# Planning and Approximate Reasoning

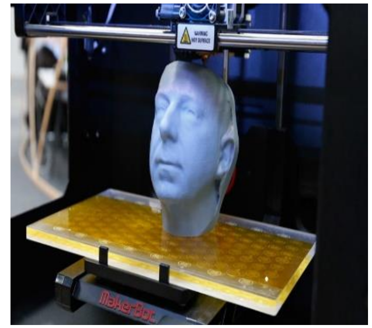
Hatem A. Rashwan

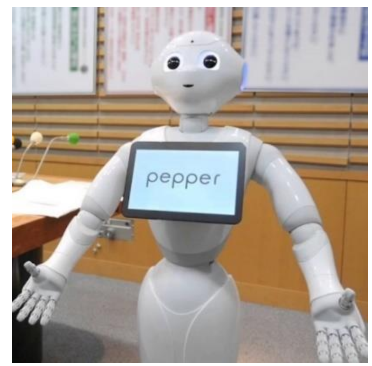
# Representation for High Level Planning

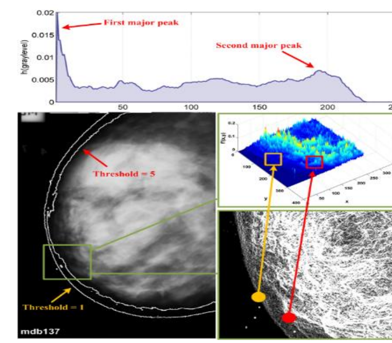
MESIIA – MIA

# About the instructor

- **Research Interests: c**
  - Computer Vision,
  - Pattern Recognition
  - Machine Learning
  - Artificial Intelligence
- **Applications**:
  - Vision-based robotic systems
  - Scene understanding
  - Productivity






- Research Group: [http://deim.urv.cat/~rivi/](http://deim.urv.cat/~rivi/)
  - The IRCV group is constituted by faculty from the Department of Computer Science and Mathematics (DEIM) and the Department of Electrical, Electronic and Automation Engineering (DEEEA). Both departments are physically located at the School of Engineering (ETSE) in Tarragona(Catalonia-Spain).

# What is Planning?

- *Planning is the process of thinking about an organizing the activities required to achieve a desired goal.''*

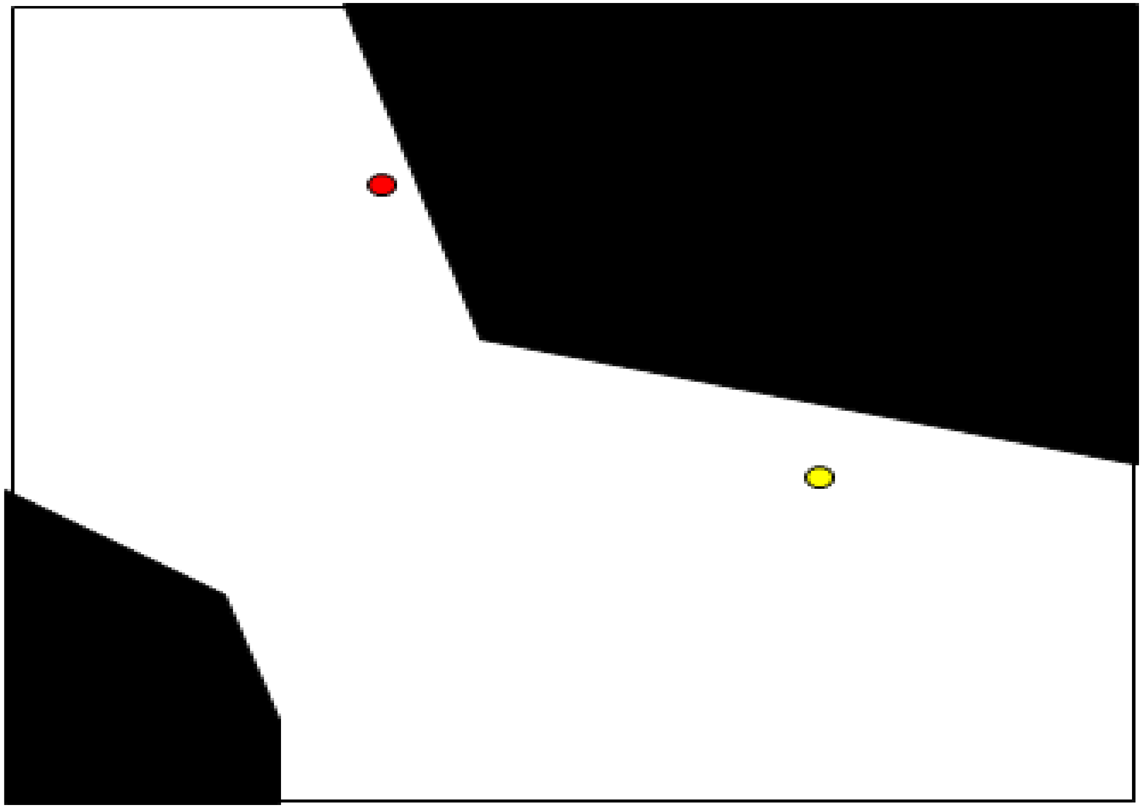
- **Given:**
  - ✓ *Model (states and actions)of the agent(s)* $\boldsymbol{M^a}$**=<**$\boldsymbol{S^a}$**,** $\boldsymbol{A^a}$**>**
  - ✓ *A model of the world* $M_w$
  - ✓ *Its actions*
  - ✓ *Belief* $\boldsymbol{b_c^a}$ *of the agent about its current state*
  - ✓ *Belief* $\boldsymbol{b_c^w}$ *of the agent about the current state of the world*
  - ✓ *Belief of the agent over the cost function* ***C*** *of its actions*

- ***Compute a plan π that:***
  - ✓ *Maps one or more belief tuples* <$\boldsymbol{b^a}$**,**$\boldsymbol{b^w}$> *on to actions* $\boldsymbol{a}$ *in* $\boldsymbol{A^a}$
  - ✓ *Reaches one of the desired states in* ***G***

# Planning Examples

## ❑ 2D path planning for omnidirectional robot

- What is $M^a$ ?
- What is $b_c^a$?
- What is $b_c^w$?
- What is $C$ ?
- What is $G$?



# Planning Examples

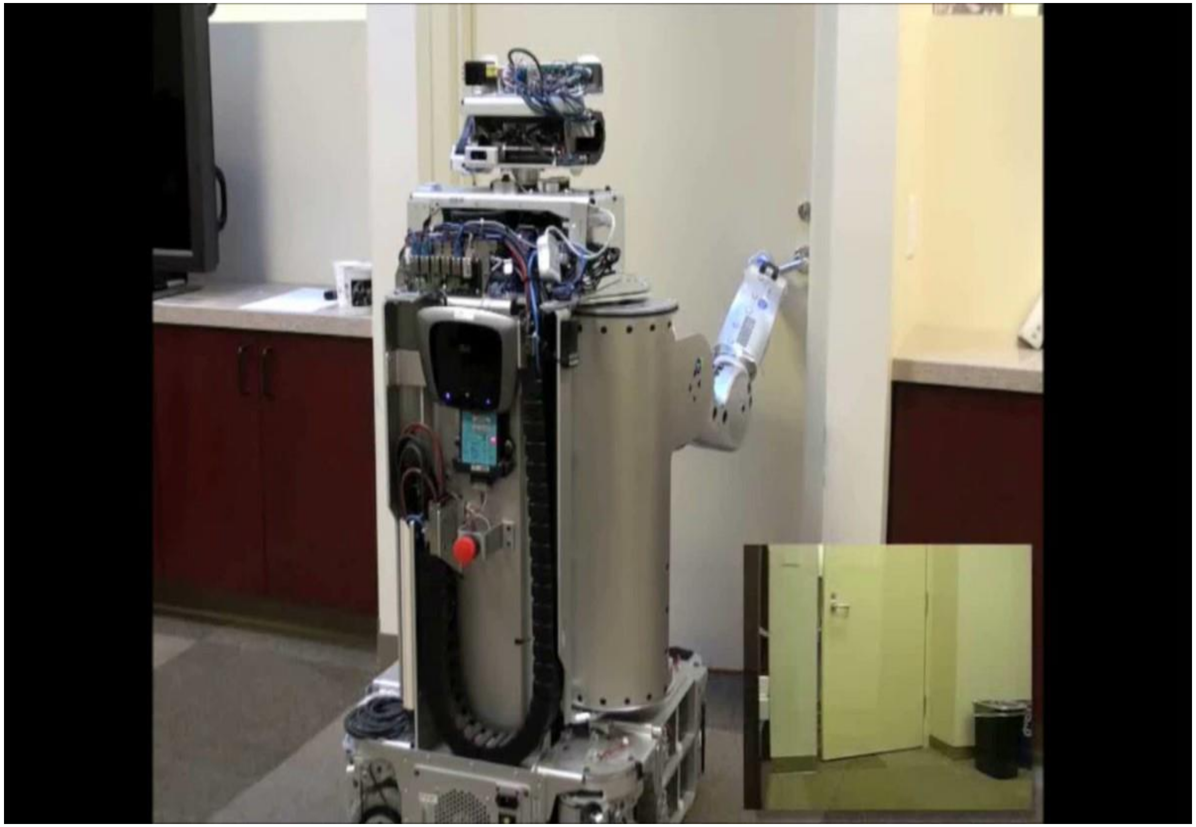
- **5D (x,y,z,direction,time) path planning for autonomous flight among people :**

  - What is $M^a$ ?
  - What is $b_c^a$?
  - What is $b_c^w$?
  - What is $C$ ?
  - What is $G$?

# Planning Examples

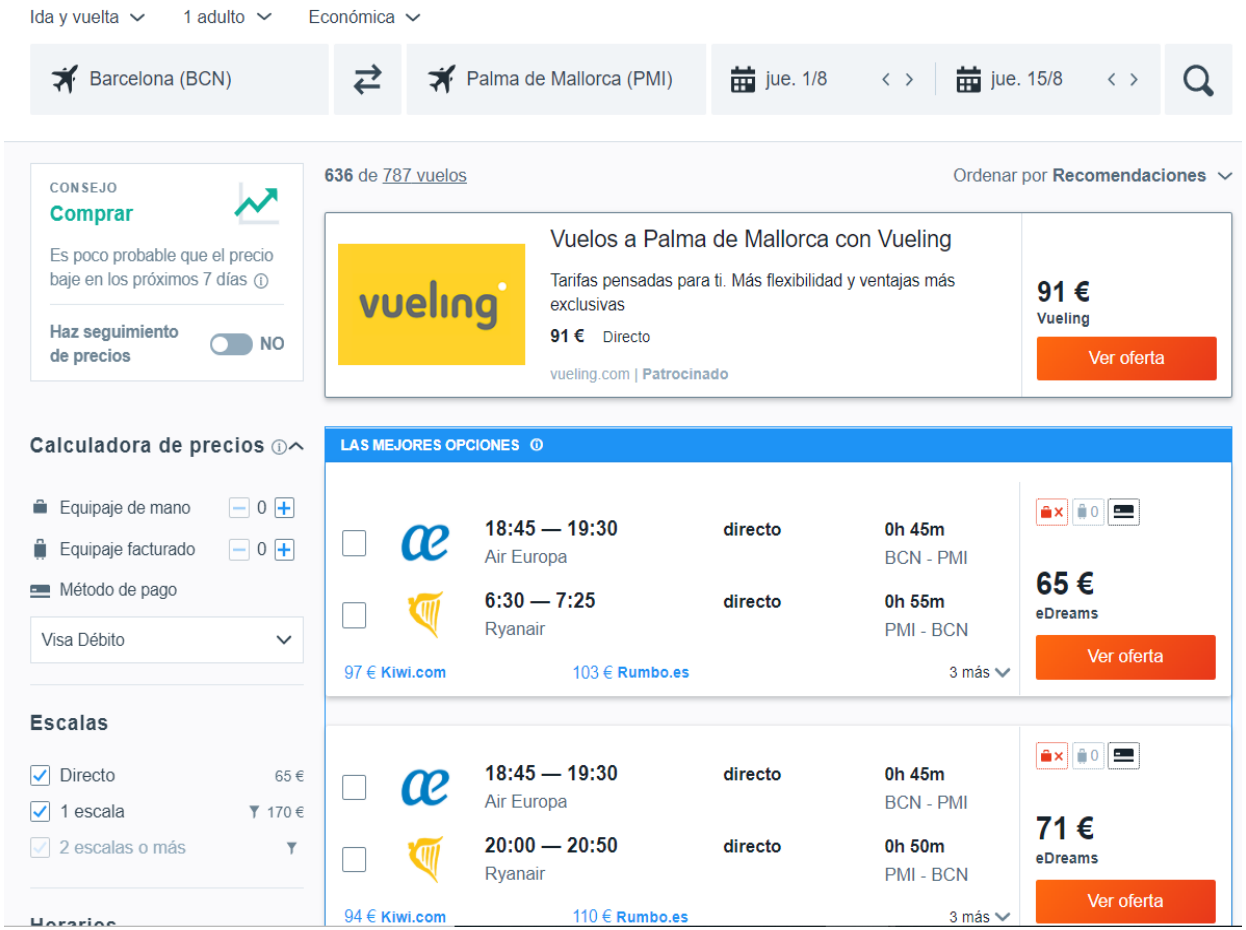
- **Motion planning for a mobile manipulator PR2 opening a door**

  - What is $M^a$ ?
  - What is $b_c^a$?
  - What is $b_c^w$?
  - What is $C$ ?
  - What is $G$?



# Planning Examples

- Planning a travel from Barcelona to Palma Mallorca

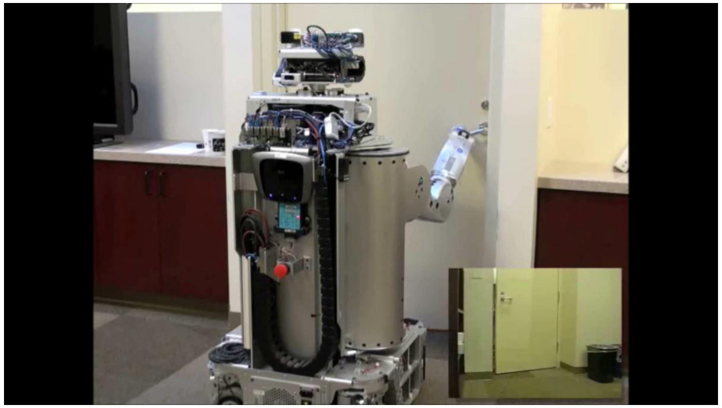



  - What is $M^a$ ?
  - What is $b_c^a$?
  - What is $b_c^w$?
  - What is $C$ ?
  - What is $G$?

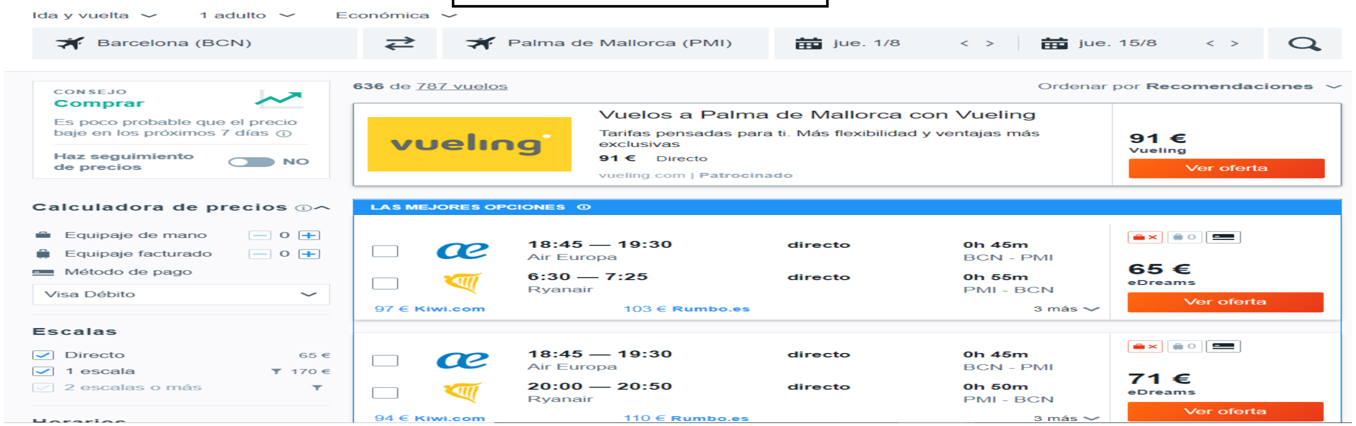
# Continuous vs. Discrete vs. Hybrid Model

Continuous

Hybrid



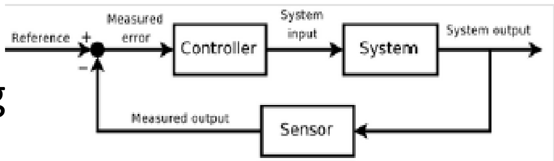
Discrete

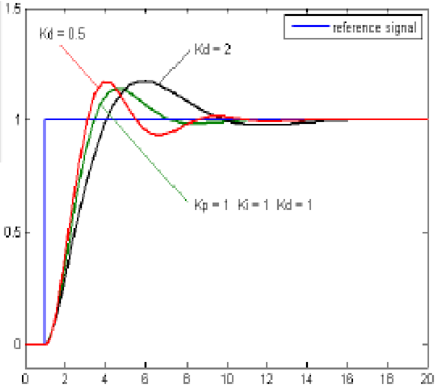


# Planning vs. Control

Local planning

Global planning




*controller*

*Images from wikipedia*

# Some of the topics covered in class

- Representation for higher level planning
- Algorithms for classical planning (Linear Planner)
- Planning Graphs and Heuristic Search
- Planning under uncertainty: MDPs and POMDPs
- Planning under uncertainty: Reinforcement Learning
- Application: planning for mobile manipulation and articulated robots

# Books covered the topics

**Books**

- Automated planning theory and practice, http://homes.dcc.ufba.br/~thiagob052/AI%20Planning/livro-recomendado.pdf **Chapters 1,2,4,6,9, and 20**
- Automated planning and acting, http://projects.laas.fr/planning/book.pdf, **Chapters 6 and 7**

**Books for PDDL**

- An Introduction to the Planning Domain Definition Language (PDDL), https://courses.cs.washington.edu/courses/cse473/06sp/pddl.pdf, http://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf

**Planning software**

For Windows:

- Visual studio Code with PDDL packages, https://marketplace.visualstudio.com/items?itemName=jan-dolejsi.pddl

For Linux:

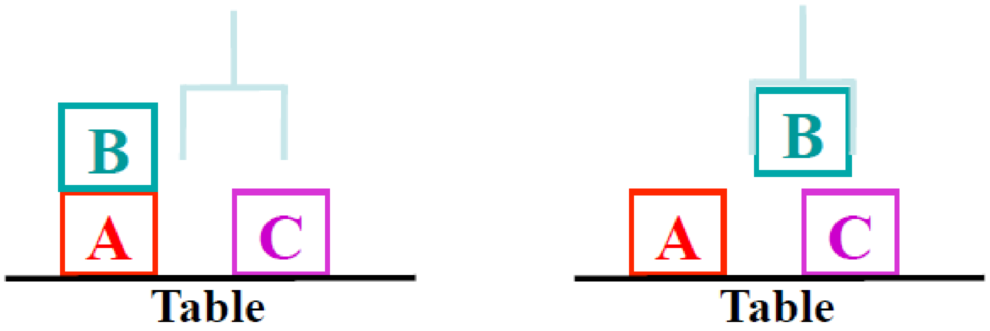
- FF (Fast-Forward) Planning Software: http://www.ai.mit.edu/courses/16.412J/ff.html
- Graphplan Planning Software: http://www.ai.mit.edu/courses/16.412J/Graphplan.html

# State, Action, Goal Representation; Classical Planning

- **Planning – Problem Solving** (as defined in Newell and Simon 1956)
  - Given the actions available in a task domain.
  - Given a problem specified as:
    - an initial state of the world,
    - goal statement - a set of goals to be achieved.
  - Find a solution to the problem,
    - a way to transform the initial state into a new state of the world where the goal statement is true.
  - Planning is "thinking…"

# Planning-Problem solving

- What is a State and Goal
- What is an Action?
- What is a Plan?
- Finding a Plan !!!

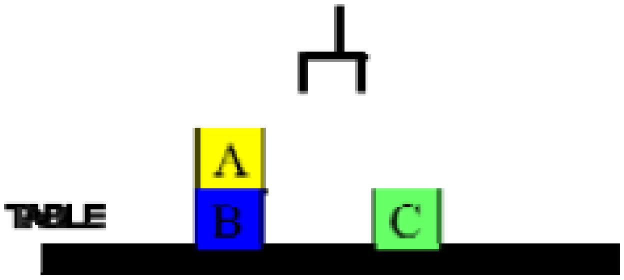



What is the Blocks World? -- The world consists of:

- ❑ A flat surface such as a tabletop
- ❑ An adequate set of identical blocks which are identified by letters (A,B,C).
- ❑ The blocks can be stacked one on one to form towers of apparently unlimited height.
- ❑ The stacking is achieved using a robot arm which has fundamental operations and states, which can be assessed using logic and combined using logical operations.
- ❑ The robot can hold one block at a time and only one block can be moved at a time.

# The Blocks World

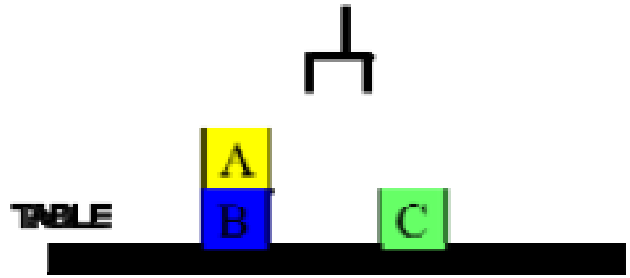
## ➢What is a State and Goal?




- We'll illustrate the techniques with reference to the blocks world
- This world contains
  - a robot arm with gripper,
  - 3 blocks (A, B and C) of equal size,
  - a table-top.
- Some domain constraints:
  - Only one block can be directly on top of another block
  - Any number of blocks can be on the table
  - The hand can only hold one block

# The Blocks World

## ➢ What is a State and Goal?



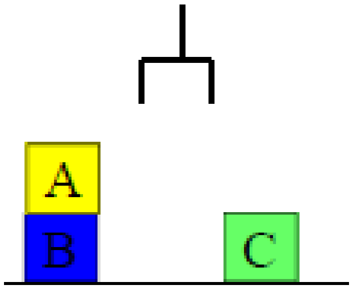

To represent this environment, we need an Ontology

- ❑ **On(x,)** *means block x* is on top of *block y*
- ❑ **OnTable(x)** --- *block x* is on the table
- ❑ **Clear(x)** --- nothing is on top of *block x*
- ❑ **Holding(x)** --- *robot* arm is holding *block x*
- ❑ **ArmEmpty()** --- *robot* arm/hand is not holding anything (block in this world)

# Blocks World State and Goal Description

## ➢ State Representation = Environment

- A representation of one state of the blocks world.
  The state in the figure is:
  - *Clear(A)*
  - *Clear(C)*
  - *On(A,B)*
  - *OnTable(B)*
  - *OnTable(C)*
  - *ArmEmpty()*




- Use the *closed world assumption*: *anything not stated is assumed to be false*

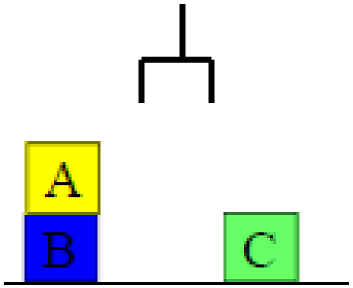
## ➢ Goal Representation

- A *goal* is represented as a set of formulae. Here is a goal:
  *OnTable(A)*
  *OnTable(B)*
  *OnTable(C)*

# Blocks World Actions Description

## Actions

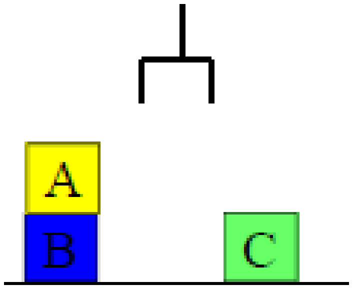
- Represented using a technique that was developed in the STRIPS planner. Each action has:
  - a **name** **---**which may have arguments;
  - a **pre-condition list** --- a list of facts which must be true for action to be executed;
  - a **delete list** --- a list of facts that are no longer true after action is performed;
  - an **add list** --- a list of facts made true by executing the action.
  - Each of the facts may contain **variables**



# Blocks World Actions Description

## Action/Operator Representation

- Basic operations
  - stack(X,Y): put block X on block Y
  - unstack(X,Y): remove block X from block Y
  - pickup(X): pickup block X from the table
  - putdown(X): put block X on the table

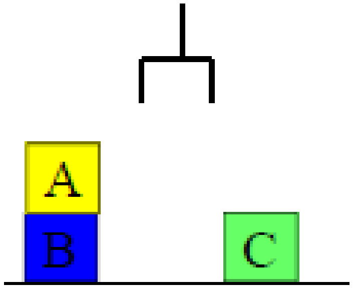



- Each operator is represented by facts that describe the state of the world before and changes to the world after an action is performed.
  - a list of **preconditions**
  - a list of new **facts to be added** (add-effects)
  - a list of **facts to be removed** (delete-effects)
  - optionally, a set of (simple) variable **constraints**

# Blocks World Actions

## Stack Operator

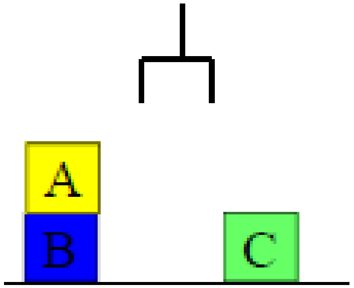
- The **stack** action occurs when the robot arm places the object it is holding [x] on top of another object [*y*]
- Form: *Stack(x,y)*
- Pre: $Clear(y) \land Holding(x)$
- Add: $ArmEmpty \land On(x,y) \land Clear(x)$
- Del: $Clear(y) \land Holding(x)$
- Constraints: $(x \neq y), x \neq Table, y \neq Table$



# Blocks World Actions

## Unstack Operator

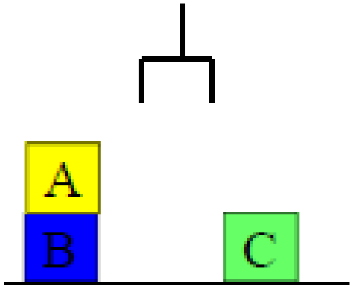
- The **unstack** action occurs when the robot arm picks up an object *x* from on top of another object *y*.
- Form: *UnStack(x,y)*
- Pre: $On(x,y) \land Clear(x) \land ArmEmpty()$
- Add: $Holding(x) \land Clear(y)$
- Del: $On(x,y) \land Clear(x) \land ArmEmpty()$
- Constraints: $x \neq y$, $x \neq Table$, $y \neq Table$



# Blocks World Actions

## Pickup Operator

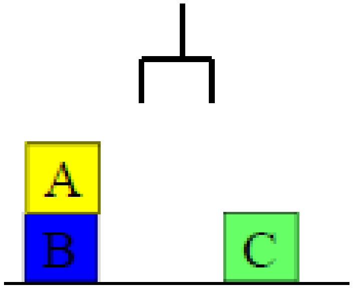
- The **pickup** action occurs when the arm picks up an object (block) from the table
- Form: *Pickup(x)*
- Pre: $OnTable(x) \land Clear(x) \land ArmEmpty()$
- Add: $Holding(x)$
- Del: $OnTable(x) \land Clear(x) \land ArmEmpty()$
- Constraints: $x \neq table$



# Blocks World Actions

## Putdown Operator

- The **putdown** action occurs when the arm places the object *x* onto the table
- Form: $PutDown(x)$
- Pre: $Holding(x)$
- Add: $OnTable(x) \wedge ArmEmpty \wedge Clear(x)$
- Del: $Holding(x)$
- Constraints: $x \neq table$



# Planning and Agents

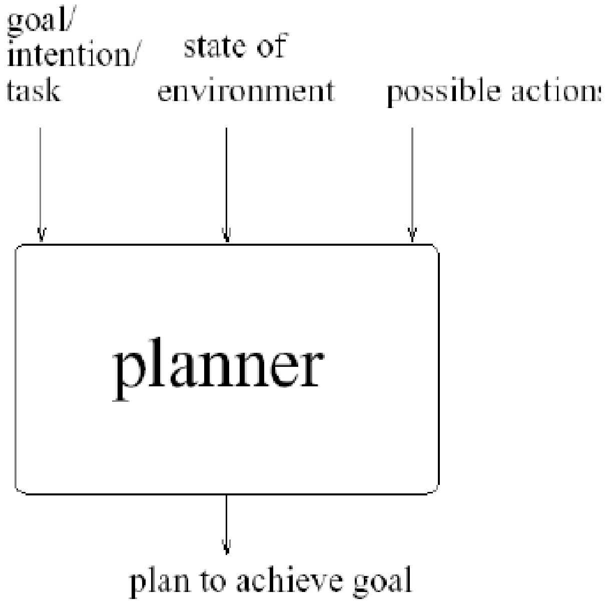
- Since the early 1970s, the AI planning community has been closely concerned with the design of artificial agents
- Planning is essentially *automatic programming*: the design of a course of action that will achieve some desired goal
- Within the symbolic AI community, it has long been assumed that some form of AI planning system will be a central component of any artificial agent
- Building largely on the early work of Fikes & Nilsson, many planning algorithms have been proposed, and the theory of planning has been well-developed

# Planning and Agents

**Means-Ends Reasoning**

- Idea is to give an *agent*:
  - representation of goal/intention to achieve;
  - representation of actions it can perform; and
  - representation of the environment;
- Then have the agent **generate a plan to achieve the goal.**

The plan is generated entirely by the planning system, without human intervention.



# STRIPS Planning

- STRIPS maintains two additional data structures:
  - *State List* - all currently true predicates.
  - *Goal Stack* - a push down stack of goals to be solved, with current goal on top of stack.

*If the current goal is not satisfied* by present state,

- Find *goal* in the *add list of an operator*, and

*push operator and preconditions list on stack*. (=Subgoals)

- When a *current goal is satisfied*, *POP* it from stack.
- When an *operator is on top of the stack*,
  - record the application of that *operator – update the plan* sequence, and
  - use the operator’s add and delete lists to *update the current state*

# Planning in STRIPS

- Uses means-ends reasoning (actions = means, goals = ends)
- States of the world and goals are represented as a set/list of predicates that are true (e.g. on(x,y) ..)

1. The current state is initialized to the start state
2. The goal is placed on the goal stack
3. Loop through the following steps to produce a plan

**If the top item on the goal stack is:**

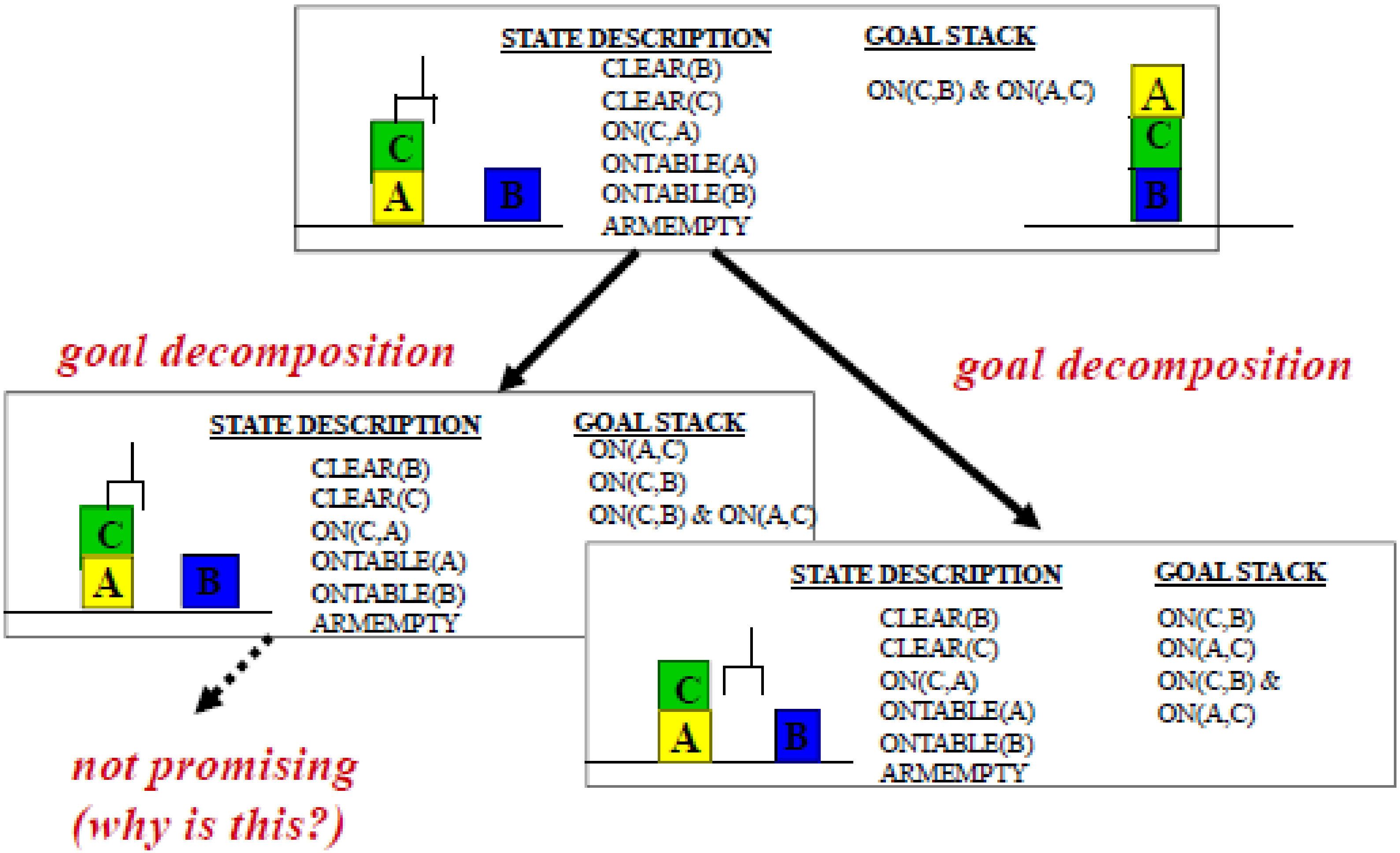
- empty (the goal stack is empty), return the actions executed – they form the plan to achieve the goal
- a goal, and it is satisfied in the current state, remove it from the stack (no replacement necessary)

# Planning in STRIPS

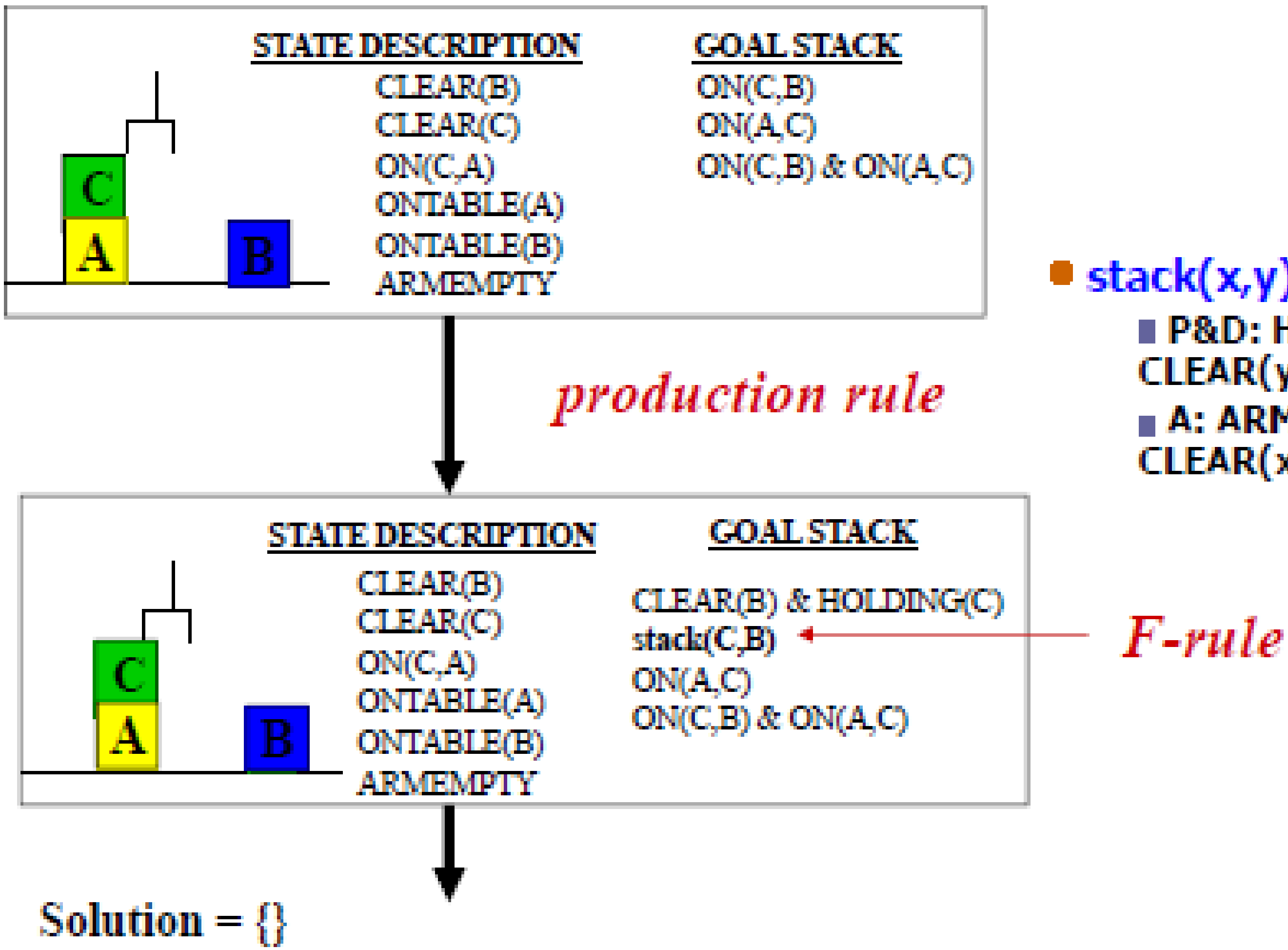
**If the top item on the goal stack is:**

- empty (the goal stack is empty), return the actions executed – they form the plan to achieve the goal
- a goal, and it is satisfied in the current state, remove it from the stack (no replacement necessary)
- a complex goal, break it into subgoals, placing all subgoals on the goal stack (the original goal is pushed down in the goal stack)
- a predicate, find an action that will make it true, then place that action (with variables bound appropriately) and its preconditions on the goal stack (preconditions first)
- an action and its preconditions are satisfied, perform the action, updating the world state using the delete and add lists of the action Add this action to the partial plan

# STRIPS in action

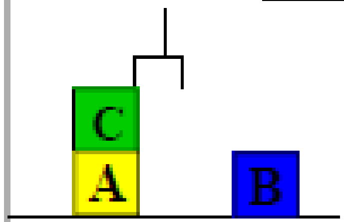


# STRIPS in action



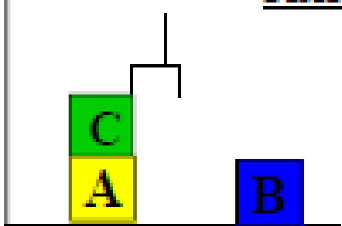

- stack(x,y)
  - P&D: HOLDING(x), CLEAR(y)
  - A: ARMEMPTY, ON(x,y), CLEAR(x)

# STRIPS in action




| STATE DESCRIPTION | GOAL STACK |
|---|---|
| CLEAR(B) | CLEAR(B) & HOLDING(C) |
| CLEAR(C) | stack(C,B) |
| ON(C,A) | ON(AC,) |
| ONTABLE(A) | ON(C,B) & ON(A,C) |
| ONTABLE(B) | |
| ARMEMPTY | |

*production rule*

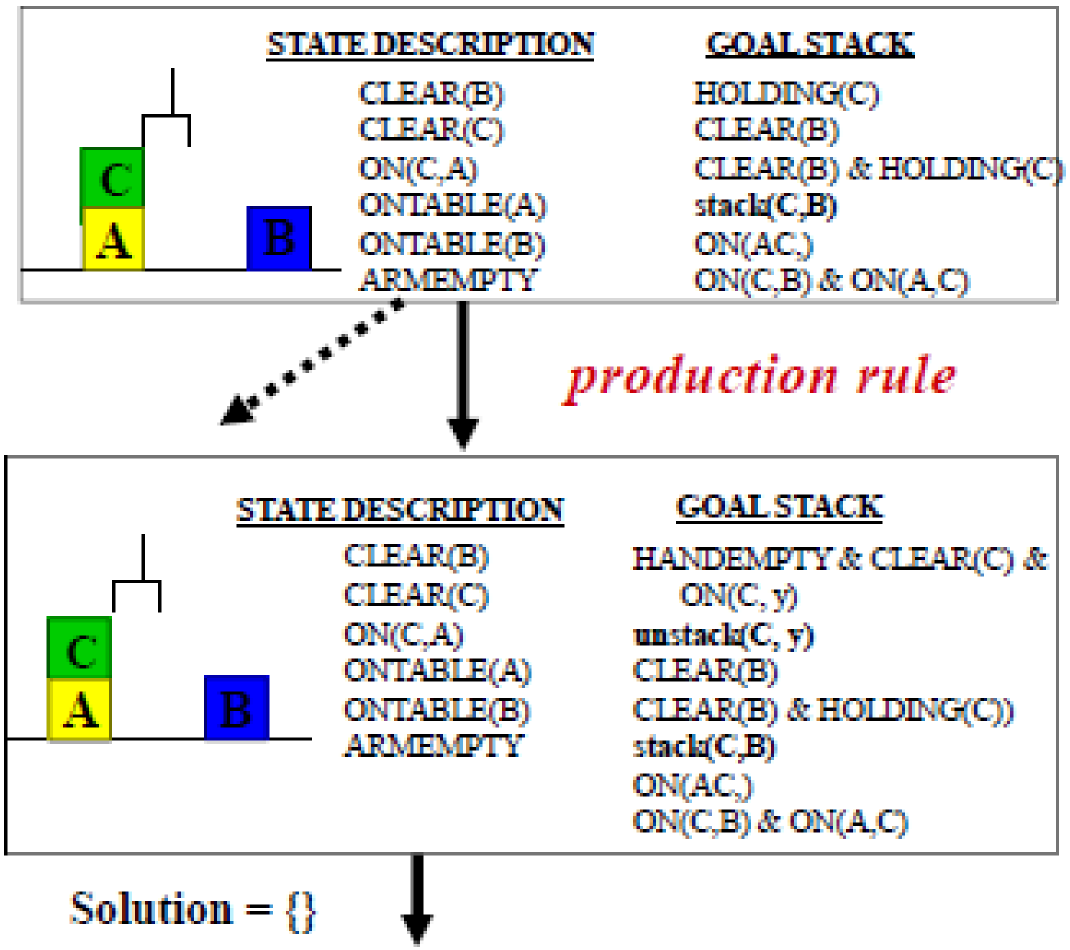



| STATE DESCRIPTION | GOAL STACK |
|---|---|
| CLEAR(B) | HOLDING(C) |
| CLEAR(C) | CLEAR(B) |
| ON(C,A) | CLEAR(B) & HOLDING(C) |
| ONTABLE(A) | stack(C,B) |
| ONTABLE(B) | ON(AC,) |
| ARMEMPTY | ON(C,B) & ON(A,C) |

*goal decomposition*

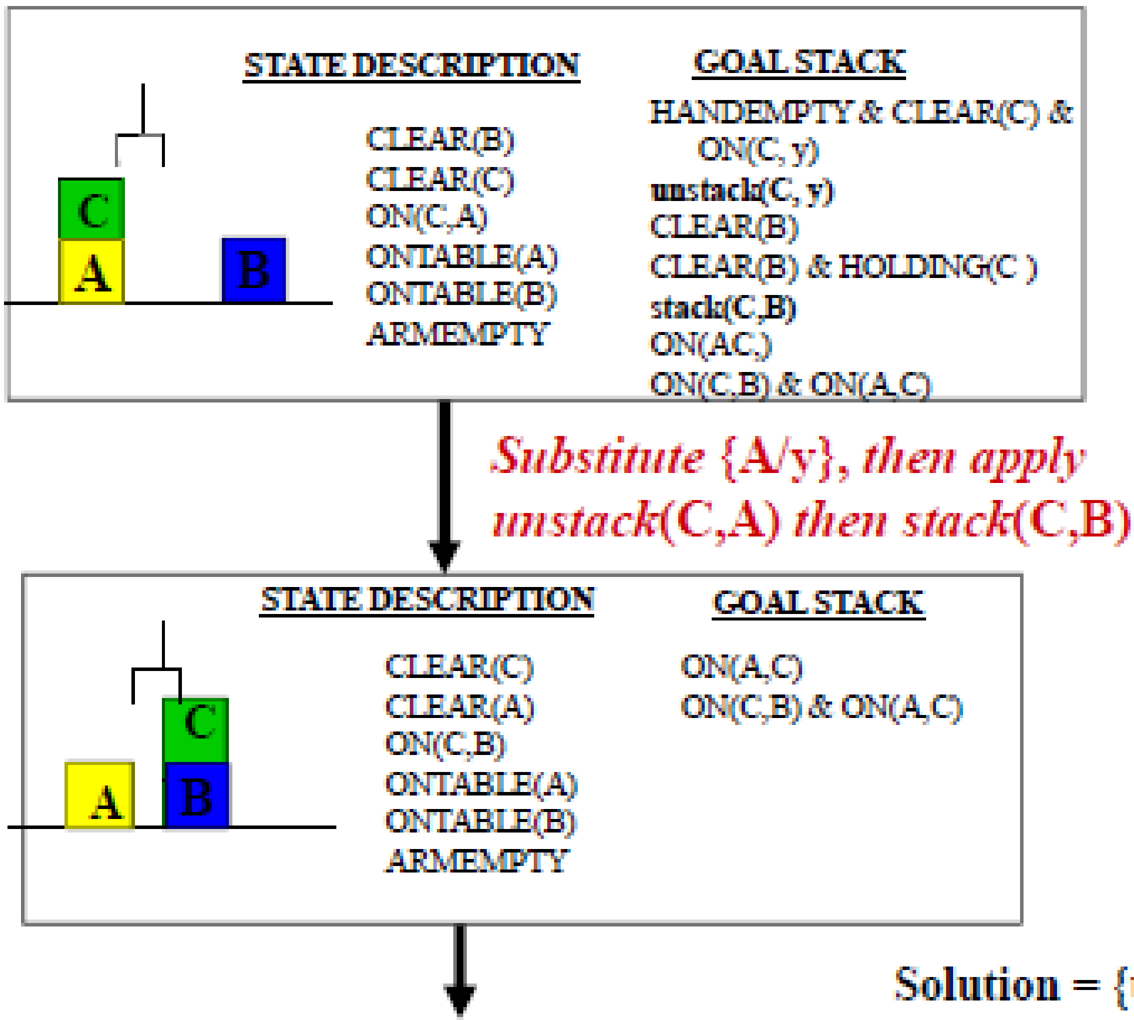
Solution = {}

# STRIPS in action




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

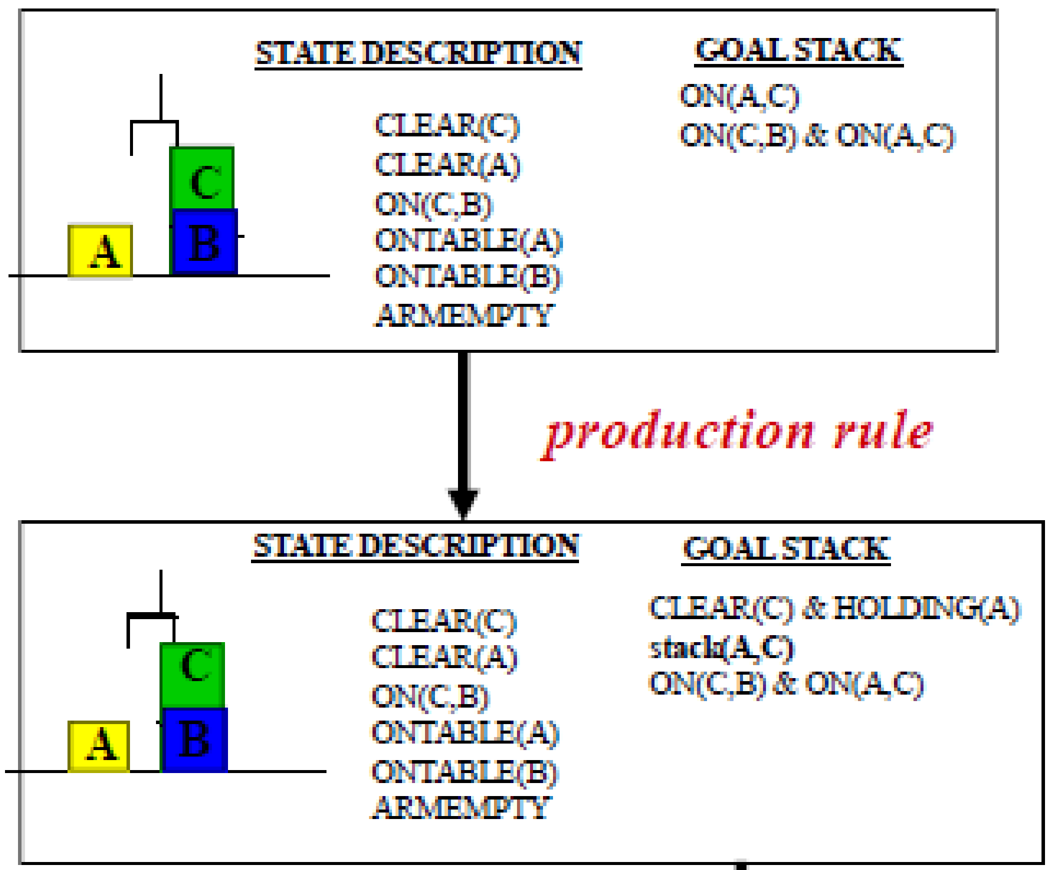
# STRIPS in action




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

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

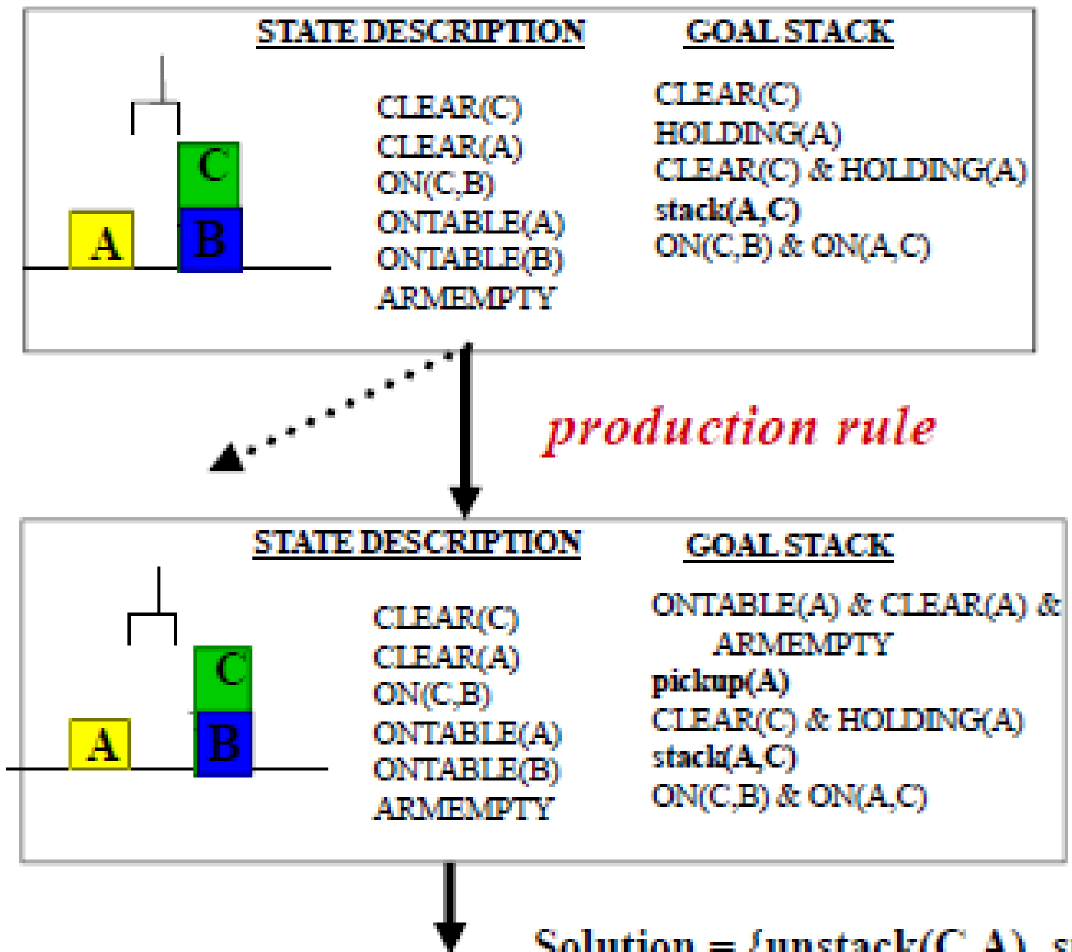
# STRIPS in action




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

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

# STRIPS in action

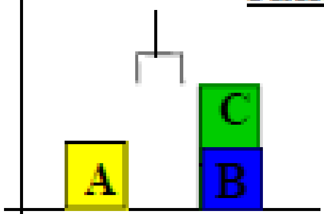



*goal decomposition*

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

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

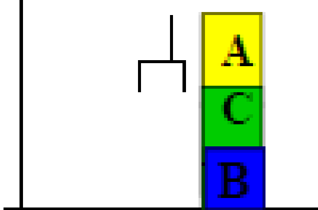
# STRIPS in action




| STATE DESCRIPTION | GOAL STACK |
|---|---|
| CLEAR(C)<br>CLEAR(A)<br>ON(C,B)<br>ONTABLE(A)<br>ONTABLE(B)<br>ARMEMPTY | ONTABLE(A) & CLEAR(A) & ARMEMPTY<br>**pickup(A)**<br>CLEAR(C) & HOLDING(A)<br>**stack(A,C)**<br>ON(C,B) & ON(A,C) |

*Apply pickup(A) and then stack(A,C)*

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




| STATE DESCRIPTION | GOAL STACK |
|---|---|
| ON(A,C)<br>ON(C,B)<br>CLEAR(A)<br>ONTABLE(B)<br>ARMEMPTY | NIL |

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

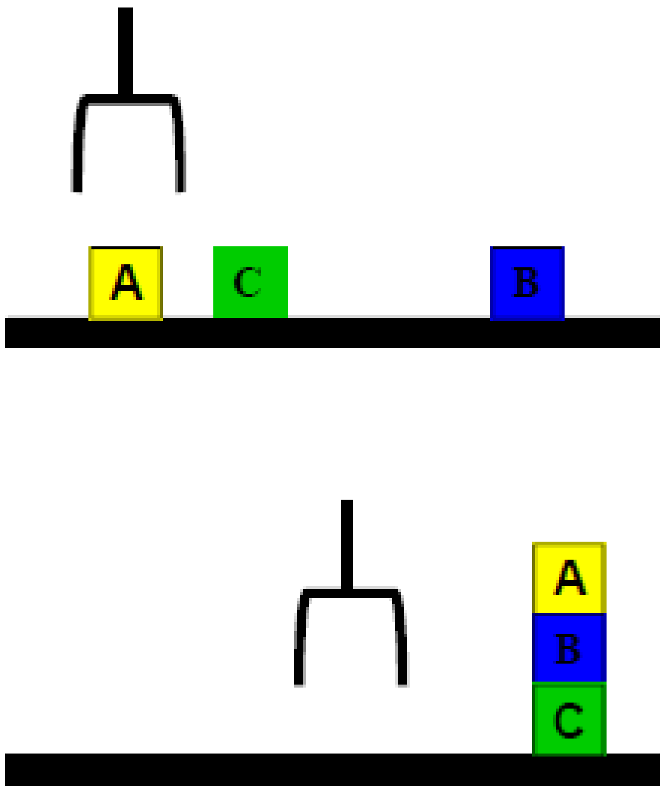
# Typical BW Planning Problem

Initial state:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Goal:

on(b,c)
on(a,b)
ontable(c)




A plan:

pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

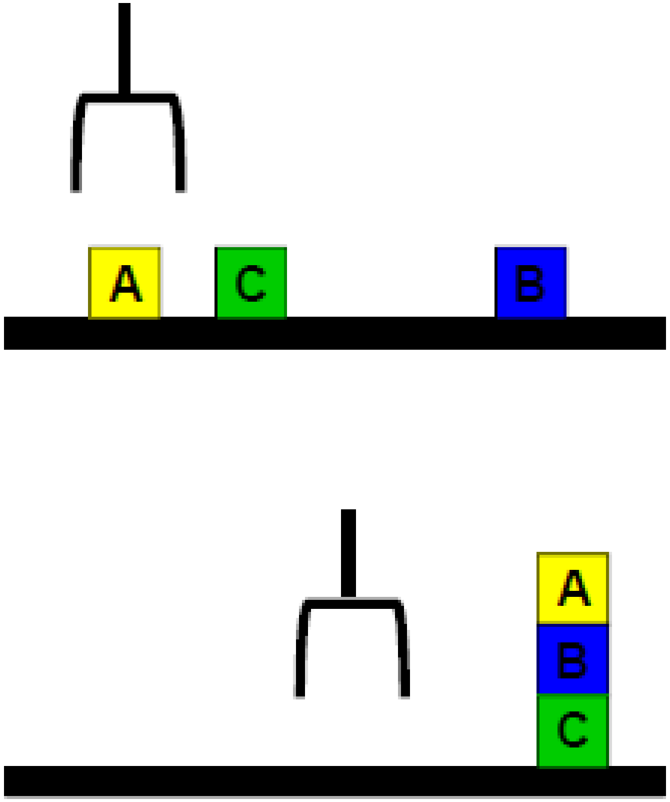
# Typical BW Planning Problem

Initial state:

- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:

- on(a,b)
- on(b,c)
- ontable(c)

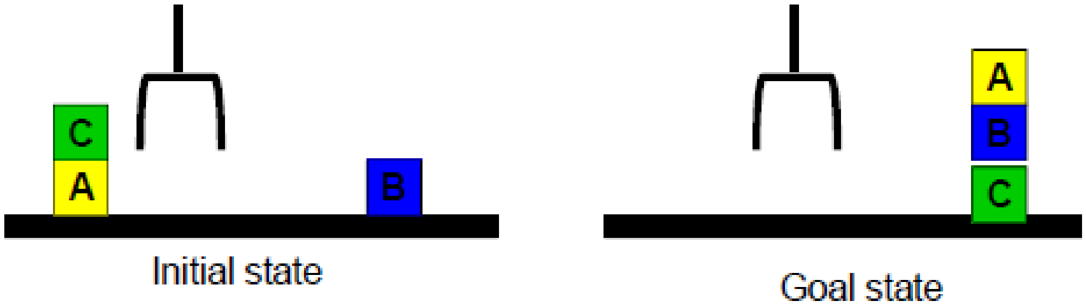



A plan:

- pickup(a)
- stack(a,b)

- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

# Goal Interaction




- Simple planning algorithms assume that the goals to be achieved are independent
  - Each can be solved separately and then the solutions concatenated
- This planning problem, called the "Sussman Anomaly," is the classic example of the goal interaction problem:
  - Solving on(A,B) first (by doing unstack(C,A), stack(A,B) will be undone when solving the second goal on(B,C) (by doing unstack(A,B), stack(B,C)).
  - Solving on(B,C) first will be undone when solving on(A,B)
- Classic STRIPS could not handle this, although minor modifications can get it to do simple cases

# End