Fundamentals of Artificial Intelligence

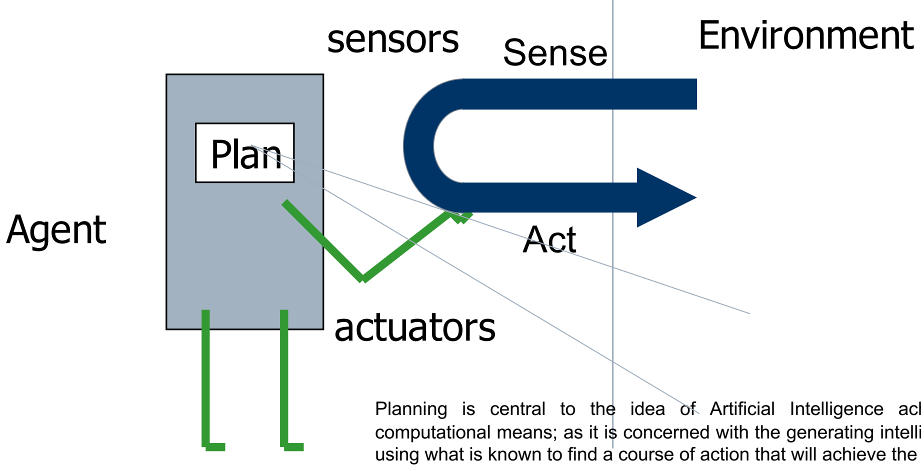
Introduction to Planning

Shyamanta M Hazarika
Mechanical Engineering
Indian Institute of Technology Guwahati
s.m.hazarika@iitg.ac.in

http://www.iitg.ac.in/s.m.hazarika/

Planning Agent

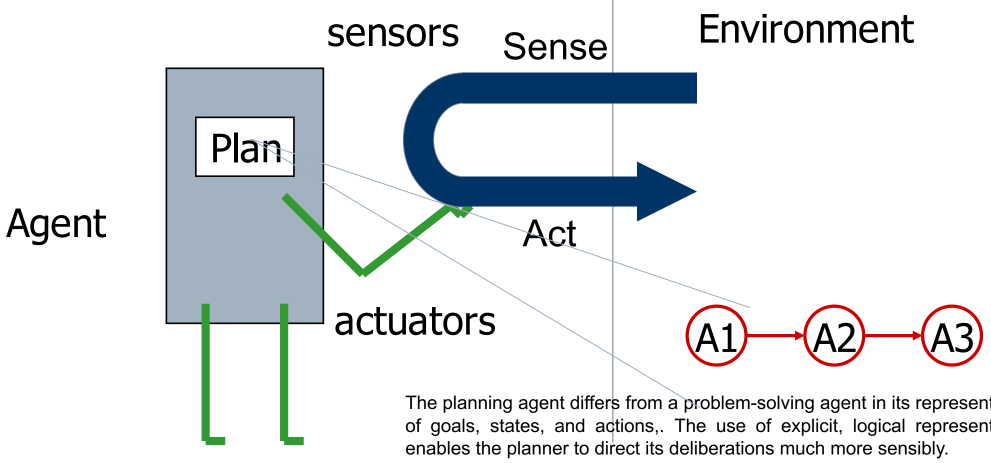
Planning is one of the most useful ways that an intelligent agent can take advantage of the knowledge it has and its ability to reason about action and consequences.



Planning is central to the idea of Artificial Intelligence achieved through computational means; as it is concerned with the generating intelligent behaviour, using what is known to find a course of action that will achieve the goal.

Planning Agent

A planning agent is very similar to a problem-solving agent, in that it constructs plans that achieve its goals, and then executes them.



The planning agent differs from a problem-solving agent in its representations of goals, states, and actions,. The use of explicit, logical representations enables the planner to direct its deliberations much more sensibly.

What is Planning?

- Planning is **reasoning about future events** in order to **establish a series of actions to accomplish a goal**.

  A common approach to planning is representing a current state and **determining the series of actions necessary to reach the goal state**. (or vice versa)
  - Problem solving technique
  - Plans are created by searching through a space of possible actions until the sequence necessary to accomplish the task is discovered.
    - Planning is a specific kind of state space search
    - Deals with steps and goals that interact

Search in Planning

- Planning **involves search** through a space
  - Progression: **choose an action whose preconditions are met** until a goal state is reached
    - A forward approach, simple algorithm, but can have large branching factor.
  - Regression: **choose an action that matches an unachieved subgoal** while adding unmet preconditions to the set of subgoals. Continue until the set of subgoals is empty.
    - A backward approach, goal oriented, tends to be more efficient.

Early Planning Systems

1956 – Logic Theorists – Newell and Simon

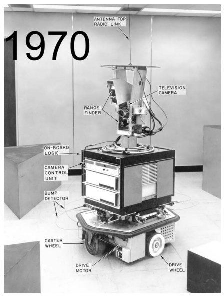
- One of the first to use heuristics
- Proved theorems in propositional calculus,
- Operated by using backward reasoning from the theorem to the axioms
- Limited by its heuristics and
- Certain theorems could not be proven

1957-1969 – GPS – Newell and Simon

- How to solve human intelligence problems?
- Areas – propositional calculus proofs, puzzles, symbolic integrations, etc.
- Introduced means-end analysis
  - Find the difference between the current state and goal; used a table to find an action to minimize the difference between the two states.

Non-Hierarchical Planners

- **Earliest Method of Planning**
  - Made **no distinction between more and less important plan elements**
  - Slowed by getting **hung up on less important elements**
  - **Lack of structure** led to poor performance with complex problems
  - Example: **STRIPS**
    - **ST**anford **R**esearch **I**nstitute **P**lanning **S**ystem
      - Fikes and Nilsson, 1971
    - Used to run the SHAKEY robot of the 1970's



Hierarchical Planners

- Makes a **distinction between more and less important parts** of the plan
  - Example: When purchasing a new Car, we first need to decide where to get the funds. It doesn't make sense to find a good parking place on campus before you have the money!
  - Example: **ABSTRIPS**
    - Abstract-Based STRIPS
    - Like STRIPS; but plans in a hierarchy, greatly reduces search space, and is more efficient at solving large problems
    - Certain preconditions are judged as more important than others by adding weights to those elements
    - Finds early recognition of bad paths and gets rid of wasted search
    - Uses a hierarchy of abstraction levels; Solves highest level of abstraction. If that passes, it increases level of detail.

Classical Planning Assumptions

- Perfect Information
- Deterministic Effects
- Instantaneous Execution
- Solo Agent
- No concern over time, cost, resources
- Finite and Static

These **assumptions were made early on because complex tasks were too complex to solve**. These assumptions were used to complete smaller tasks (such as the blocks world examples).

Modern approaches deal with the scaling issue.

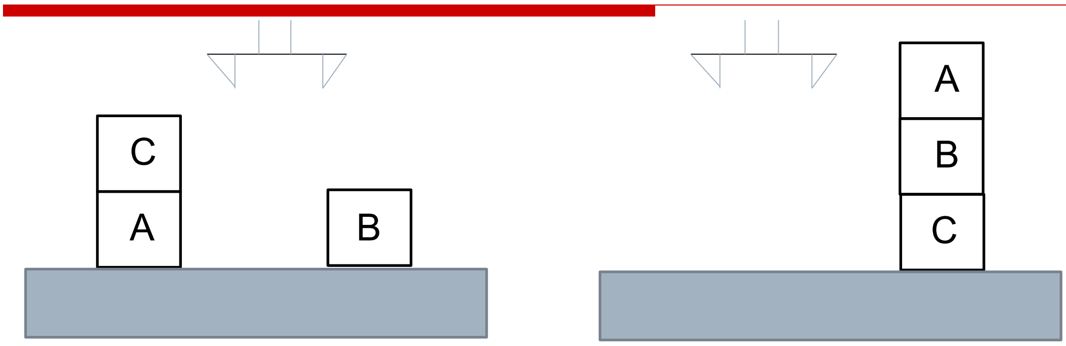
The Planning Problem

- Find a **sequence of actions** that achieves a given **goal** when executed from a given **initial world state**.

  Given
  - an initial state description,
  - a set of action descriptions - defining the possible primitive actions by the agent, and
  - a goal state description or predicate,

  compute a plan, which is
  - a **sequence of action instances** - such that executing them in the initial state will change the world to a **state satisfying the goal-state** description.
- Goals are usually specified as a conjunction of subgoals to be achieved.

Initial State and Goal State



Initial State Final State

The initial state and goal state for a simple planning problem from the block world is shown. The **purpose of planning is to find a sequence of actions that gets from the initial state to the goal state**, or from the goal state back to the initial state.

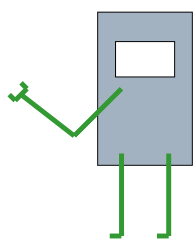
11

The Planning Problem

Planning and problem solving are different because of the differences in the representations of goals, states, and actions, and the differences in the representation and construction of action sequences.

Consider the robot needs to solve the following

Get a quart of milk; a dark chocolate and a good book.



12

Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The successor function is a black box!
    - It must be "applied" to a state to know which actions are possible in that state and what are the effects of each one.

Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The successor function is a black box!
    - Suppose the goal is HAVE(MILK).
    - From some initial state where HAVE(MILK) is not satisfied, the successor function must be repeatedly applied to eventually generate a state where HAVE(MILK) is satisfied.
    - An **explicit representation of the possible actions and their effects** would help the problem solver select the relevant actions.

      Otherwise, in the real world an agent would be overwhelmed by irrelevant actions.

Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The goal test is another black-box function.
    - States are domain-specific data structures, and heuristics must be supplied for each new problem.

15

Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The goal test is another black-box function.
  - Suppose that the goal is HAVE(MILK)∧HAVE(BOOK)
  - Without an **explicit representation of the goal**, the problem solver cannot know that a state where HAVE(MILK) is already achieved is more promising than a state where neither HAVE(MILK) nor HAVE(BOOK) is achieved .

16

Goal of Planning

- Choose actions to achieve a certain goal
- But isn't it exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The goal may consist of several nearly independent sub-goals, but there is no way for the problem solver to know it!
  - HAVE(MILK) and HAVE(BOOK) may be **achieved by two nearly independent sequences** of actions

Representation in Planning

- Planning **opens up the black-boxes** by using logic to represent:
  - Actions
  - States
  - Goals

The first key idea behind planning is to "open up " the representation of states, goals, and actions. Planning algorithms use descriptions in some formal language, usually first-order logic or a subset thereof.

Problem solving → Planning ← Logic representation

The second key idea is to add actions to the plan wherever they are needed, rather than in an incremental sequence starting at the initial state. E.g., BUY(Milk).

Making "obvious" decisions first, can reduce the branching factor and reduce the need to backtrack.

The final key idea behind planning is that most parts of the world are independent of most other parts.

A conjunctive goal like "get a quart of milk and a chocolate and a book" and solve it with a divide-and-conquer strategy.

Planning vs. Problem Solving

- Planning and problem solving methods can often solve the same sorts of problems.
- Planning is more powerful because of the representations and methods used.
  - States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
  - Search often proceeds through plan space rather than state space
    - Though first we will talk about state-space planners.
  - Subgoals can be planned independently
    - Reducing the complexity of the planning problem

Situation Calculus

- Situation calculus is a dialect of First Order Logic in which beliefs about a changing world can be represented.

This is not the only way to represent a changing world! It is simple and powerful. Lends it self naturally to various sorts of reasoning, including planning.

- Situations and actions are explicitly taken to be objects in the domain.

You may be in the situation of `holding a crystal vase'; the action of dropping and breaking it moves you to the `next' situation of having a broken crystal vase.

Ontology

situation variable: a "timestamp" added to fluents;

fluents are predicates or functions whose values may vary from situation to situation.

situation: a "snapshot" of world when nothing changes!

action: represented as logical terms E.g., Forward, Turn(right); constant and function symbols for actions are application domain dependent.

Situation Calculus

- Situation calculus is a dialect of First Order Logic in which beliefs about a changing world can be represented.

This is not the only way to represent a changing world! It is simple and powerful. Lends it self naturally to various sorts of reasoning, including planning.

- Situations and actions are explicitly taken to be objects in the domain.

You may be in the situation of `holding a crystal vase'; the action of dropping and breaking it moves you to the `next' situation of having a broken crystal vase.

Ontology

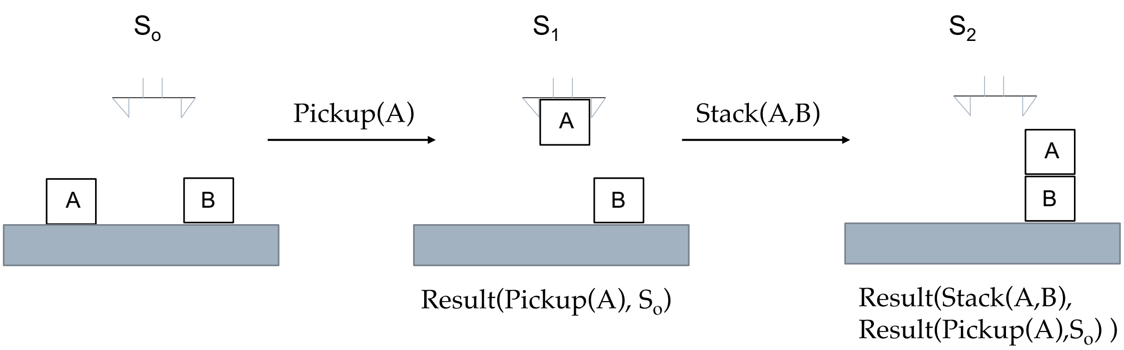
Result(a, s): function that **returns next situation** after applying action a in situation s

fluents – functions and predicates that vary from one situation to the next.

Atemporal or **external predicates and functions** are also allowed – they don't have a situation as an argument. E.g., Gold(g1);

Situation Calculus: Example

Given: block(A), block(B), onTable(A), onTable(B)



Result: block(A), block(B), onTable(B), on(A,B)

Situation Calculus

- Sequences of Actions in Situation Calculus

  Result([], S) = S
  Result([a|seq],S)=Result(seq,Result(a,S))

- Describe a world as it stands, define a number of actions, and then attempt to prove there is a sequence of actions that results in some goal being achieved.
- What has to go in our knowledge base to prove these things?
  - Need to have a description of actions.

Representing Actions

- Possibility Axioms: specifies under what conditions it's possible to execute action a in state s

  $\text{Preconditions} \Rightarrow \text{Poss}(a,s)$
  - E.g. $\text{At}(\text{Agent},x,s) \wedge \text{Adjacent}(x,y) \Rightarrow \text{Poss}(\text{Go}(x,y),s)$

- Effect Axioms: specifies changes that result when action a is executed in state s

  $\text{Poss}(a,s) \Rightarrow \text{Changes resulting from taking action}$
  - E.g. Poss(Go(x,y),s) ⇒ At(Agent,y,Result(Go(x,y),s))

The Frame Problem

Effect axioms are too simple. They say what changes, but not what stays the same.

- The Frame Problem:
  - Representing all that stays the same must be done in addition to describing what changes when an action is applied.
  - Since almost everything stays the same from one situation to the next, must find an efficient solution for stating what doesn't change.

Solving the Frame Problem

- Consider how each fluent evolves over time instead of writing the effects of each action.
- Axioms are called **successor-state axioms**
  - Specifies truth value of fluent in the next state as a function of the action and truth value in the current state
- $Poss(a,s) \land \gamma^{+}_{F}(x,a,s) \rightarrow F(x,do(a,s))$
- $Poss(a,s) \land \gamma^{-}_{F}(x,a,s) \rightarrow \neg F(x,do(a,s))$
  - γ^{+}_{F} describes the conditions under which action a in situation s makes the fluent F become true in the successor situation do(a,s)
  - γ^{-}_{F} describes the conditions under which performing action a in situation s makes fluent F false in the successor situation.

Successor State Axiom

- $Poss(a,s) \rightarrow [F(x,do(a,s)) \leftrightarrow \gamma^{+}_{F}(x,a,s) \vee (F(x,s) \wedge \neg\gamma^{-}_{F}(x,a,s))]$

  Given that it is possible to perform a in s, the fluent F would be true in the resulting situation do(a,s) iff performing a in s would make it true, or it is true in s and performing a in s would not make it false.

  - Example:

    $Poss(a,s) \rightarrow [broken(o,do(a,s)) \leftrightarrow (a=drop(o) \wedge fragile(o)) \vee (broken(o,s) \wedge a \neq repair(o,s))]$

Planning in the Situation Calculus

- Situation calculus enables a KB agent to reason about the results of actions
  - projecting future results
  - finding a plan
- Proving a plan achieves a goal and requires
  - a goal to prove
  - an initial state description that says what is and isn't true
  - successor-state axioms, which take into consideration implicit effects.
  - an efficient inference procedure using this kind of axioms.

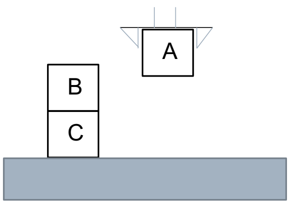
The STRIPS Representation

- STRIPS is an alternative representation to the pure situation calculus for planning.
- In STRIPS, the world we are trying to deal with satisfies the following criteria
  - Only one action can occur at a time.
  - Actions are effectively instantaneous.
  - Nothing changes except as result of planned actions.
- Actions are not represented explicitly as part of the world model; we do not reason about them directly.
  - Actions are thought of as operators that syntactically transform world models.
    - Main advantage of this way of representation is that it avoids the frame problem; changes what needed and leave the rest unaffected.

Representing States

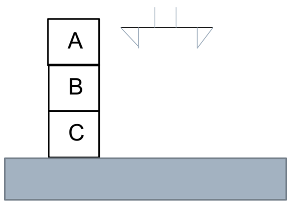
World states are represented as sets of facts. We will also refer to facts as propositions.

State 1



holding(A)
clear(B)
on(B,C)
onTable(C)

State 2



handEmpty
clear(A)
on(A,B)
on(B,C)
onTable(C)

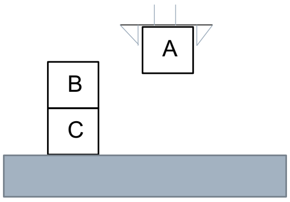
Closed World Assumption (CWA)

Fact not listed in a state are assumed to be false. Under CWA we are assuming the agent has full observability.

Representing Goals

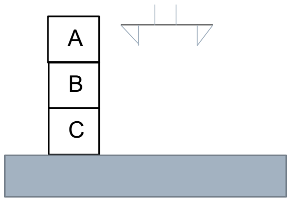
Goals are represented as sets of facts.
For example {on(A,B)} is a goal in the blocks world.

State 1



holding(A)
clear(B)
on(B,C)
onTable(C)

State 2



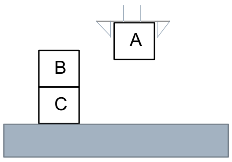
handEmpty
clear(A)
on(A,B)
on(B,C)
onTable(C)

A goal state is any state that contains all the goal facts.

State 1 is NOT a goal state for the goal { on(A,B) }.
State 2 is a goal state for the goal { on(A,B) }.

Representing Actions

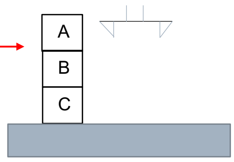
State 1



holding(A)
clear(B)
on(B,C)
onTable(C)

PutDown(A,B) →

State 2



handEmpty
clear(A)
on(A,B)
on(B,C)
onTable(C)

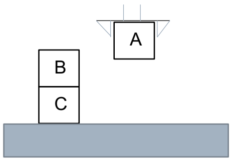
A STRIPS action definition specifies:

1. a set PRE of preconditions facts
2. a set ADD of add effect facts
3. a set DEL of delete effect facts

Stack(A,B)
PRE: { holding(A), clear(B) }
ADD: { on(A,B), handEmpty, clear(A) }
DEL: { holding(A), clear(B) }

Semantics of STRIPS Actions

State 1

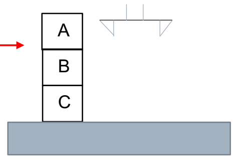


holding(A)
clear(B)
on(B,C)
onTable(C)

S

PutDown(A,B)

State 2



handEmpty
clear(A)
on(A,B)
on(B,C)
onTable(C)

S ∪ ADD – DEL

Stack(A,B)
PRE: { holding(A), clear(B) }
ADD: { on(A,B), handEmpty, clear(A) }
DEL: { holding(A), clear(B) }

A STRIPS action is **applicable** (or allowed) in a state when its preconditions PRE are contained in the state. Taking an action in a state S results in a new state S ∪ ADD – DEL i.e. add the ADD effects and remove the DEL effects.

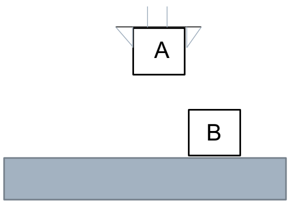
33

STRIPS Planning Problems

A **STRIPS planning problem** specifies:
a. an initial state S
b. a goal G
c. a set of STRIPS actions

Objective: Find a short action sequence reaching a goal state, or report that the goal is unachievable.

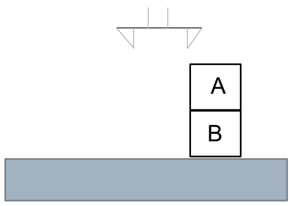
Example **Initial State**



holding(A)
clear(B)
onTable(B)

on(A,B)
handEmpty
onTable(B)

Goal



Plan: Stack(A,B)

34

STRIPS Action Schemas

For convenience we typically specify problems via action schemas rather than writing out individual STRIPS actions.

Given a set of schemas, an initial state, and a goal, propositional planners compile schemas into ground actions and then ignore the existence of objects thereafter.

Stack(x,y):
PRE: { holding(x), clear(y) }
ADD: { on(x,y), handEmpty, clear(x)}
DEL: { holding(x), clear(y) }

Stack(A,B):
PRE: { holding(A), clear(B) }
ADD: { on(A,B), handEmpty, clear(A)}
DEL: { holding(A), clear(B) }

Stack(B,A):
PRE: { holding(B), clear(A) }
ADD: { on(B,A), handEmpty, clear(B)}
DEL: { holding(B), clear(A) }

Each way of replacing variables with objects from the initial state and goal yields a "ground" STRIPS action.

The Basic Idea

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

1. The system has a database and a goal stack.
2. It focuses attention on solving the top goal.

   This may involve solving sub-goals, which are then pushed onto the stack.

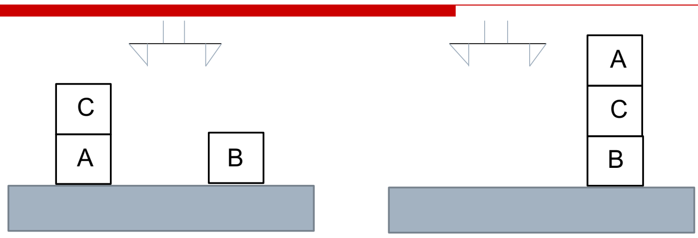
1. Place goal Goal-1 in goal stack.
2. Considering top Goal-1, place onto it its sub-goals.
3. Then try to solve sub-goal GoalS1-2, and continue.

GoalS1-2
GoalS1-1
Goal-1

Stack Manipulation Rules

	Top of the Stack	Action
1.	Compound or single goal matching the current state description.	Remove it
2.	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
3.	Single-literal goal not matching the current state description.	Find new rule whose instantiated add-list includes the goal, and 1. Replace the goal with the instantiated rule; 2. Place the rule's instantiated precondition formula on top of stack
4.	Rule	1. Remove rule from stack; 2. Update database using rule; 3. Keep track of rule (for solution**)**
5.	Nothing	Stop.

STRIPS Planning Example



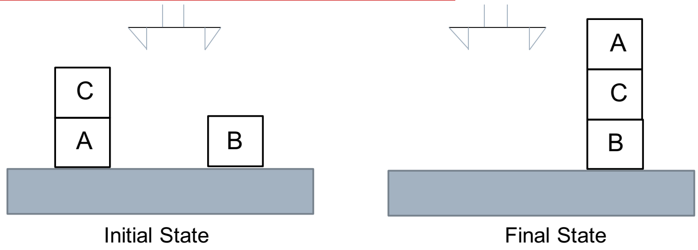
Initial State   Final State

1. Place the original goal on stack.

Stack: on(A,C) and on(C,B)

Database:
clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example



2. Since top goal is unsatisfied compound goal, list its unsatisfied sub-goals on top of it:

Stack:

on(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

39

STRIPS Planning Example

3. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
   a. Replace the goal with the instantiated rule;
   b. Place the rule's instantiated precondition formula on top of stack

Stack(C,B):
PRE: { holding(C), clear(B) }
ADD: { on(C,B), handEmpty, clear(C)}
DEL: { holding(C), clear(B) }

Stack:

holding(C) and clear(B)
Stack(C,B) ↔ on(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

4. Since top goal is unsatisfied compound goal, list its sub-goals on top of it:

Stack:

clear(B)
holding(C)
holding(C) and clear(B)
Stack(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

5. Single goal on top of stack matches data base, so remove it:

Stack:

~~clear(B)~~
holding(C)
holding(C) and clear(B)
Stack(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

6. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
   a. Replace the goal with the instantiated rule;
   b. Place the rule's instantiated precondition formula on top of stack

UnStack(C,A):
PRE: { on(C,A), handEmpty, clear(C) }
ADD: { holding(C), clear(B)}
DEL: {on(C,A), handEmpty, clear(C)}

Stack:

on(C,A) and clear(C) and handempty
UnStack(C,A) ⟷ holding(C)
holding(C) and clear(B)
Stack(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

7. Compound goal on top of stack matches data base, so remove it:

Stack:

~~on(C,A) and clear(C) and handempty~~
UnStack(C,B)
holding(C) and clear(B)
Stack(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

8. Top item is rule, so:
    a. Remove rule from stack;
    b. Update Database using rule;
    c. Keep track of rule (for solution)

UnStack(C,A):
ADD: { holding(C), clear(A)}
DEL: {on(C,A), handEmpty, clear(C)}

Solution: {Unstack(C,A)}

~~UnStack(C,B)~~
holding(C) and clear(B)
Stack(C,B)
on(A,C)
Stack: on(A,C) and on(C,B)

Database: (updated)
holding(C)
clear(A)
clear(B)
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
~~handEmpty~~

STRIPS Planning Example

9. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A)}

~~holding(C) and clear(B)~~
Stack(C,B)
on(A,C)
Stack: on(A,C) and on(C,B)

Database: (unchanged)
holding(C)
clear(A)
clear(B)
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
~~handEmpty~~

STRIPS Planning Example

10. Top item is rule, so:
    a. Remove rule from stack.
    b. Update database using rule.
    c. Keep track of rule (for solution).

stack(C,B):
ADD: {handEmpty,on(C,B),clear(C)}
DEL: {holding(C),clear(B)}

Solution: {Unstack(C,A), Stack(C,B)}

Stack:
~~Stack(C,B)~~
on(A,C)
on(A,C) and on(C,B)

Database: (updated)
clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

11. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
    a. Replace the goal with the instantiated rule;
    b. Place the rule's instantiated precondition formula on top of stack

Solution: {Unstack(C,A),Stack(C,B)}

Stack:
holding(A) and clear(C)
Stack(A,C) ⟷ on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)
clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

12. Since top goal is unsatisfied compound goal, list its unsatisfied sub-goals on top of it:

Solution: {Unstack(C,A),Stack(C,B)}

Stack:

- clear(C)
- holding(A)
- holding(A) and clear(C)
- Stack(A,C)
- on(A,C) and on(C,B)

Database: (unchanged)

- clear(C)
- on(C,B)
- ~~holding(C)~~
- clear(A)
- ~~clear(B)~~
- ~~on(C,A)~~
- ~~clear(C)~~
- onTabel(A)
- onTable(B)
- handEmpty

49

STRIPS Planning Example

13. Single goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B)}

Stack:

- ~~clear(C)~~
- holding(A)
- holding(A) and clear(C)
- Stack(A,C)
- on(A,C) and on(C,B)

Database: (unchanged)

- clear(C)
- on(C,B)
- ~~holding(C)~~
- clear(A)
- ~~clear(B)~~
- ~~on(C,A)~~
- ~~clear(C)~~
- onTabel(A)
- onTable(B)
- handEmpty

50

STRIPS Planning Example

14. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
    a. Replace the goal with the instantiated rule;
    b. Place the rule's instantiated precondition formula on top of stack

Solution: {Unstack(C,A),Stack(C,B)}

Stack:

ontable(A) and clear(A) and handEmpty

Pickup(A) ⟷ holding(A)

holding(A) and clear(C)

Stack(A,C)

on(A,C) and on(C,B)

Database: (unchanged)

clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

15. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B)}

~~ontable(A) and clear(A) and handEmpty~~

Pickup(A)

holding(A) and clear(C)

Stack(A,C)

on(A,C) and on(C,B)

clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

STRIPS Planning Example

16. Top item is rule, so:
    a. Remove rule from stack.
    b. Update database using rule.
    c. Keep track of rule (for solution).

pickup(A):
ADD: {holding(A)}
DEL: {onTable(A),clear(A),handEmpty}

Solution: {Unstack(C,A),Stack(C,B), Pickup(A)}

Stack:
~~Pickup(A)~~
holding(A) and clear(C)
Stack(A,C)
on(A,C) and on(C,B)

Database: (updated)
holding(A)
clear(C)
on(C,B)
~~holding(C)~~
~~clear(A)~~
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
~~onTabel(A)~~
onTable(B)
~~handEmpty~~

STRIPS Planning Example

16. Top item is rule, so:
    a. Remove rule from stack.
    b. Update database using rule.
    c. Keep track of rule (for solution).

pickup(A):
ADD: {holding(A)}
DEL: {onTable(A),clear(A),handEmpty}

Solution: {Unstack(C,A),Stack(C,B), Pickup(A)}

Stack:
~~Pickup(A)~~
holding(A) and clear(C)
Stack(A,C)
on(A,C) and on(C,B)

Database: (updated)
holding(A)
clear(C)
on(C,B)
~~holding(C)~~
~~clear(A)~~
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
~~onTabel(A)~~
onTable(B)
~~handEmpty~~

STRIPS Planning Example

17. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B), Pickup(A)}

Stack:

~~holding(A) and clear(C)~~
Stack(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

holding(A)
clear(C)
on(C,B)
~~holding(C)~~
~~clear(A)~~
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
~~onTabel(A)~~
onTable(B)
~~handEmpty~~

55

STRIPS Planning Example

18. Top item is rule, so:
    a. Remove rule from stack.
    b. Update database using rule.
    c. Keep track of rule (for solution).

stack(A,C):
ADD: {handEmpty,on(A,C),clear(A)}
DEL: {holding(A),clear(C)}

Solution: {Unstack(C,A),Stack(C,B), Pickup(A), Stack(A,C)}

Stack:

~~Stack(A,C)~~
on(A,C) and on(C,B)

Database: (updated)

on(A,C)
~~holding(A)~~
~~clear(C)~~
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
~~onTabel(A)~~
onTable(B)
handEmpty

56

STRIPS Planning Example

19. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B), Pickup(A), Stack(A,C)}

Stack: $\boxed{\text{on(A,C) and on(C,B)}}$

Database: (unchanged)

on(A,C)
on(C,B)
clear(A)
onTable(B)
handEmpty

STRIPS Planning Example

19. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B), Pickup(A), Stack(A,C)}

Stack: ~~on(A,C) and on(C,B)~~

Database: (unchanged)

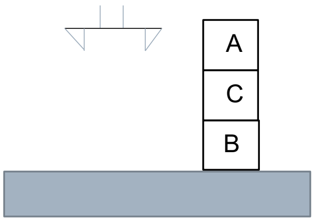
on(A,C)
on(C,B)
clear(A)
onTable(B)
handEmpty

STRIPS Planning Example

19. Compound goal on top of stack matches data base, so remove it:

Stack is empty, so stop.

Solution: {Unstack(C,A),Stack(C,B), Pickup(A), Stack(A,C)}

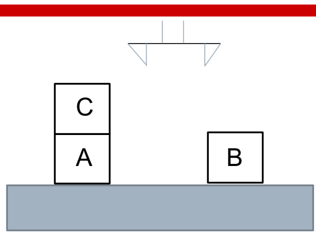


Final State

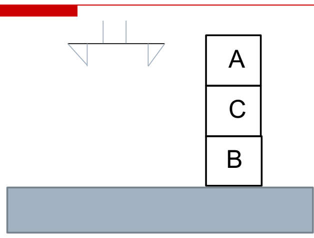
Stack:

on(A,C)
on(C,B)
clear(A)
onTable(B)
handEmpty

STRIPS Planning Example



Initial State



Final State

In solving this problem; we branched in the right direction.

In practice, searching can be guided by

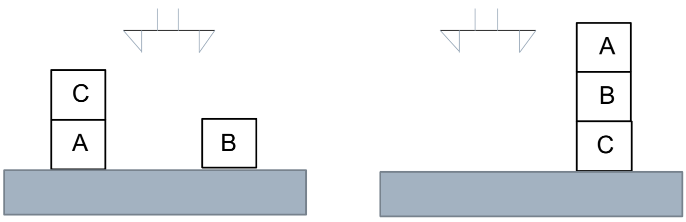
1. Heuristic information e.g., try to achieve "holding(x)" last.
2. Detecting unprofitable paths e.g., when newest goal set has become a superset of the original goal set.
3. Considering useful operator side effects (by scanning the stack).

Planning as Search

- Progression: forward chaining
  - like state-space search except for representation
  - inefficient due to large situation space to explore
- Regression: backward chaining
  - start from the goal state and solve its sub-goals (preconditions)
  - more efficient and goal-directed than progression (fewer applicable operators)

Sussman Anomaly

on(C,A)
clear(C)
clear(B)
onTable(A)
onTable(B)
handEmpty



Initial State   Final State

on(A,B)
on(B,C)
clear(A)
onTable(C)
handEmpty

Noninterleaved planners typically separate the goal into sub-goals:

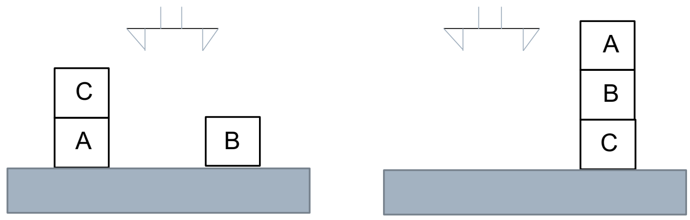
a. on(A,B)
b. on(B,C)

This is the Sussman anomaly; that illustrates a weakness of noninterleaved planning algorithms which were prominent in the early 1970s.

1. Suppose the planner starts by pursuing Goal 1. The basic step is to move C out of the way, then move A atop B. But while this sequence accomplishes Goal 1, the agent would be left with no option but to undo Goal 1 in order to pursue Goal 2.
2. If instead the planner starts with Goal 2, the most efficient solution is to move B. But again, the planner cannot pursue Goal 1 without undoing Goal 2:

Sussman Anomaly

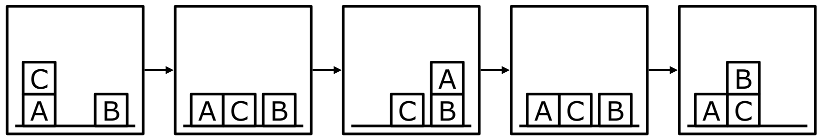
on(C,A)
clear(C)
clear(B)
onTable(A)
onTable(B)
handEmpty



Initial State   Final State

on(A,B)
on(B,C)
clear(A)
onTable(C)
handEmpty

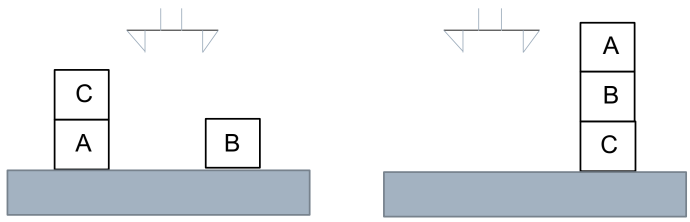
unstack(C,A) to clear(A) | stack(A,B) for **on(A,B)** | unstack(A,B) to clear(B) | stack(B,C) for **on(B,C)**



on(A,B) is undone to clear(B) when solving **on(B,C)**

Sussman Anomaly

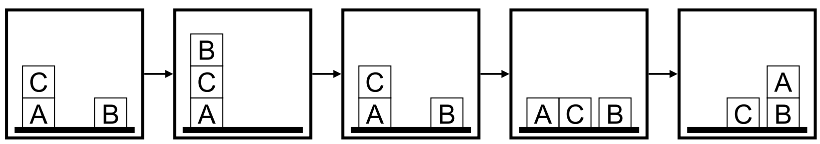
on(C,A)
clear(C)
clear(B)
onTable(A)
onTable(B)
handEmpty



Initial State   Final State

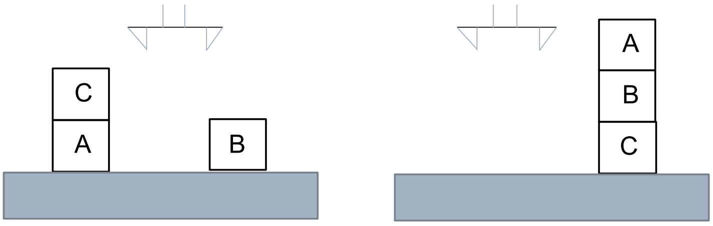
on(A,B)
on(B,C)
clear(A)
onTable(C)
handEmpty

stack(B,C) for **on(B,C)** | unstack(B,C) to clear(C) | unstack(C,A) to clear(A) | stack(A,B) for **on(A,B)**



on(B,C) is undone to clear(C) when solving **on(A,B)**

Sussman Anomaly



Initial State Final State

1. Begin work on ON(A,B) by clearing A i.e., putting C on table.
2. Achieve ON(B,C) by stacking B on C
3. Achieve ON(A,B) by stacking A on B.

We couldn’t do this using a stack within STRIPS; but interleaving!

Interleaving vs. Non-interleaving Planner

- Non-interleaving planner
  - $G_1 \wedge G_2$:
    either all the steps for achieving G_1 occur before G_2,
    or all of the steps for achieving G_1 occur after G_2
  - all of the steps for a sub/goal must occur “atomically”
  - e.g. STRIPS
  - can't solve the Sussman Anomaly
- Interleaving planner
  - can intermix order of sub/goal steps
  - can solve the Sussman Anomaly by interleaving steps:
    unstack(C,A), Pickup(B), Stack(B,C), Stack(A,B)