Chapter 10
Planning

Instructor
LE Thanh Sach, Ph.D.

Instructor's Information

LE Thanh Sach, Ph.D.

Office:

Department of Computer Science,

Faculty of Computer Science and Engineering,

HoChiMinh City University of Technology.

Office Address:

268 LyThuongKiet Str., Dist. 10, HoChiMinh City, Vietnam.

E-mail: LTSACH@cse.hcmut.edu.vn

E-home: http://cse.hcmut.edu.vn/~ltsach/

Tel: (+84) 83-864-7256 (Ext: 5839)

Acknowledgment

The slides in this PPT file are composed using the materials supplied by

- **Prof. Stuart Russell and Peter Norvig**: They are currently from University of California, Berkeley. They are also the author of the book “Artificial Intelligence: A Modern Approach”, which is used as the textbook for the course

- **Prof. Tom Lenaerts**, from Université Libre de Bruxelles

Planning

- The Planning problem
- Planning language
- Planning with State-space search
- Stack of goals

What is Planning

- Generate sequences of actions to perform tasks and achieve objectives.
  - States, actions and goals
- Search for solution over abstract space of plans.
- Assists humans in practical applications
  - design and manufacturing
  - military operations
  - games
  - space exploration

What is Planning? Difficulty of real world problems

- Assume a problem-solving agent using some search method …
  - Which actions are relevant?
    - ✓ Exhaustive search vs. backward search
  - What is a good heuristic functions?
    - ✓ Good estimate of the cost of the state?
    - ✓ Problem-dependent vs, -independent
  - How to decompose the problem?
    - ✓ Most real-world problems are ***nearly*** decomposable.

Planning language

- What is a good language?
  - Expressive enough to describe a wide variety of problems.
  - Restrictive enough to allow efficient algorithms to operate on it.
  - Planning algorithm should be able to take advantage of the logical structure of the problem.

- STRIPS and ADL

Planning language: General language features

- Representation of states
  - Decompose the world in logical conditions and represent a state as a *conjunction of positive literals.*
    - ✓ Propositional literals: *Poor* $\wedge$ *Unknown*
    - ✓ First Order (FO)-literals (grounded and function-free): *At(Plane1, Melbourne)* $\wedge$ *At(Plane2, Sydney)*
  - Closed world assumption
- Representation of goals
  - Partially specified state and represented as a *conjunction of positive ground literals*
  - A goal is *satisfied* if the state contains all literals in goal.

Planning language: General language features

- Representations of actions
  - Action = PRECOND + EFFECT

    Action(Fly(p,from, to),

    PRECOND: At(p,from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)

    EFFECT: ¬AT(p,from) ∧ At(p,to))

  = action schema (p, from, to need to be instantiated)
    - ✓ Action name and parameter list
    - ✓ Precondition (conj. of function-free literals)
    - ✓ Effect (conj of function-free literals and P is True and not P is false)
  - Add-list vs delete-list in Effect

Planning language: Language semantics?

- How do actions affect states?
  - An action is applicable in any state that satisfies the precondition.
  - For FO action schema applicability involves a substitution θ for the variables in the PRECOND.

$At(P1,JFK) \wedge At(P2,SFO) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge Airport(SFO)$

Satisfies : $At(p,from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

With $\theta = \{p/P1, from/JFK, to/SFO\}$

Thus the action is applicable.

Planning language: Language semantics?

- The result of executing action a in state s is the state s'
  - s' is same as s except
    - ✓ Any positive literal *P* in the effect of *a* is added to *s'*
    - ✓ Any negative literal *¬P* is removed from *s'*

At(P1,SFO) ∧ At(P2,SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)

  - STRIPS assumption: (avoids representational frame problem)

    every literal NOT in the effect remains unchanged

Planning language: Expressiveness and extensions

- STRIPS is simplified
  - Important limit: function-free literals
  - Allows for propositional representation
- Function symbols lead to infinitely many states and actions
- Recent extension:Action Description language (ADL)

  Action(Fly(p:Plane, from: Airport, to: Airport),
  PRECOND: At(p,from) $\wedge$ *(from* $\neq$ *to)*
  EFFECT: $\neg$*At(p,from)* $\wedge$ *At(p,to))*

Standardization : *Planning domain definition language (PDDL)*

Example: air cargo transport

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

Goal(At(C1,JFK) ∧ At(C2,SFO))

Action(Load(c,p,a)

PRECOND: *At(c,a) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*

EFFECT: *¬At(c,a) ∧In(c,p))*

Action(Unload(c,p,a)

PRECOND: *In(c,p) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*

EFFECT: *At(c,a) ∧ ¬In(c,p))*

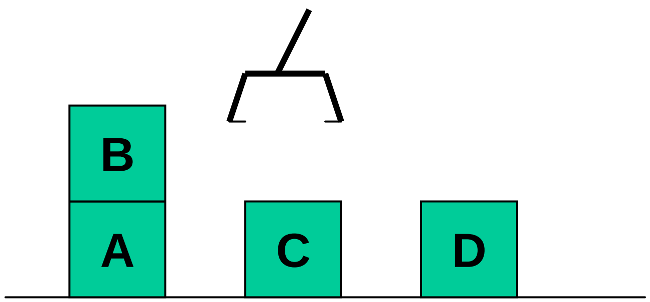
Action(Fly(p,from,to)

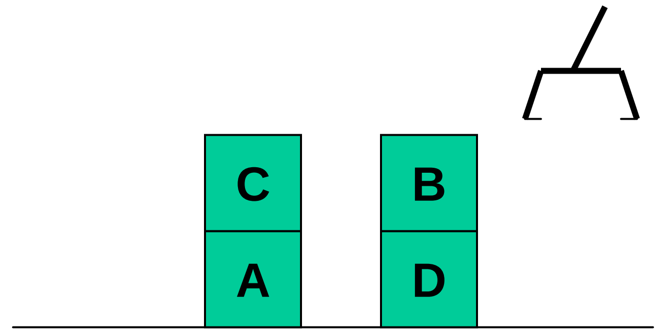
PRECOND: *At(p,from) ∧Plane(p) ∧Airport(from) ∧Airport(to)*

EFFECT: *¬ At(p,from) ∧ At(p,to))*

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

Example: Blocks world





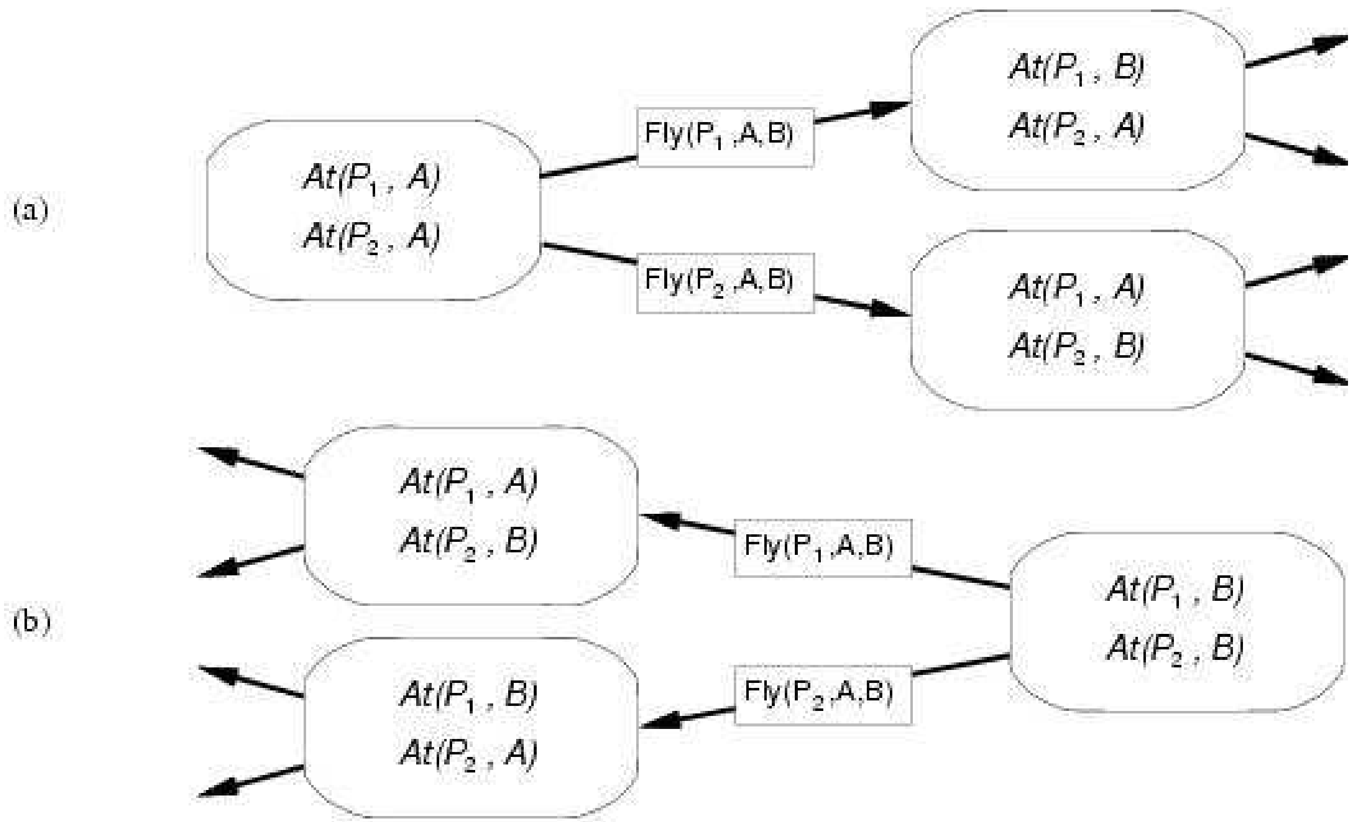
START:
ON(B,A) ^
ONATBLE(A) ^
ONATBLE(C) ^
ONATBLE(D) ^
CLEAR(B) ^
CLEAR(C) ^
CLEAR(D) ^
AMREMPTY

GOAL:
ON(C,A) ^
ON(B,D) ^
ONATBLE(A) ^
ONATBLE(D) ^
CLEAR(C) ^
CLEAR(B) ^
AMREMPTY

Planning with state-space search

- Both forward and backward search possible
- Progression planners
  - forward state-space search
  - Consider the effect of all possible actions in a given state
- Regression planners
  - backward state-space search
  - To achieve a goal, what must have been true in the previous state.

Progression and regression



Progression algorithm

- Formulation as state-space search problem:
  - Initial state = initial state of the planning problem
    - ✓ Literals not appearing are false
  - Actions = those whose preconditions are satisfied
    - ✓ Add positive effects, delete negative
  - Goal test = does the state satisfy the goal
  - Step cost = each action costs 1
- No functions … any graph search that is complete is a complete planning algorithm.
- Inefficient: (1) irrelevant action problem (2) good heuristic required for efficient search

Regression algorithm

- How to determine predecessors?
  - What are the states from which applying a given action leads to the goal?
    - Goal state = $At(C1, B) \wedge At(C2, B) \wedge \ldots \wedge At(C20, B)$
    - Relevant action for first conjunct: $Unload(C1,p,B)$
    - Works only if pre-conditions are satisfied.
    - Previous state= $In(C1, p) \wedge At(p, B) \wedge At(C2, B) \wedge \ldots \wedge At(C20, B)$
    - Subgoal At(C1,B) should not be present in this state.
- Actions must not undo desired literals (consistent)
- Main advantage: only relevant actions are considered.
  - Often much lower branching factor than forward search.

Regression algorithm

- General process for predecessor construction
  - Give a goal description G
  - Let A be an action that is relevant and consistent
  - The predecessors is as follows:
    - ✓ Any positive effects of A that appear in G are deleted.
    - ✓ Each precondition literal of A is added , unless it already appears.
- Any standard search algorithm can be added to perform the search.
- Termination when predecessor satisfied by initial state.
  - In FO case, satisfaction might require a substitution.

Heuristics for state-space search

- Neither progression or regression are very efficient without a good heuristic.
  - How many actions are needed to achieve the goal?
  - Exact solution is NP hard, find a good estimate
- Two approaches to find admissible heuristic:
  - The optimal solution to the relaxed problem.
    - ✓ Remove all preconditions from actions
  - The subgoal independence assumption:

    The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.

Block World Example

- Actions List:
  - STACK(X,Y):
    - ✓ Precondition: CLEAR(Y) ^ HOLDING(X)
    - ✓ Delete-List: CLEAR(Y) ^ HOLDING(X)
    - ✓ Add-List: ARMEMPTY ^ ON(X,Y)
  - UNSTACK(X,Y):
    - ✓ Precondition: ON(X,Y) ^ CLEAR(X) ^ ARMEMPTY
    - ✓ Delete-List: ON(X,Y) ^ ARMEMPTY
    - ✓ Add-List: HOLDING(X) ^ CLEAR(Y)

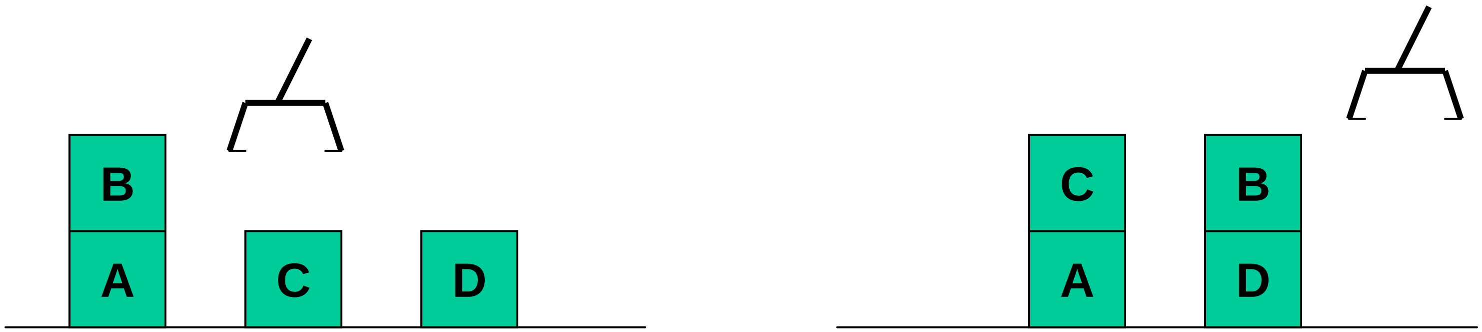
Block World Example

❖ Actions List: (cont.)

- PICKUP(X):
  - ✓ Precondition: CLEAR(X) ^ ONTABLE(X) ^ ARMEMPTY
  - ✓ Delete-List: ONTABLE(X) ^ ARMEMPTY
  - ✓ Add-List: HOLDING(X)

- PUTDOWN(X):
  - ✓ Precondition: HOLDING(X)
  - ✓ Delete-List: HOLDING(X)
  - ✓ Add-List: ONTABLE(X) ^ ARMEMPTY

Block World Example



START:
ON(B,A) ^
ONATBLE(A) ^
ONATBLE(C) ^
ONATBLE(D) ^
CLEAR(B) ^
CLEAR(C) ^
CLEAR(D) ^
AMREMPTY

GOAL:
ON(C,A) ^
ON(B,D) ^
ONATBLE(A) ^
ONATBLE(D) ^
CLEAR(C) ^
CLEAR(B) ^
AMREMPTY

Block World Example

START:
ON(B,A) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(D) ^ AMREMPTY

S0

STACK(X,Y):
Precondition: CLEAR(Y) ^ HOLDING(
Delete-List: CLEAR(Y) ^ HOLDING(
Add-List: ARMEMPTY ^ ON(X,Y)

ON(C,A)
ON(B,D)
~~ONATBLE(A)~~
~~ONATBLE(D)~~
~~CLEAR(C)~~
~~CLEAR(B)~~
~~AMREMPTY~~

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

STACK(C,A)
ON(B,D)

Block World Example

START:
ON(B,A) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(D) ^ AMREMPTY

S0

UNSTACK(X,Y):
Precondition: ON(X,Y) ^ CLEAR(X) ^ ARMEMPTY
Delete-List: ON(X,Y) ^ ARMEMPTY
Add-List: HOLDING(X) ^ CLEAR(Y)

STACK(C,A)
ON(B,D)

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

UNSTACK(B,A)
HOLDING(C)
STACK(C,A)
ON(B,D)

ON(B,A)
CLEAR(B)
ARMEMPTY
UNSTACK(B,A)
HOLDING(C)
STACK(C,A)
ON(B,D)

Block World Example

HOLDING(B) ^ CLEAR(A) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(D)

S1

PICKUP(X):

Precondition:	CLEAR(X) ^ ONTABLE(X) ^ A
Delete-List:	ONTABLE(X) ^ ARMEMPTY
Add-List:	HOLDING(X)

Actions: (1) **UNSTACK(B,A)**

~~ON(B,A)~~
~~CLEAR(B)~~
~~ARMEMPTY~~
UNSTACK(B,A)
HOLDING(C)
STACK(C,A)
ON(B,D)

HOLDING(C)
STACK(C,A)
ON(B,D)

PICKUP(C)
STACK(C,A)
ON(B,D)

Block World Example

HOLDING(B) ^ CLEAR(A) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(D)

S1

PICKUP(X):
Precondition: CLEAR(X) ^ ONTABLE(X) ^ A
Delete-List: ONTABLE(X) ^ ARMEMPTY
Add-List: HOLDING(X)

Actions: (1) **UNSTACK(B,A)**

S1

PICKUP(C)
STACK(C,A)
ON(B,D)

~~CLEAR(C)~~
~~ONTABLE(C)~~
ARMEMPTY
PICKUP(C)
STACK(C,A)
ON(B,D)

ARMEMPTY
PICKUP(C)
STACK(C,A)
ON(B,D)

Block World Example

HOLDING(B) ^ CLEAR(A) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(D)

S1

STACK(X,Y):

Precondition:	CLEAR(Y) ^ HOLDING(X)
Delete-List:	CLEAR(Y) ^ HOLDING(X)
Add-List:	ARMEMPTY ^ ON(X,Y)

Actions: (1) **UNSTACK(B,A)**

ARMEMPTY
PICKUP(C)
STACK(C,A)
ON(B,D)

STACK(B,D)
PICKUP(C)
STACK(C,A)
ON(B,D)

CLEAR(D)
HOLDING(B)
STACK(B,D)
PICKUP(C)
STACK(C,A)
ON(B,D)

Block World Example

HOLDING(B) ^ CLEAR(A) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(D)

STACK(X,Y):

Precondition:	CLEAR(Y) ^ HOLDING(X)
Delete-List:	CLEAR(Y) ^ HOLDING(X)
Add-List:	ARMEMPTY ^ ON(X,Y)

S1

Actions:
(1) **UNSTACK(B,A),**
(2) **STACK(B,D)**

S2

ARMEMPTY ^ ON(B,D) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(A)

~~CLEAR(D)~~
~~HOLDING(B)~~
STACK(B,D)
PICKUP(C)
STACK(C,A)
ON(B,D)

PICKUP(C)
STACK(C,A)
ON(B,D)

Block World Example

HOLDING(B) ^ CLEAR(A) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(D)

S1

PICKUP(X):

Precondition: CLEAR(X) ^ ONTABLE(X) ^ ARM
Delete-List: ONTABLE(X) ^ ARMEMPTY
Add-List: HOLDING(X)

Actions:

(1) **UNSTACK(B,A),**

(2) **STACK(B,D)**

(3) **PICKUP(C)**

S2

ARMEMPTY ^ ON(B,D) ^ ONATBLE(A) ^ ONATBLE(C) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(A)

~~ARMEMPTY~~
~~PICKUP(C)~~
STACK(C,A)
ON(B,D)

STACK(C,A)
ON(B,D)

HOLDING(C) ^ ON(B,D) ^ ONATBLE(A) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(A)

S3

Block World Example

HOLDING(C) ^ ON(B,D) ^ ONATBLE(A) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C) ^ CLEAR(A)

S3

STACK(X,Y):

Precondition:	CLEAR(Y) ^ HOLDING(X)
Delete-List:	CLEAR(Y) ^ HOLDING(X)
Add-List:	ARMEMPTY ^ ON(X,Y)

S4

ARMEMPTY ^ ON(C,A) ^ ON(B,D) ^ ONATBLE(A) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C)

Actions:

(1) **UNSTACK(B,A),**

(2) **STACK(B,D)**

(3) **PICKUP(C)**

(4) STACK(C,A)

~~HOLDING(C)~~

~~STACK(C,A)~~

ON(B,D)

ON(B,D)

Block World Example

ARMEMPTY ^ ON(C,A) ^ ON(B,D) ^ ONATBLE(A) ^ ONATBLE(D) ^ CLEAR(B) ^ CLEAR(C)

S4

STACK(X,Y):
Precondition: CLEAR(Y) ^ HOLDIN
Delete-List: CLEAR(Y) ^ HOLDIN
Add-List: ARMEMPTY ^ ON(X,

Actions:
(1) **UNSTACK(B,A),**
(2) **STACK(B,D)**
(3) **PICKUP(C)**
(4) **STACK(C,A)**

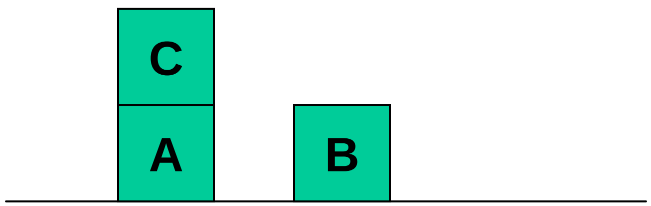
~~ON(B,D)~~

Output:

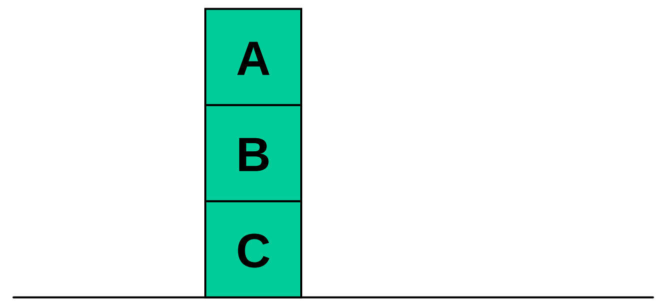
Actions:
(1) **UNSTACK(B,A),**
(2) **STACK(B,D)**
(3) **PICKUP(C)**
(4) **STACK(C,A)**

Block World Example

❖ Exercise:



Start



Goal