CS 4649/7649 RIP
Robot Intelligence: Planning

STRIPS & Linear Planning Analysis

Sungmoon Joo

School of Interactive Computing
College of Computing
Georgia Institute of Technology

Course Info.

- **Course Website: joosm.github.io/RIP2014**
  - **Lecture slides**
  - **Project examples**
- **Email me(sungmoon.joo@cc.gatech.edu)**
  - **Introduce yourself**
  - **Project ideas**
- **Git**

What is Planning?

Planning:
"devising a plan of action to achieve one's goal" (Russel & Norvig)

Given:
- **States**
- **Actions**
- **Initial State and Goal State**
- **Constraints**

Task 1: Find a sequence of actions that take you from Init. to Goal

Task 2: Find actions that take you from any state to Goal

Task 3: Decide the best action to take now in order to improve your odds of reaching Goal

Task 4: Find a continuous path (in state space) that takes you from Init. to Goal

…

First Representation: Predicate Logic

Unless, some how, we can describe the world, we cannot devise a plan.

Statement → Predicate

(Unary) Predicate: P(x)
Introduce a (functional) symbol(P) for the predicate, and put the subject(x) as an argument to the functional symbol.

N-ary predicate is defined similarly

Examples:
- **x is happy → Happy(x): Unary predicate**
- **The suitcase contains a bomb → Contains(Suitcase, Bomb): Binary predicate**
- **X is less than Y → LessThan(x,y):Binary predicate**
- **$P(x_1,x_2,\ldots,x_n)$: n-ary predicate**

***Proposition can be considered as a 0-nary predicate.**
***Predicate Logic ~ First Order Logic(FOL)**

First Representation: Predicate Logic

Objects (Constants): $a, 123, house, mike, robot, suitcase$

Variables: $x, y, z...$

Relations (Predicates): $LessThan, Contains, Parent, Happy$

Connectives: $\neg, \vee, \wedge, \Rightarrow$

Any expression is either true or false: $LessThan(1,2)$

$\neg Contains(Suitcase, Bomb)$

Truth Table

A	B	$\neg A$	$A \wedge B$	$A \Rightarrow B$	$\neg A \vee B$
F	F	T	F	T	T
F	T	T	F	T	T
T	F	F	F	F	F
T	T	F	T	T	T

Situation Calculus*

To represent and reason about dynamical worlds

To represent 'change', 'state (implicitly time)' is introduced.

- **Fluents** = **Aspects of the world that change**

$$Contains(Suitcase, Laptop, \mathbf{S0})$$
$$Working(Robot, \mathbf{S7})$$

- **Actions** are *reified* functions of constants. (They can be treated as constants themselves)

$$Put(Laptop, Suitcase) \qquad Open(Car) \qquad Lock(Car)$$

- The **do function**: $do(\alpha, \sigma_0) \rightarrow \sigma_1$

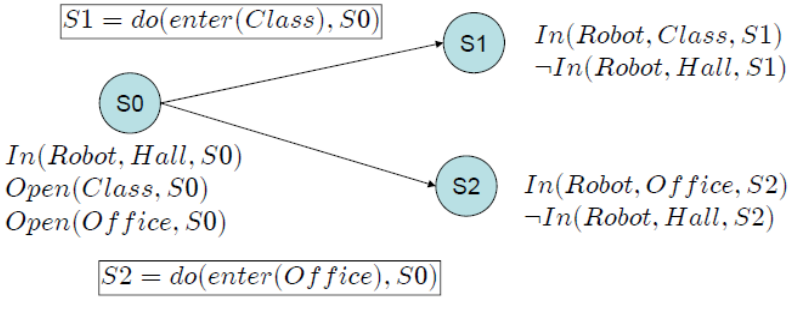
$$\alpha = action$$
$$\sigma = state$$

***Modern version is different from the original for clarity. Calculus = study about 'change'.**

Situation Calculus

- Effect axioms describe how a world changes by an action
- Effect Axioms (Positive + Negative) : Specify the outcome of an action

$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow In(Robot, rm, \mathbf{do(enter(rm), s)})$$
$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow \neg In(Robot, Hall, \mathbf{do(enter(rm), s)})$$

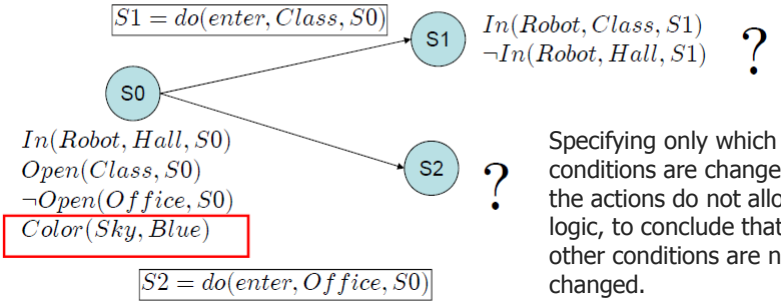


Situation Calculus: Frame Problem*

$Room(rm)$

- Effect Axioms (Positive + Negative)

$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow In(Robot, rm, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$
$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow \neg In(Robot, Hall, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$



Specifying only which conditions are changed by the actions do not allow, in logic, to conclude that all other conditions are not changed.

***First recognized by McCarthy & Hayes (1969)**

Frame Axioms

Effect Axioms (Positive + Negative)

$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow In(Robot, rm, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$
$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow \neg In(Robot, Hall, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$

For each unchanged fluent we add:

$$Open(Office, s) \Rightarrow Open(Office, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$
$$\neg Open(Office, s) \Rightarrow \neg Open(Office, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$

$$Color(Sky, Blue, s) \Rightarrow Color(Sky, Blue, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$
$$\neg Color(Sky, Blue, s) \Rightarrow \neg Color(Sky, Blue, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$

How many in total? (for n distinct fluents and m distinct actions)

$2nm$ (Not exponential – but often not practical)

Explicitly specify that all conditions not affected by actions are not changed while executing that action

Frame Problem

- (Representational) Frame Problem
  Significant because the real world has very many fluents
  Size of axioms: O(mn)
- Inferential Frame Problem
  Problem of projecting forward the results of a t step of actions in time O(nt)
- Ramification Problem
  Deals with the indirect consequences of an action
  The frame problem in the context of actions with indirect effects

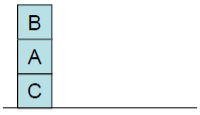
STRIPS (Fikes & Nilson 1971)

- Restrict state descriptions to conjunctions of fluents (literals)
  - Why?
- Represent actions (operators) as three parts:
  - PC: set of preconditions
  - D: Delete List
  - A: Add List
- Use recursive search to implement planning

STRIPS: Blocks World

Constants:

$A, B, C, Table$
$IsBlock(A), IsBlock(B)...$



Ground Literals:

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

Actions:

$move(x, y, z)$

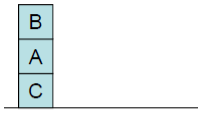
Domain Axioms:

$On(y, x) \wedge On(z, x) \wedge (x \neq Table) \Rightarrow (y = z)$

STRIPS: Blocks World

Constants:

$A, B, C, Table$
$IsBlock(A), IsBlock(B)...$



Ground Literals:

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

Actions: $move(x, y, z)$

PC: $On(x, y)$ D: $On(x, y)$ A: $On(x, z)$
$Clear(x)$ $Clear(z)$ $Clear(y)$
$Clear(z)$ $Clear(Table)$

Domain Axioms:

$On(y, x) \wedge On(z, x) \wedge (x \neq Table) \Rightarrow (y = z)$

STRIPS: Blocks World

Constants:

$A, B, C, Table$
$IsBlock(A), IsBlock(B)...$

Ground Literals:

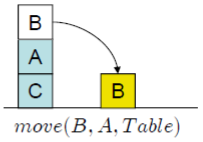
$On(B, A)$ $On(C, Table)$
$Clear(B)$ $Clear(Table)$

Actions: $move(x, y, z)$

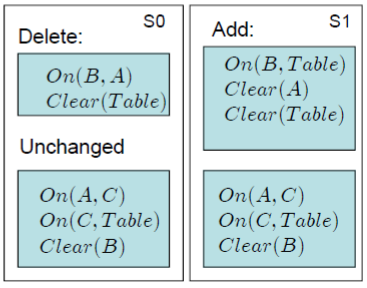
PC: $On(x, y)$ $Clear(x)$ $Clear(z)$
D: $On(x, y)$ $Clear(z)$
A: $On(x, z)$ $Clear(y)$ $Clear(Table)$

Domain Axioms:

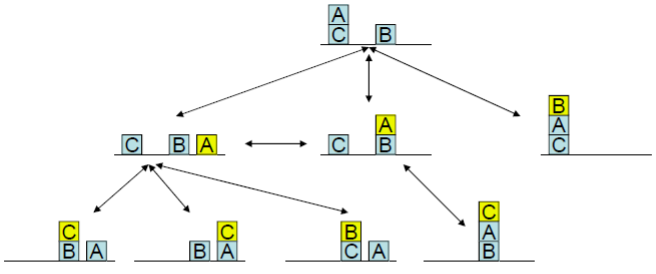
$On(y, x) \wedge On(z, x) \wedge (x \neq Table) \Rightarrow (y = z)$



$move(B, A, Table)$



Planning as Search



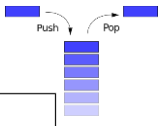
▪ ▪ ▪

Linear Planning: How STRIPS Works

- Goal is to speed up search
- Non-Linear Planning:
  - Interleave search for subgoals
  - Keeps a **set** of unachieved goals
- Linear Planning:
  - Solve one goal at a time
  - Search with a **stack** of unachieved goals

Based on General Problem Solver (GPS)

- Newell, Simon & Ernst 1960's
- Introduced means-ends analysis
- Using a recursive goal stack to speed up planning



Fikes & Nilson '71

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

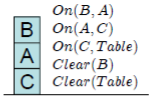
Means – Ends Analysis

- Means-Ends Analysis
  - Search only relevant aspect of problem
  - What means (operators) are available to achieve the desired ends(goal)
  - Find difference between goal and current state
  - Find operator to reduce the difference
  - Perform means-ends analysis on new subgoals

STRIPS Solves Blocks World

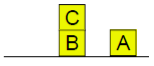
	Stack:	State:
1		$On(B,A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$
	$On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$Clear(Table)$

init



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

goal



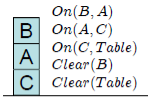
$On(A,Table) \wedge On(B,Table) \wedge On(C,B)$

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

STRIPS Solves Blocks World

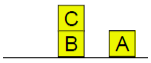
	Stack:	State:
4	$Clear(A)$ $Clear(Table)$ $move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$
5	$On(B,A)$ $Clear(B)$ $Clear(Table)$ $move(B,A,Table)$ $Clear(Table)$ $move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$
6	$move(B,A,Table)$ $Clear(Table)$ $move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$

init



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

goal



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

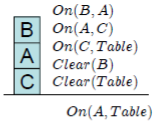
	PC	D	A
$move(x,y,z)$	$On(x,y)$ $Clear(x)$ $Clear(z)$	$On(x,y)$ $Clear(z)$	$On(x,z)$ $Clear(y)$ $Clear(Table)$

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then Pop
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

STRIPS Solves Blocks World

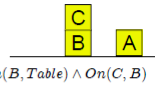
	Stack:	State:
4	$Clear(A)$ $Clear(Table)$ $move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$
5	$On(B,A)$ $Clear(B)$ $Clear(Table)$ $move(B,A,Table)$ $Clear(Table)$ $move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$
6	$move(B,A,Table)$ $Clear(Table)$ $move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$

init



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

goal



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

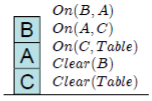
	PC	D	A
$move(x,y,z)$	$On(x,y)$ $Clear(x)$ $Clear(z)$	$On(x,y)$ $Clear(z)$	$On(x,z)$ $Clear(y)$ $Clear(Table)$

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then Pop
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

STRIPS Solves Blocks World

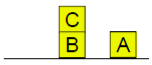
	Stack:	State:
7	$Clear(Table)$ $move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,Table)$ $Clear(A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$
8	$move(A,C,Table)$ $On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(B,Table)$ $Clear(A)$ $On(A,C)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$
9	$On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(A,Table)$ $Clear(C)$ $On(B,Table)$ $Clear(A)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$

init



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

goal



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

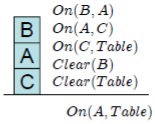
	PC	D	A
$move(x,y,z)$	$On(x,y)$ $Clear(x)$ $Clear(z)$	$On(x,y)$ $Clear(z)$	$On(x,z)$ $Clear(y)$ $Clear(Table)$

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then Pop
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

STRIPS Solves Blocks World

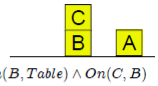
	Stack:	State:
9	$On(B,Table)$ $On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(A,Table)$ $Clear(C)$ $On(B,Table)$ $Clear(A)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$
10	$On(C,B)$ $On(A,Table) \wedge On(B,Table) \wedge On(C,B)$	$On(A,Table)$ $Clear(C)$ $On(B,Table)$ $Clear(A)$ $On(C,Table)$ $Clear(B)$ $Clear(Table)$

init



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

goal

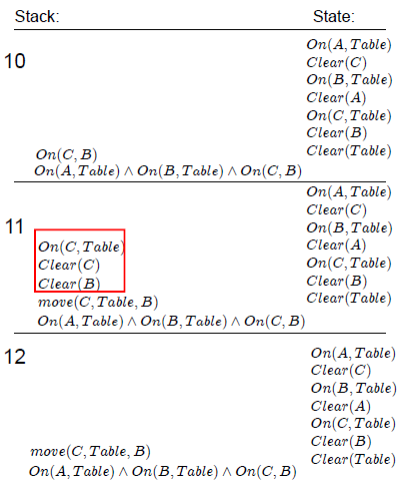


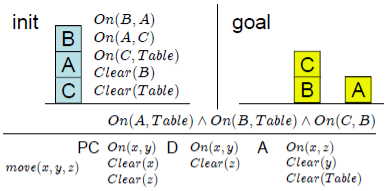
$On(A,Table) \wedge On(B,Table) \wedge On(C,B)$

	PC	D	A
$move(x,y,z)$	$On(x,y)$ $Clear(x)$ $Clear(z)$	$On(x,y)$ $Clear(z)$	$On(x,z)$ $Clear(y)$ $Clear(Table)$

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then Pop
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

STRIPS Solves Blocks World

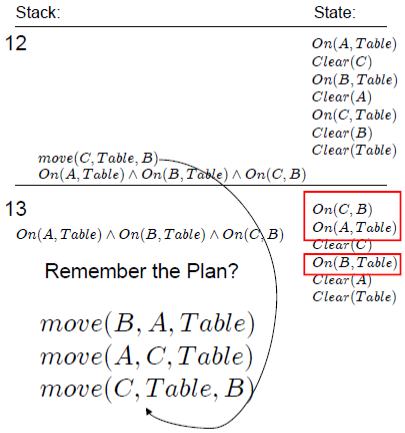


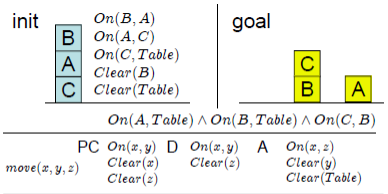


	PC	D	A
$move(x, y, z)$	$On(x, y)$ $Clear(x)$ $Clear(z)$	$On(x, y)$ $Clear(z)$	$On(x, z)$ $Clear(y)$ $Clear(Table)$

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then Pop
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

STRIPS Solves Blocks World

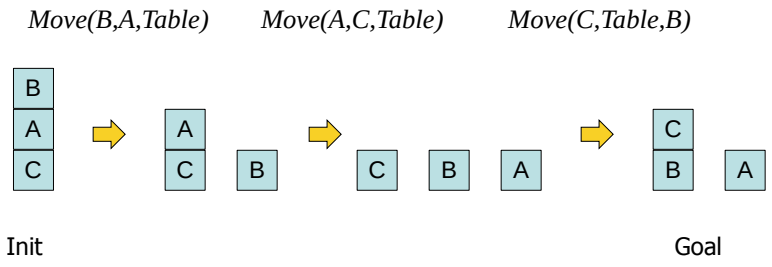




	PC	D	A
$move(x, y, z)$	$On(x, y)$ $Clear(x)$ $Clear(z)$	$On(x, y)$ $Clear(z)$	$On(x, z)$ $Clear(y)$ $Clear(Table)$

STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then Pop
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

STRIPS Solves Blocks World



Did STRIPS Solve Our Problems?

Frame Problem?

- Yes

Ramification Problem?

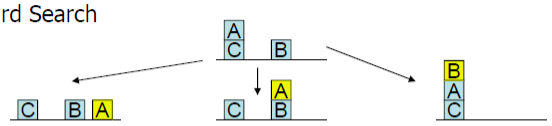
- Not yet
- Separate ground facts from inferred facts
- Only ground facts are carried over from state to state
- Potentially inefficient

Planning Method?

- Yes, linear search - but how do we evaluate it?

Back to Basics

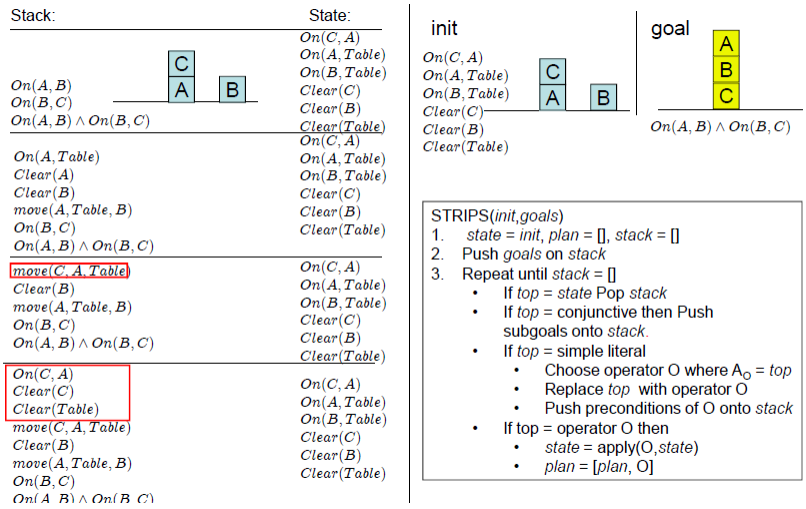
- Forward Search



- STRIPS: Means-End Analysis
  - Start at goal
  - Consider effects of actions to guide search

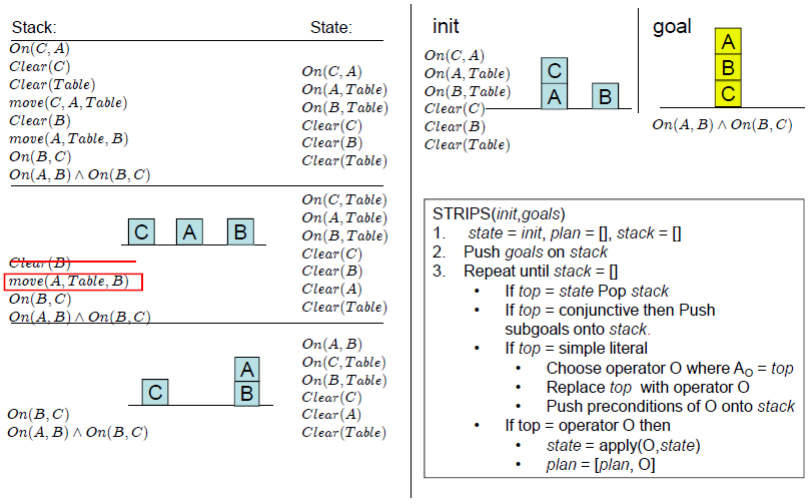
How would STRIPS solve this?

Blocks World II



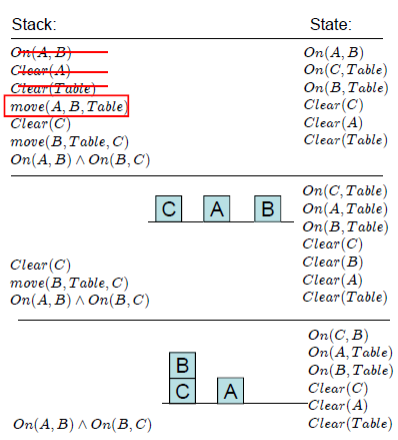
STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

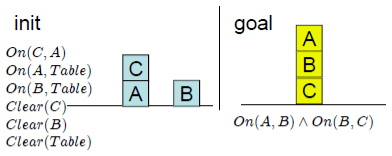
Blocks World II



STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

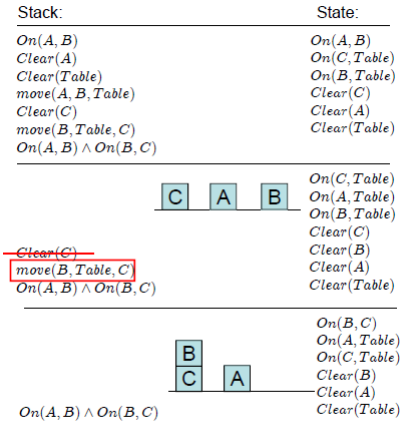
Blocks World II

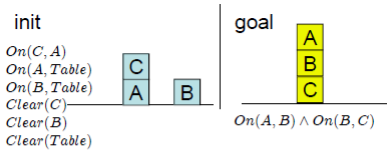




STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

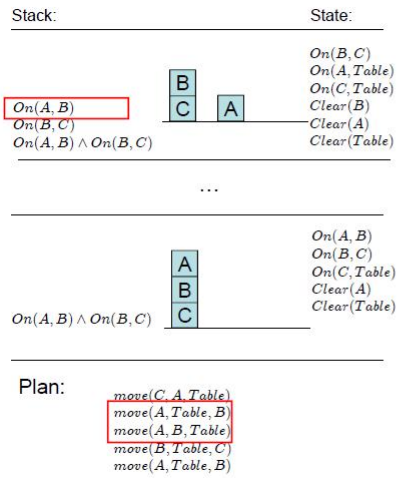
Blocks World II

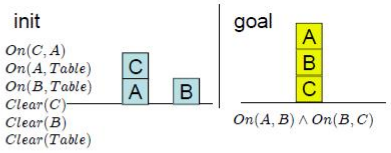




STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

Sussman Anomaly

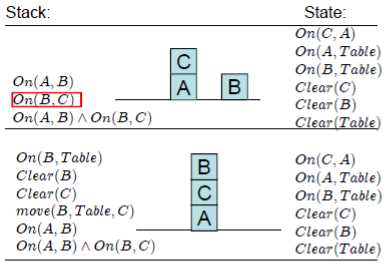


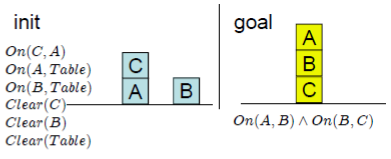


Gerald Sussman

(MIT since 1964)

Sussman Anomaly (Order Change)





Ordering doesn't matter
Still the same problem!

*Called 'anomaly' because it seemed to make sense for a conjunctive goals to first achieve one goal and then achieve another goal, and then the complete goal would be achieved.

Properties of a Planner

1) Sound: The planner produces valid plans
   - STRIPS is sound
2) Optimal: The planner produces optimal (shortest) plans
   - STRIPS is suboptimal
3) Complete: The planner finds a solution when there is one or returns that the solution is not possible.
   - STRIPS is incomplete!