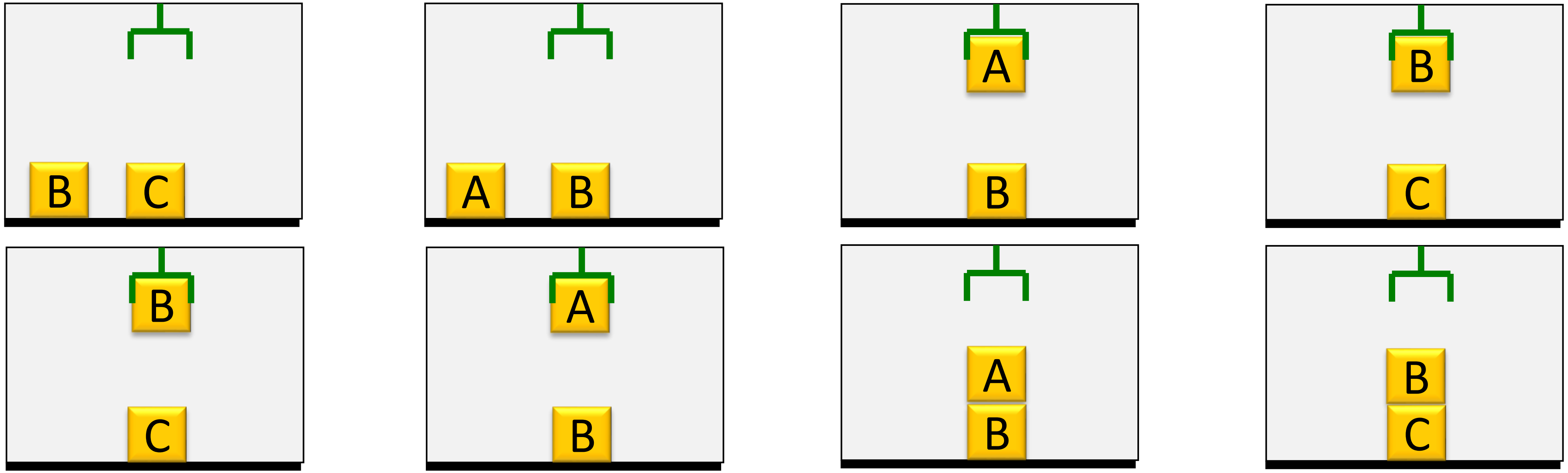


*Clear(A)	*Clear(B)		
On(A, TABLE)	On(B, TABLE)	Clear(B)	Clear(C)
*Handempty	*Handempty	*Holding(A)	*Holding(B)
Pickup(A)	**Pickup(B)**	**Stack(A, B)**	**Stack(B, C)**
Holding(A)	Holding(B)	Handempty	Handempty
¬On(A, TABLE)	¬On(B, TABLE)	On(A, B)	On(B, C)
¬Handempty	¬Handempty	¬Clear(B)	¬Clear(C)
		¬Holding(A)	

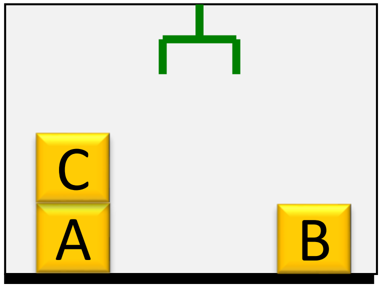


Pickup(B) ← Pickup(A)

Plan = {Pickup(B), Pickup(A), Stack A, B), Stack(B, C)}

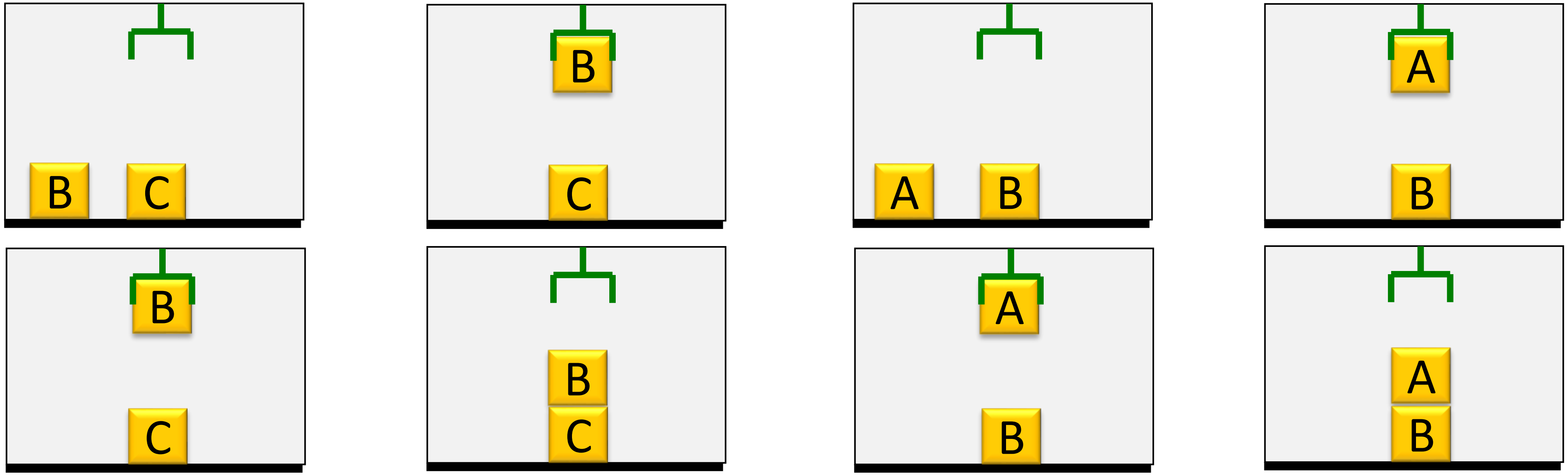


*Clear(B)	*Clear(A)		
On(B, TABLE)	On(A, TABLE)	Clear(B)	Clear(C)
*Handempty	*Handempty	*Holding(A)	*Holding(B)
Pickup(B)	**Pickup(A)**	**Stack(A, B)**	**Stack(B, C)**
Holding(B)	Holding(A)	Handempty	Handempty
¬On(B, TABLE)	¬On(A, TABLE)	On(A, B)	On(B, C)
¬Handempty	¬Handempty	¬Clear(B)	¬Clear(C)
		¬Holding(A)	

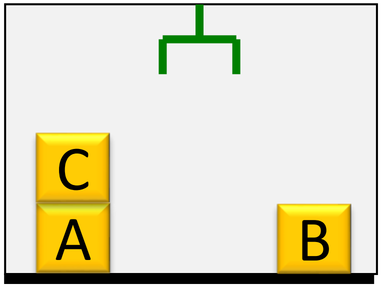


Pickup(B) ← Stack(B, C) ← Pickup(A)

Plan = {Pickup(B), Stack(B, C), Pickup(A), Stack A, B)}



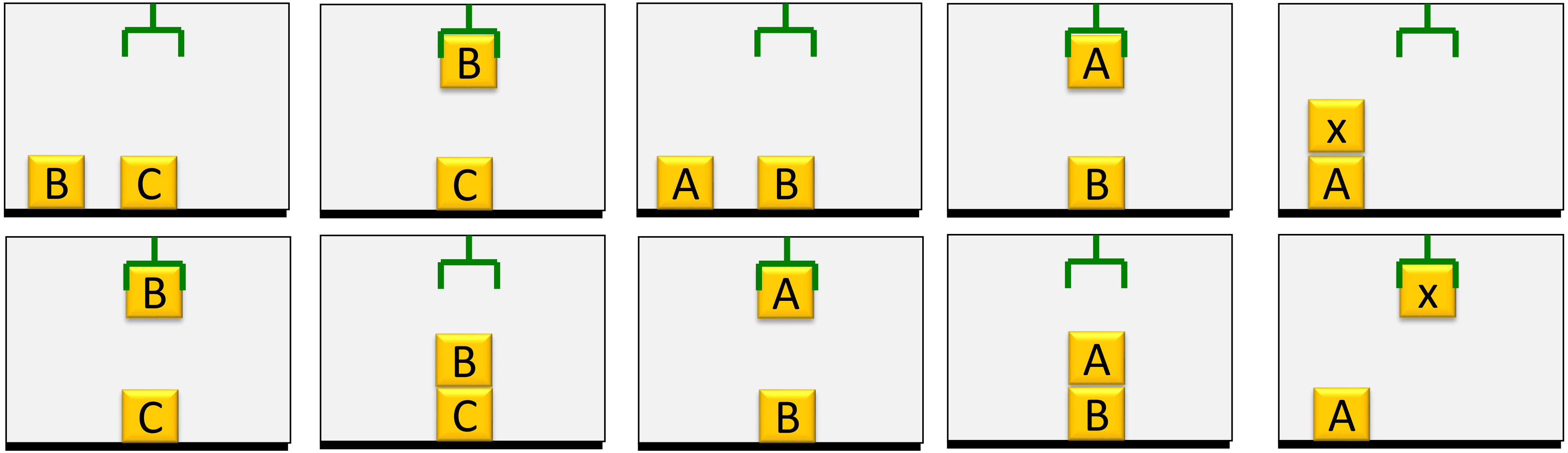
Clear(B)		*Clear(A)	
On(B, TABLE)	Clear(C)	On(A, TABLE)	Clear(B)
Handempty	Holding(B)	*Handempty	*Holding(A)
Pickup(B)	**Stack(B, C)**	**Pickup(A)**	**Stack(A, B)**
Holding(B)	Handempty	Holding(A)	Handempty
¬On(B, TABLE)	On(B, C)	¬On(A, TABLE)	On(A, B)
¬Handempty	¬Clear(C)	¬Handempty	¬Clear(B)
			¬Holding(A)



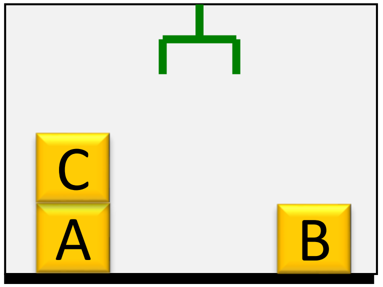
Unstack(x, A)

Plan = {Pickup(B), Stack(B, C), Pickup(A), Stack A, B), Unstack(x, A) }

x = C in step Unstack(x, A)

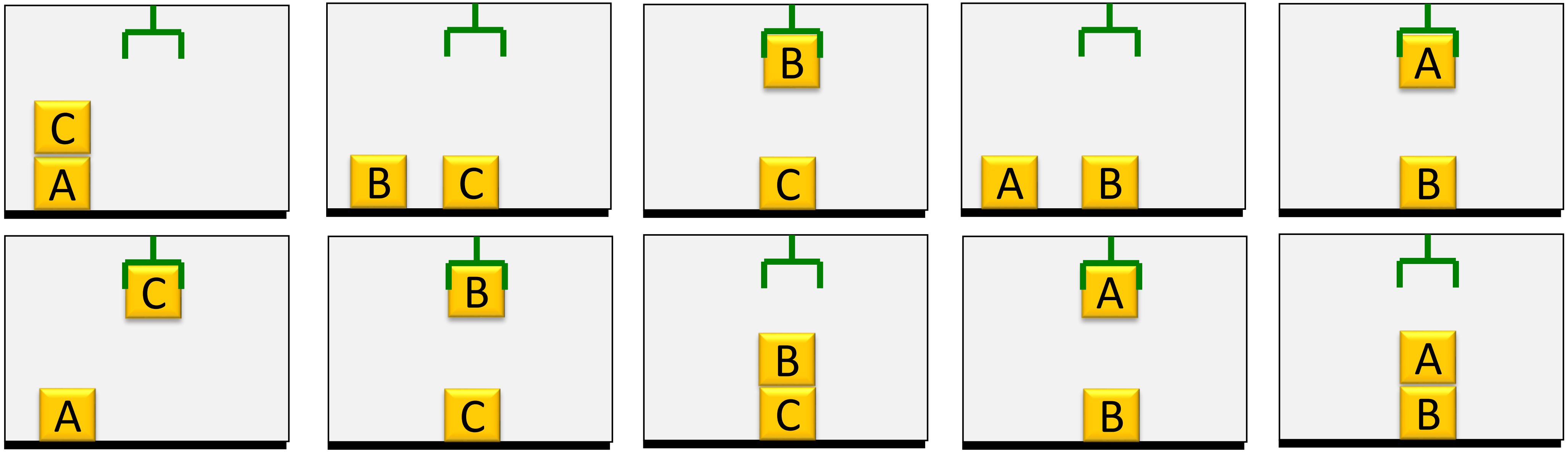


Clear(B)		*Clear(A)		On(x, A)
On(B, TABLE)	Clear(C)	On(A, TABLE)	Clear(B)	Clear(x)
Handempty	Holding(B)	*Handempty	*Holding(A)	Handempty
Pickup(B)	**Stack(B, C)**	**Pickup(A)**	**Stack(A, B)**	**Unstack(x, A)**
Holding(B)	Handempty	Holding(A)	Handempty	¬Handempty
¬On(B, TABLE)	On(B, C)	¬On(A, TABLE)	On(A, B)	Clear(A)
¬Handempty	¬Clear(C)	¬Handempty	¬Clear(B)	Holding(x)
			¬Holding(A)	¬On(x, A)

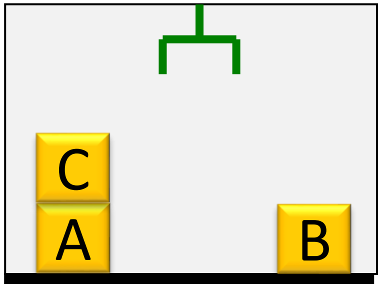


Unstack(C, A) ← Pickup(B) ← Stack(B, C) ← Pickup(A) ← Stack(A, B)

Plan = {Unstack(x, A), Pickup(B), Stack(B, C), Pickup(A), Stack A, B)}

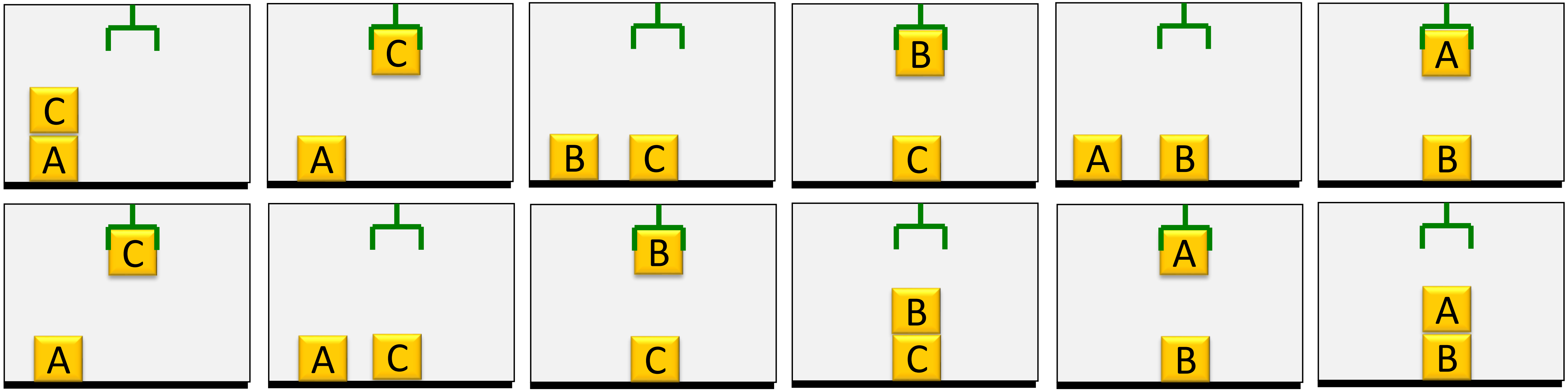


On(C, A)	Clear(B)		*Clear(A)	
Clear(C)	On(B, TABLE)	Clear(C)	On(A, TABLE)	Clear(B)
Handempty	Handempty	Holding(B)	*Handempty	*Holding(A)
Unstack(C, A)	**Pickup(B)**	**Stack(B, C)**	**Pickup(A)**	**Stack(A, B)**
¬Handempty	Holding(B)	Handempty	Holding(A)	Handempty
Clear(A)	¬On(B, TABLE)	On(B, C)	¬On(A, TABLE)	On(A, B)
Holding(C)	¬Handempty	¬Clear(C)	¬Handempty	¬Clear(B)
¬On(C, A)				¬Holding(A)



Unstack(C, A) ← Putdown(C) ← Pickup(B)

Plan = {Unstack(C, A), Putdown(C), Pickup(B), Stack(B, C), Pickup(A), Stack A, B)}



On(C, A)		Clear(B)		*Clear(A)	
Clear(C)		On(B, TABLE)	Clear(C)	On(A, TABLE)	Clear(B)
Handempty	Holding(C)	Handempty	Holding(B)	*Handempty	*Holding(A)
Unstack(C, A)	**Putdown(C)**	**Pickup(B)**	**Stack(B, C)**	**Pickup(A)**	**Stack(A, B)**
¬Handempty	¬Holding(C)	Holding(B)	Handempty	Holding(A)	Handempty
Clear(A)	On(C, TABLE)	¬On(B, TABLE)	On(B, C)	¬On(A, TABLE)	On(A, B)
Holding(C)	Handempty	¬Handempty	¬Clear(C)	¬Handempty	¬Clear(B)
¬On(C, A)					¬Holding(A)

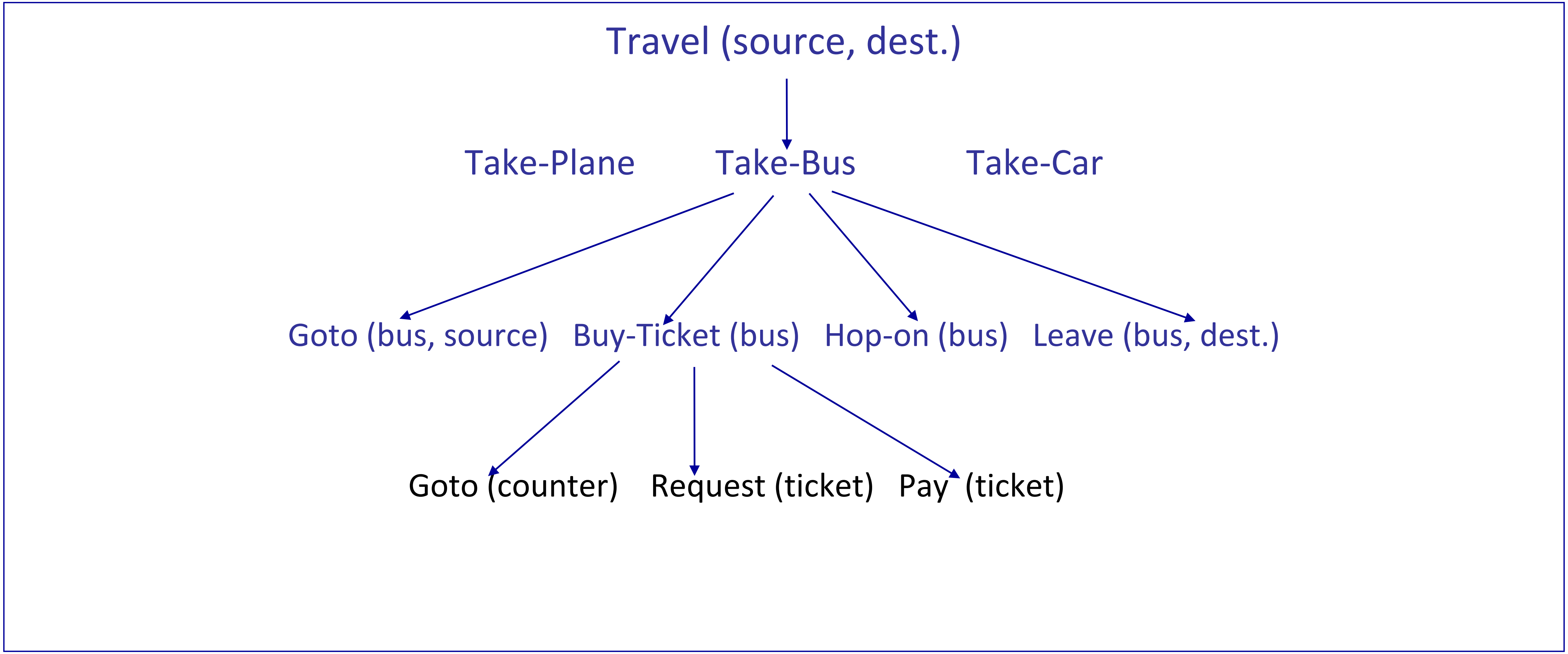
Hierarchical Planning

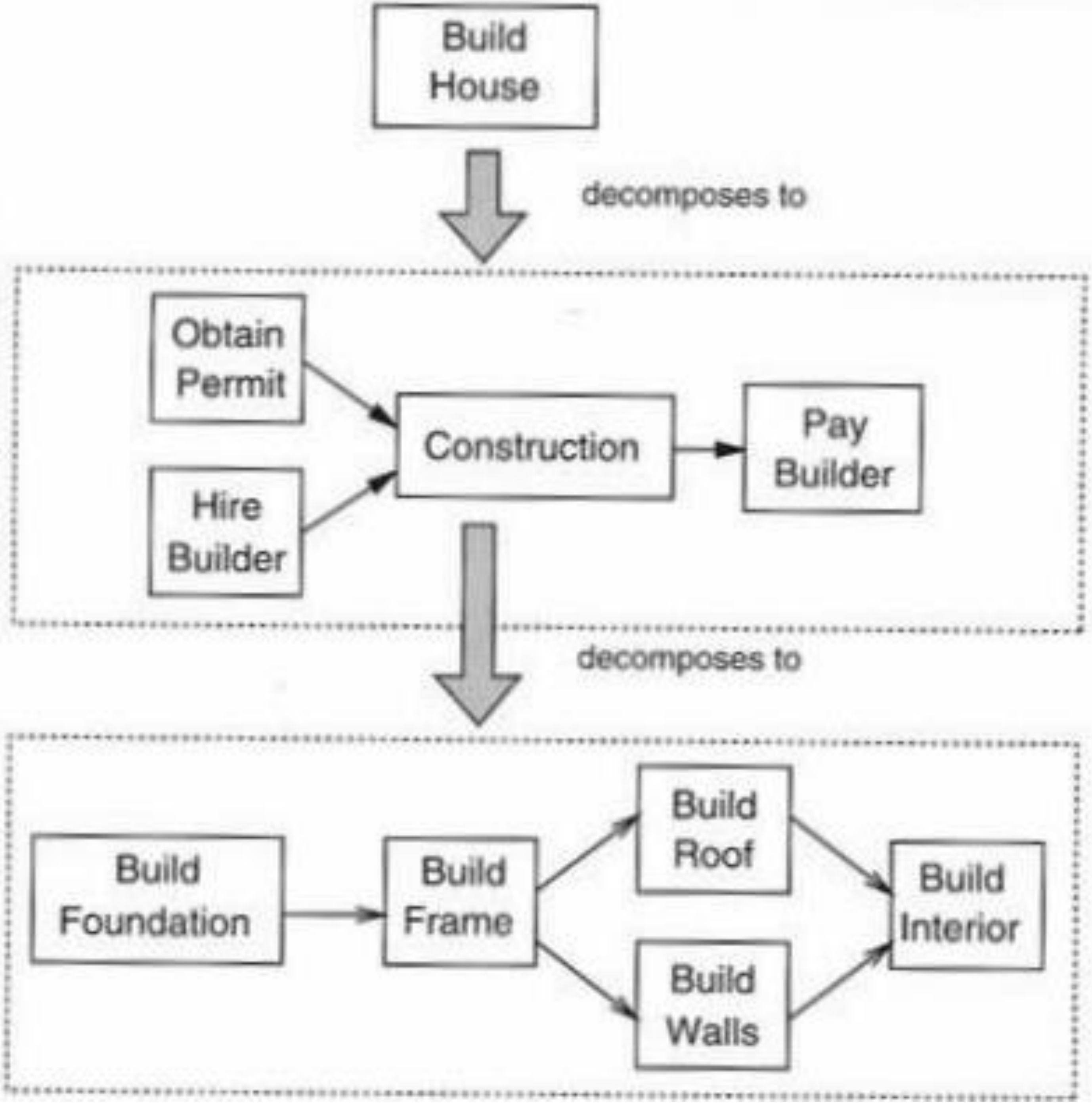
- **Hierarchical Planning / Plan Decomposition**
- Plans are organized in a hierarchy. Links between nodes at different levels in the hierarchy denote a decomposition of a "complex action" into more primitive actions (operator expansion).
- Example:
  - move (x, y, z)
- operator
- expansion

pickup (x, y)   putdown (x, z)

- *The lowest level corresponds to executable actions of the agent.*

Hierarchical Plan - Example




Build House
decomposes to
Obtain Permit
Hire Builder
Construction
Pay Builder
decomposes to
Build Foundation
Build Frame
Build Roof
Build Walls
Build Interior

Timeline of Planner

Date	Planner	Authors	Significant Contributions
1956	Logic Theorist	Newell, Shaw, Simon	first serious use of heuristics; backward search
1959	Gelerntner's Geometry Theorem-Proving Machine	Gelerntner	first system to handle conjunctive subgoals
1957-1969	GPS	Newell, Shaw, Simon	introduced means-end analysis; separating domain knowledge from general search knowledge
1969	QA3	Green	planned via resolution theorem proving; introduced situations in propositions; required use of frame axioms
1971	STRIPS	Fikes and Nilsson	introduced STRIPS operators; triangle tables to learn MACROPs and cope with uncertainty
1973	HACKER	Sussman	idea of using debugging experts to revise a completed plan; introduces notion of *protection*; reorders goals to cope with protection violations
1973	WARPLAN	Warren	introduced (?) the idea of promotion; also an early use of the idea of skeleton plans; written in PROLOG
1974	INTERPLAN	Tate	like HACKER, but promotes subgoals if reordering fails
1974	ABSTRIPS	Sacerdoti	clearly introduces concept of abstraction hierarchy, criticality of operators
1975	Waldinger's Planner	Waldinger	introduced goal regression
1975	NOAH	Sacerdoti	introduced nonlinear planning; uses critics to fix/improve plans
1977	NONLIN	Tate	improved NOAH with dependency-directed backtracking, plan modification operators
1979	OPM	Hayes-Roth and Hayes-Roth	used blackboard structure and island-driven planning approach
1980	MOLGEN	Stefik	introduced layers of planning control: strategy, design, planning
1983	SIPE	Wilkins	concept of "resources" in preconditions; first to deal with replanning
1987	TWEAK	Chapman	provably correct and complete; introduces constraint posting, modal truth criterion
1991	SNLP	Soderland and Weld	sound, complete, systematic nonlinear planner; uses concept of lifting; basis of modern nonlinear planning systems
1991	O-PLAN	Currie and Tate	can handle time, resources, hierarchical planning, uses heuristics in search, justifies reasons for each choice made
1992	UCPOP	Penberthy and Weld	hierarchical, nonlinear, multiagent, conditional effects, et al.