Methods of Artificial Intelligence

4. Planning I: Classical Planning

Nohayr Muhammad

Winter Term 2022/2023

November 25th, 2022

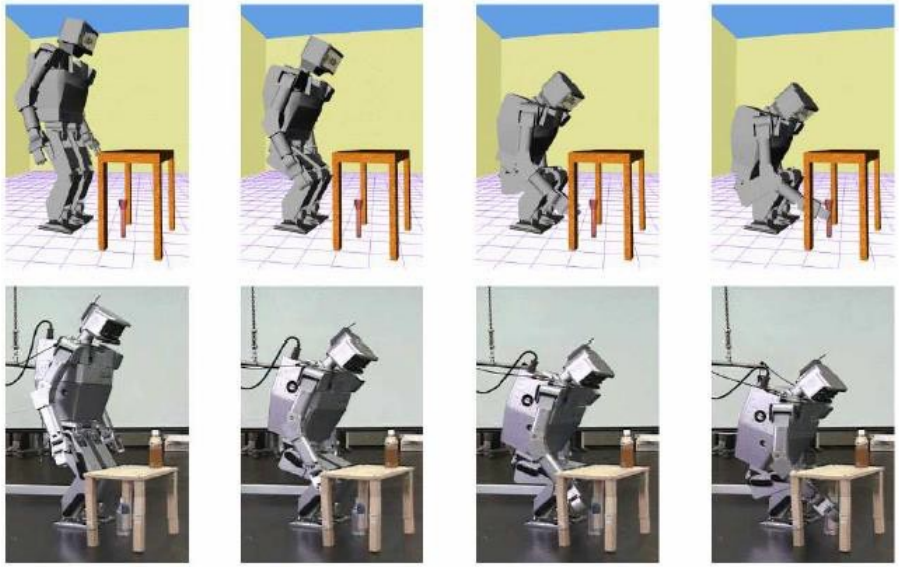
Today..

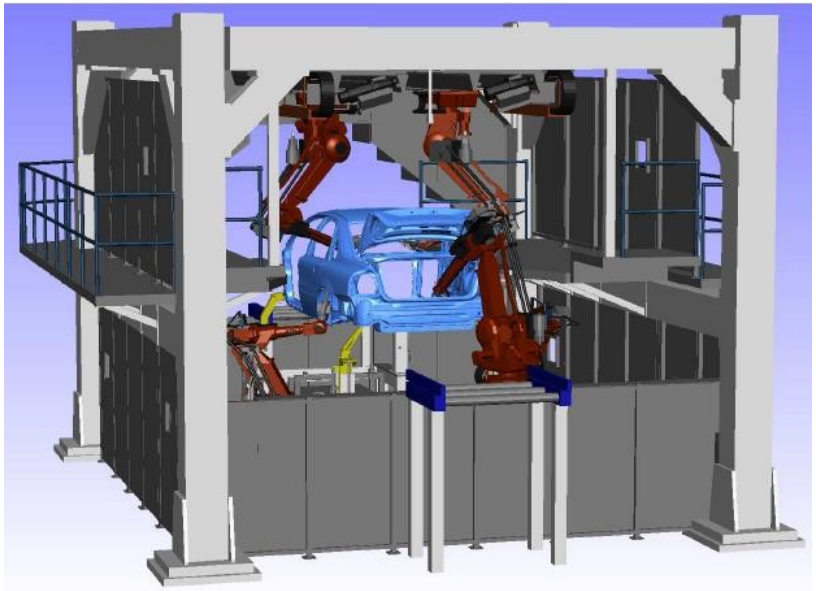
- An overview of Planning Problems and challenges
- Deductive planning: Situation Calculus
- State-based planning: Strips
- PDDL

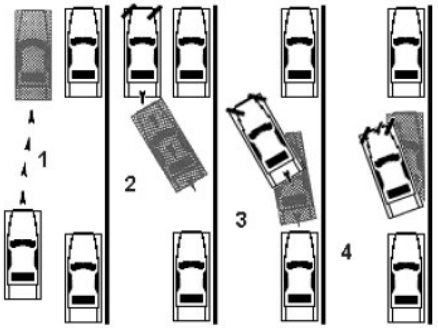
Planning Systems

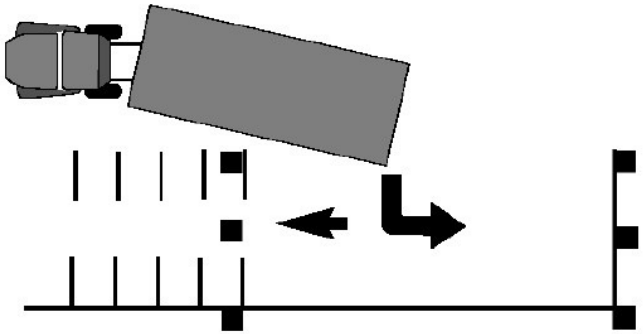
General Remarks

Planning Problems









Searching Vs. Planning

Local Search: We are not interested in the sequence of actions.

Classical Search: We are not interested in the internal representation of states. States are represented as a single entity (which may be quite a complex object, but its internal structure is not used by the search algorithm).

Planning: We are interested in both the sequence of actions and the internal representation of states. States have structured representations (collections of properties) which are used by the planning algorithm.

AI and Planning

AI planning deals with the

- formalization,
- implementation and
- evaluation

of algorithms for constructing plans.

Planning Formalized

Plan (classical view): find sequence of actions that transforms given state into goal state (in a fully observable, deterministic, static environment).

Plan (alternative view): assign an action to every possible state

Some approaches:

- **deductive** planning: e.g. Situation Calculus.
- **state-based** planning: e.g. Strips.
- **probabilistic** planning: e.g. Markov Decision Processes

Components of Planning Systems

Planning language

- often subsets of FOL
- states and goals can often be represented as ‘conjunctions’ of ground literals
- operators can be described as rules
- operator semantics: change state by applying an action

Algorithms for constructing a plan

Potential Challenges in Planning

Frame problem:

actions change the world, but many things remain unchanged

Qualification problem:

actions may have an endless list of preconditions

Ramification problem:

actions may have unintended side-effects

Examples

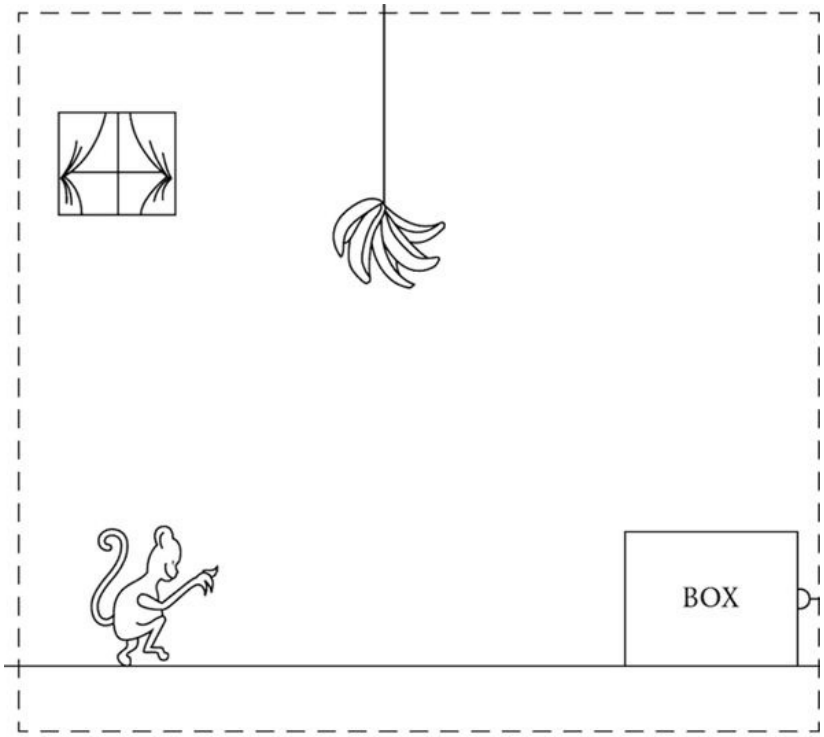
Kiva Systems Warehouse Automation (Youtube)

- Intelligent Warehouse Systems manage hundreds of robots
- items are stored in portable storage units (PSUs)
- robots transport PSUs to human operators that pick and pack items

Frame problem:

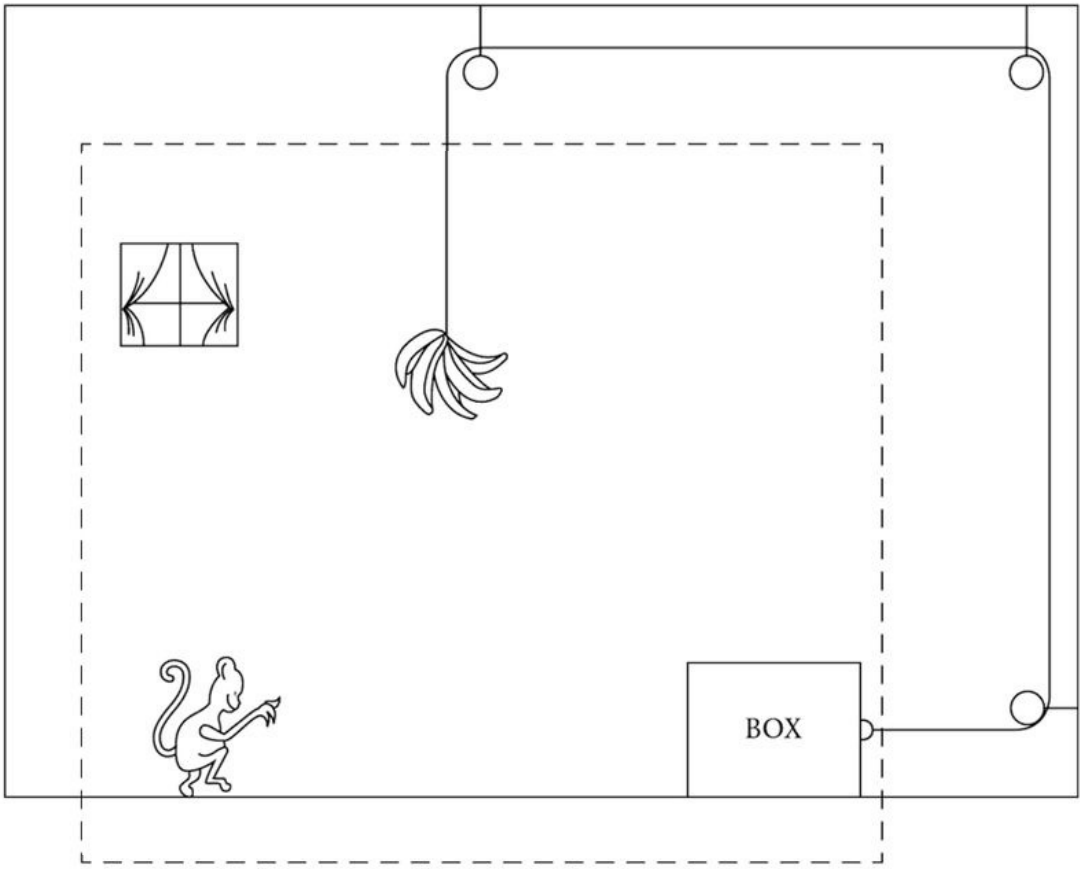
- When moving one PSU, state of all other PSUs remains unchanged
- Planning system has to transfer this information from previous to future state in an efficient way
- Question: What is an an adequate set of data that represents a situation and environment?

Frame Problem



The Monkey and bananas problem.
Source: M.A. Boden, Artificial Intelligence and Natural Man (1977:387) MIT Press

Frame Problem

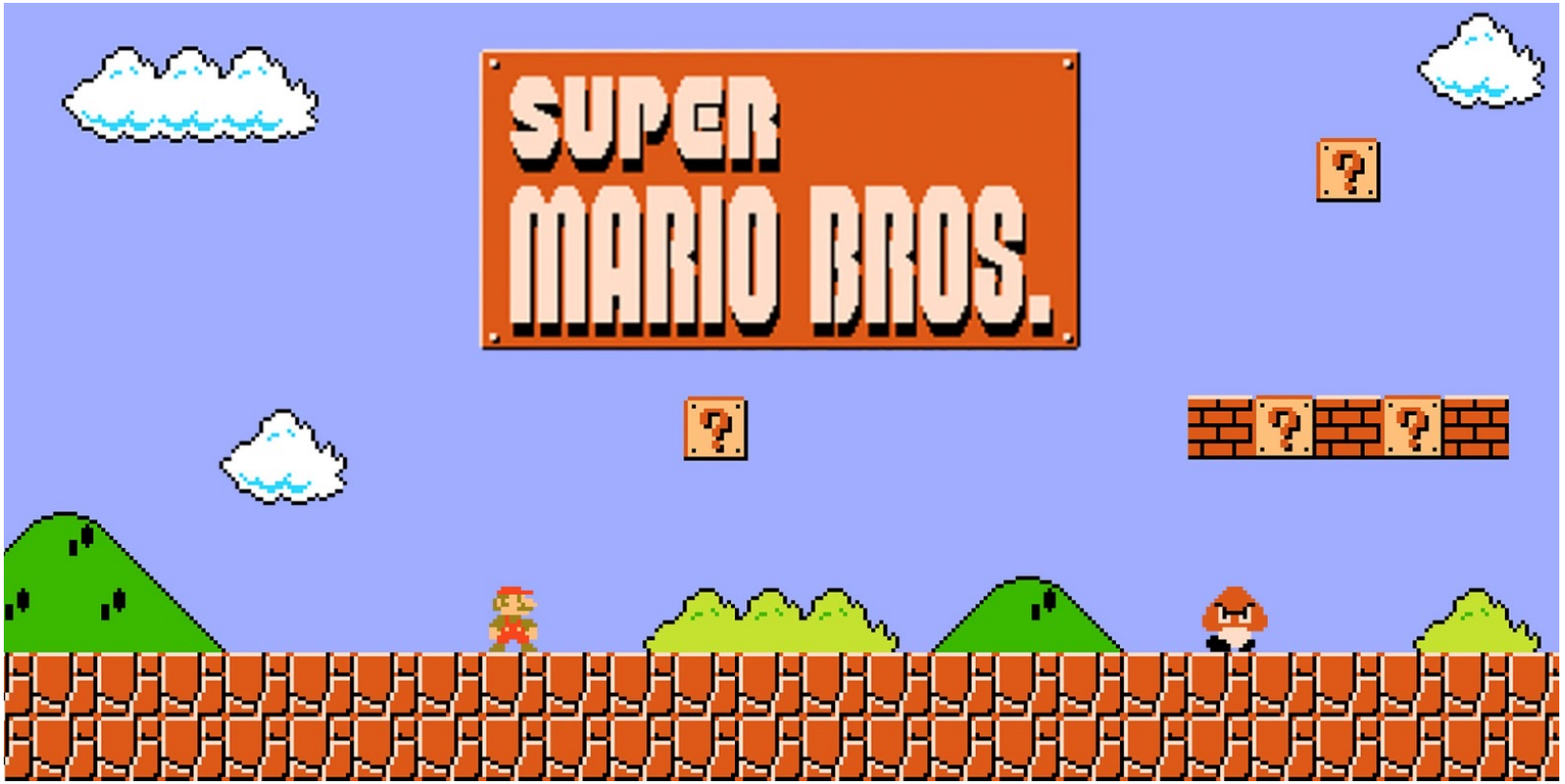


The Monkey and bananas problem.
Source: M.A. Boden, Artificial Intelligence and Natural Man (1977:387) MIT Press

Frame Problem

https://www.nintendo.de/Spiele/NES/Super-Mario-Bros-803853.html

Frame Problem



https://www.nintendo.de/Spiele/NES/Super-Mario-Bros-803853.html

Examples

Kiva Systems Warehouse Automation (Youtube)

Qualification problem: we may be unable to move a robot because

- it ran out of power
- it has a malfunction
- the path is blocked by another robot
- …

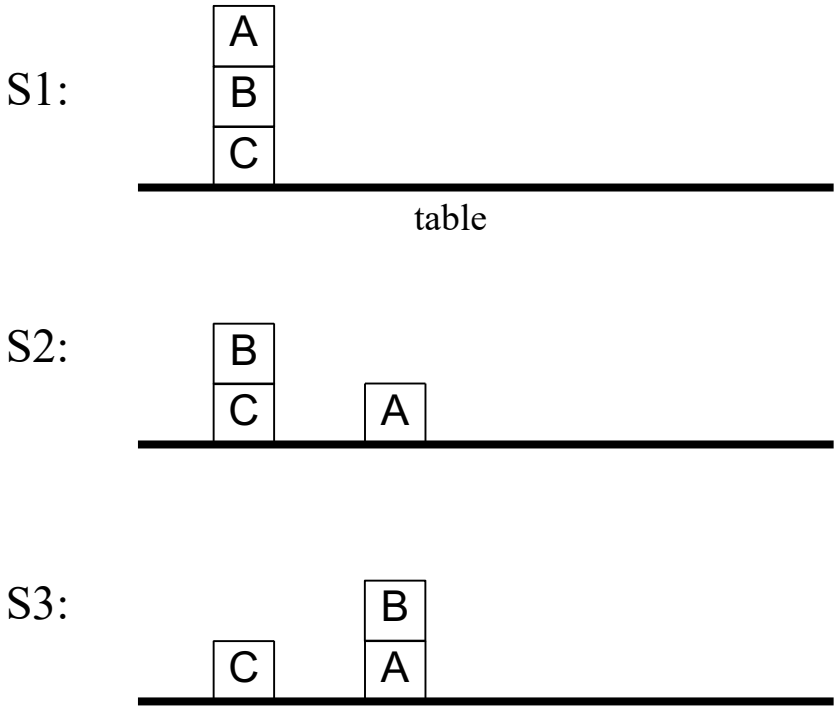
Examples

Kiva Systems Warehouse Automation (Youtube)

Ramification problem: when robot moves a PSU with one particular item

- it also moves all other items stored in the PSU

A Simple Planning Domain: Blocksworld



Logical Representation:

- constants: `A,B,C` (blocks)
- Predicate symbols:
  `on(block,block)`,
  `ontable(block)`
  `clear(block)`
- Function symbols:
  `put(block,block)`,
  `put_on_table(block)`

Logical State Representation

S1:

- `onTable(C)`
- `on(B,C)`
- `on(A,B)`
- `clear(A)`

S2:

- `onTable(C)`
- `onTable(A)`
- `on(B,C)`
- `clear(B)`
- `clear(A)`

Exercise

S1:

- `onTable(C)`
- `on(B,C)`
- `on(A,B)`
- `clear(A)`

Given the state above. Give an example of the

- Frame problem
- Qualification problem

S1:

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

- Frame problem: when putting A on the table, A stays clear, B stays on C and C stays on the table

- Qualification problem: we may be unable to put A on the table because it is too heavy

Deductive Planning Systems

The Situation Calculus

Deductive Planning

Basic example of Deductive Planning is the **Situation Calculus**

History:

- Introduced by McCarthy (1963)
- used for plan construction by resolution by Green (1969).
- Further developments in the context of the event calculus (Kowalski, 1986) and Fluent calculus (Thielscher, 2000).

Features:

- States are described by logical formulas
- Operators are described by logical axioms
- Plans can be computed by Theorem Provers
- Situation Calculus does not use closed-world assumption

Situations and Fluents

- Predicates that change over time get an additional argument
- This additional argument represents the current situation
- We call these predicates fluents

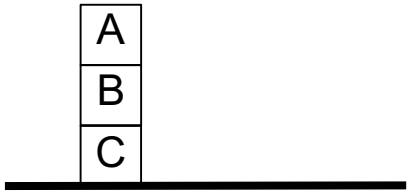
S1:

- onTable(C, S1)
- on(B, C, S1)
- on(A, B, S1)
- clear(A, S1)

- Formally, situations are terms. They are either
  - Atomic situations (constants) like S1 or
  - Complex situations (created from atomic situations using operators)

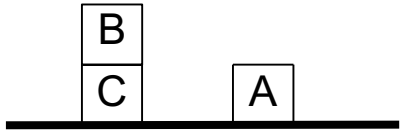
Example

S1:



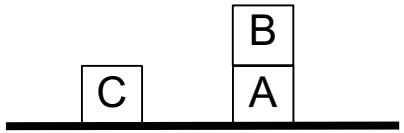
- onTable(C, S1)
- on(B,C, S1)
- on(A,B, S1)
- clear(A, S1)

put_on_table(A,S1)



- onTable(C, put_on_table(A,S1))
- on(B,C, put_on_table(A,S1))
- onTable(A, put_on_table(A,S1))
- clear(A, put_on_table(A,S1))
- clear(B, put_on_table(A,S1))

put(B,A, put_on_table(A,S1))



- onTable(C, put(B,A, put_on_table(A,S1)))
- on(B,A, put(B,A, put_on_table(A,S1)))
- onTable(A, put(B,A, put_on_table(A,S1)))
- clear(B, put(B,A, put_on_table(A,S1)))
- clear(C, put(B,A, put_on_table(A,S1)))

Effect Axioms

Remark: In the following different variables refer to different entities.This must be made explicit by adding a statement like X≠Y in some frameworks

Effect Axioms: *describe preconditions and effects of operators*

$$on(X, Y, put(X, Y, S)) \leftarrow clear(X,S) \wedge clear(Y,S)$$
$$clear(Z, put(X, Y, S)) \leftarrow on(X,Z,S) \wedge clear(X,S) \wedge clear(Y,S)$$

$$clear(Y, put\_on\_table(X, S)) \leftarrow on(X,Y,S) \wedge clear(X,S)$$
$$ontable(X, put\_on\_table(X, S)) \leftarrow clear(X,S)$$

Frame Problem Revisited

Frame Problem in Situation Calculus:

we do not only need axioms describing the effects of actions, but also axioms describing what remains unchanged.

Frame axioms describe what relations remain unaffected by actions

Example of a frame axiom for put(X,Y,S):

on(Y, Z, put(X, Y, S)) ← on(Y, Z, S)

(After putting X on Y, Y will still be on Z if this was true before)

Frame Axioms

Frame Axioms: *describe properties of previous situation that remain true*

clear(X,put(X,Y,S)) ← clear(X,S) ∧ clear(Y,S)

clear(Z,put(X,Y,S)) ← clear(X,S) ∧ clear(Y,S) ∧ clear(Z,S)

ontable(Y,put(X,Y,S)) ← clear(X,S) ∧ clear(Y,S) ∧ ontable(Y,S)

ontable(Z,put(X,Y,S)) ← clear(X,S) ∧ clear(Y,S) ∧ ontable(Z,S)

on(Y,Z,put(X,Y,S)) ← clear(X,S) ∧ clear(Y,S) ∧ on(Y,Z,S)

on(W,Z,put(X,Y,S)) ← clear(X,S) ∧ clear(Y,S) ∧ on(W,Z,S)

Planning Problem

- When operators have been described, planning problems can be solved by
  1. Defining initial state
  2. Defining goal state
  3. Applying Theorem Prover

Initial state (logical atoms)

$on(d,c,s_1)$ $on(c,a,s_1)$

$clear(d,s_1)$ $clear(b,s_1)$

$ontable(a,s_1)$ $ontable(b,s_1)$

Goal (Theorem)

$on(a,b,G) \wedge on(b,c,G)$

Exercise

Consider the following fluents and operator for a warehouse system:

- Parking(P, S): PSU P is parked in situation S (can be picked up)
- Available(R, S): robot R is free for use in situation S
- Carries(R, P, S): robot R carries PSU P in situation S

- pick(R, P, S): let robot R pick PSU P in situation S

Give an example of an effect axiom and a frame axiom for the operator pick

Possible Solutions

Some Effect Axioms

- Carries(R, P, pick(R, P, S)) $\leftarrow$ Parking(P, S) $\wedge$ Available(R, S)
- $\neg$ Parking(P, pick(R, P, S)) $\leftarrow$ Parking(P, S) $\wedge$ Available(R, S)
- $\neg$ Available(R, pick(R, P, S)) $\leftarrow$ Parking(P, S) $\wedge$ Available(R, S)

Some Frame Axioms

- Carries(R, P, pick(R2, P2, S)) $\leftarrow$ Carries(R, P, S)
- Parking(P, pick(R2, P2, S)) $\leftarrow$ Parking(P, S)
- Available(R, pick(R2, P2, S)) $\leftarrow$ Available(R, S)

Planning Problems in Situation Calculus

Qualification problem: Preconditions in effect axioms can theoretically solve the qualification problem.

Frame problem: Frame axioms can theoretically solve the frame problem.

Ramification problem: Additional axioms can theoretically solve the ramification problem (”if an object is attached to another object and one of the objects is moved, the other object moves, too”).

In practice, however, it is difficult to guarantee that all cases are covered, and runtime may suffer

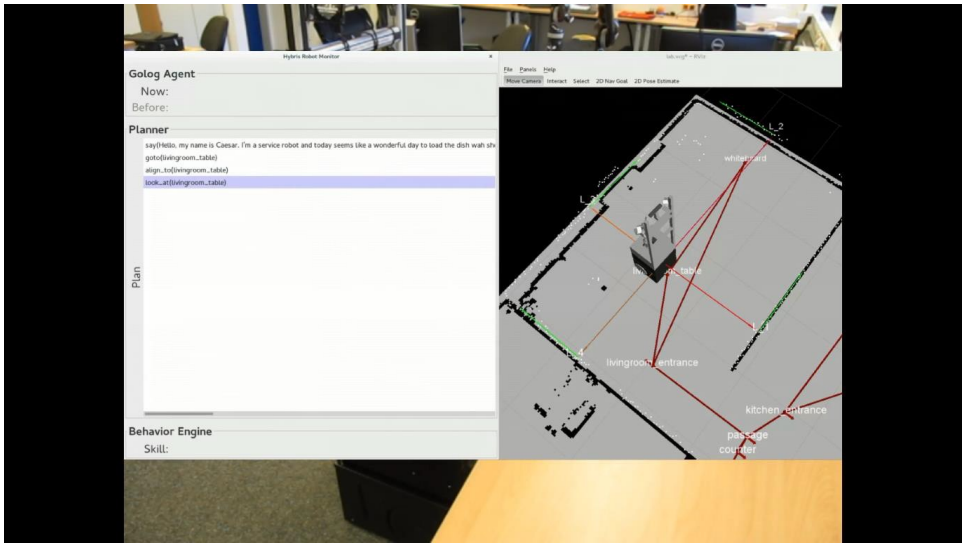
Planning with Golog

Golog combines

- Logic Programming and
- Situation Calculus

proc *DeliverCoffee*
  while $\exists x.WantsCoffee(x)$ **do**
    $\pi x.WantsCoffee(x); ServeCoffee(x)$
  endWhile
  $goto(Home)$
endProc

proc *ServeCoffee*(x)
  if $\neg HasCoffee(R)$ **then** $goto(CoffeeM); loadCoffee$ **endIf**
  $goto(x); giveCoffee(x)$
endProc



https://www.youtube.com/watch?v=qLIYJ2NiGq0

Conclusions

- **Situation Calculus** is the basic example of Deductive Planning
- States are represented by sets of fluents (logical atoms)
- Operators and their effects are described by logical axioms
- Planning can be reduced to Theorem Proving
- Deductive Reasoning ability comes for free
- Verification and computational performance can be a problem

State-Based Planning: STRIPS

State-Based Planning

Basic example of State-Based Planning is **STRIPS**
(**St**anford **R**esearch **I**nstitute **P**roblem **S**olver)

History:

- Developed by Fikes and Nilsson (1971)
- Foundation of modern state-based planners (PDDL)

Features:

- States are represented by sets of atoms
- Operators act directly on states
- Planning can be performed by Classical Search or tailor-made algorithms
- STRIPS uses closed-world assumption

States in STRIPS

State descriptions correspond to sets of ground atoms

Domain D = {a, b, c, d}

Predicates P = {ontable^1, clear^1, on^2}

s = {ontable(a), ontable(c), ontable(d), clear(a), clear(b), clear(d), on(b,c)}

Goals are usually also sets of ground atoms

G = {on(b,c), ontable(c)}.

Goal state: A state s is called a goal state if $G \subseteq s$.

Closed World Assumption: all atoms in state s are assumed to be true, all other are assumed to be false

Strips Operator Schemes

Operator scheme consists of

- An operator name followed by a sequence of variables and
- three lists that represent
  - The preconditions (PRE)
  - The positive effects (ADD)
  - The negative effects (DEL)

Variables serve as placeholders for domain objects

Operator *put:*

Operator:	put(X,Y)
PRE:	{ontable(X), clear(X), clear(Y)}
ADD:	{on(X,Y)}
DEL:	{ontable(X), clear(Y)}

Operator Instantiation

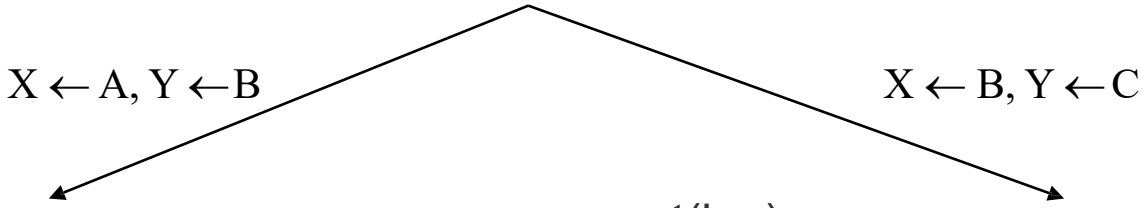
We apply an operator by instantiating the variables in the operator scheme (formally, we apply a substitution)

put(X,Y)

PRE: {ontable(X), clear(X), clear(Y)}

ADD: {on(X,Y)}

DEL: {ontable(X), clear(Y)}



put(a,b)

PRE: {ontable(a), clear(a), clear(b)}

ADD: {on(a,b)}

DEL: {ontable(a), clear(b)}

put(b,c)

PRE: {ontable(b), clear(b), clear(c)}

ADD: {on(b,c)}

DEL: {ontable(b), clear(c)}

Operator Application

- An instantiated operator o can be applied in a state s if

  $Pre(o) \subseteq s$

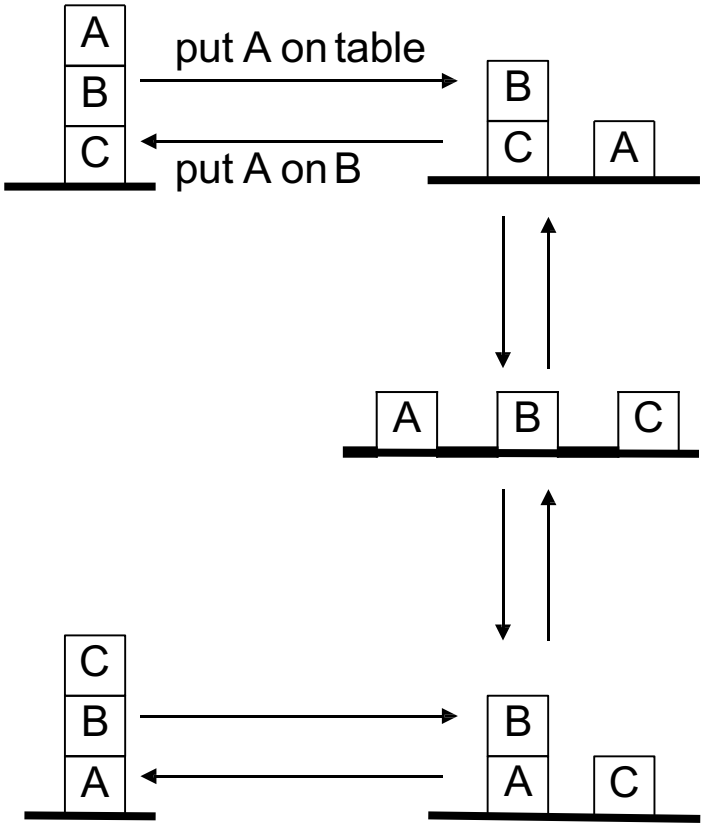
- Applying o will change the current state s to a new state s'

  $s' = (s \setminus Del(o)) \cup Add(o)$

(delete elements from del-list and add elements from add-list)

Example: State-Space Model

- Operator application spans a search space
- s_i is connected to s_j iff s_j results from s_i when applying a single operator



Example: Blocksworld Domain

The put operator for two different situations:

put(X, Y)
PRE: {ontable(X), clear(X),clear(Y)}
ADD: {on(X, Y)}
DEL: {ontable(X), clear(Y)}

put(X, Y)
PRE: {on(X, Z), clear(X), clear(Y)}
ADD: {on(X, Y), clear(Z)}
DEL: {on(X, Z), clear(Y)}

The put_on_table operator:

put_on_table(X)
PRE: {clear(X), on(X, Y)}
ADD: {ontable(X), clear(Y)}
DEL: {on(X, Y)}

Example of a Blocksworld Problem:

Blocksworld domain
GOAL: {on(A,B), on(B,C)}
INITIAL STATE:
{on(D, C), on(C, A), clear(D), clear(B), ontable(A), ontable(B)}

Exercise

Consider again the atoms:

- Parking(P): PSU P can be picked up
- Available(R): robot R is available for transportation
- Carries(R, P): robot R carries PSU P

- Define a Strips-Operator pick(R, P) that lets robot R pick PSU P
- How can you solve the frame problem for pick?

Solution

pick(?R,?P, ?S)

PRE: {Parking(P), Available(R)}

ADD: {Carries(R,P)}

DEL: {Parking(P), Available(R)}

In STRIPS, everything that we do not change explicitly, remains unchanged – therefore we do not need any frame axioms

Comparison

Situation Calculus

Effect Axioms

- Carries(R, P, pick(R, P, S)) ← Parking(P, S) ∧ Available(R, S)
- ¬ Parking(P, pick(R, P, S)) ← Parking(P, S) ∧ Available(R, S)
- ¬ Available(R, pick(R, P, S)) ← Parking(P, S) ∧ Available(R, S)

Frame Axioms

- Carries(R, P, pick(R2, P2, S)) ← Carries(R, P, S)
- Parking(P, pick(R2, P2, S)) ← Parking(P, S)
- Available(R, pick(R2, P2, S)) ← Available(R, S)

STRIPS

pick(?R,?P, ?S)

PRE: {Parking(P), Available(R)}

ADD: {Carries(R,P)}

DEL: {Parking(P), Available(R)}

Implicit in STRIPS

Planning Problems in STRIPS

Qualification problem: Pre-list.

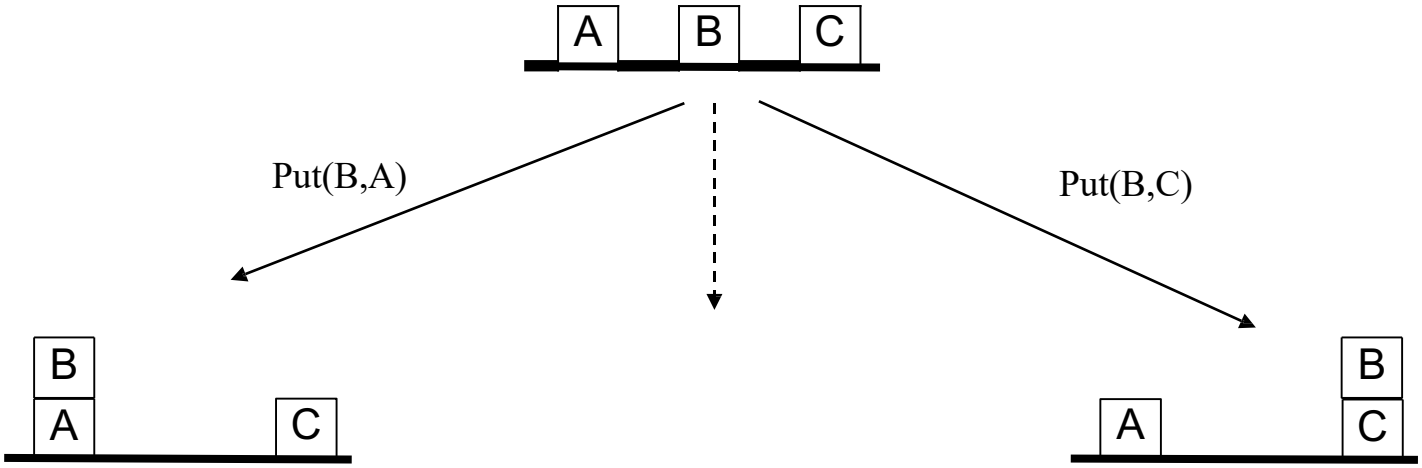
Frame problem: STRIPS transfers everything to the next state automatically if not stated otherwise (del-list)

Ramification problem: Add-list, Del-List

STRIPS ‘solves’ the Frame problem and allows a more convenient problem representation

Planning via Classical Search

- Planning in STRIPS can be seen as a classical search problem
- States are sets of atoms
- Successors in search space are obtained by applying operators
  - one successor for each applicable operator
  - successor state is obtained from current state by deleting elements from del-list and adding elements from add-list



Planning Algorithms

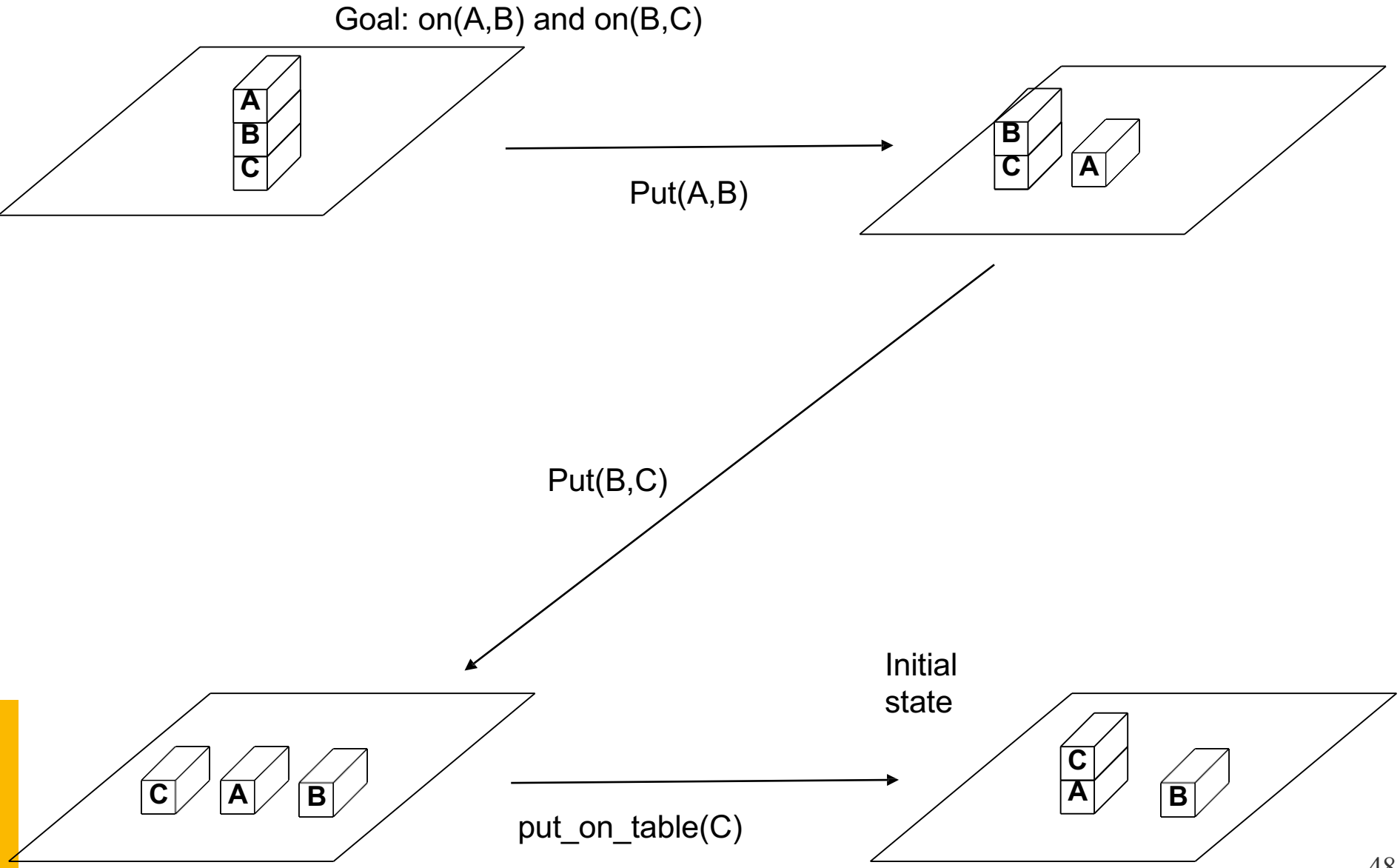
Classical Search (Forward Planning):

- uninformed search: usually variants of depth-first search (performance)
- informed search with variants of A*

Backward Planning (relevant-state search):

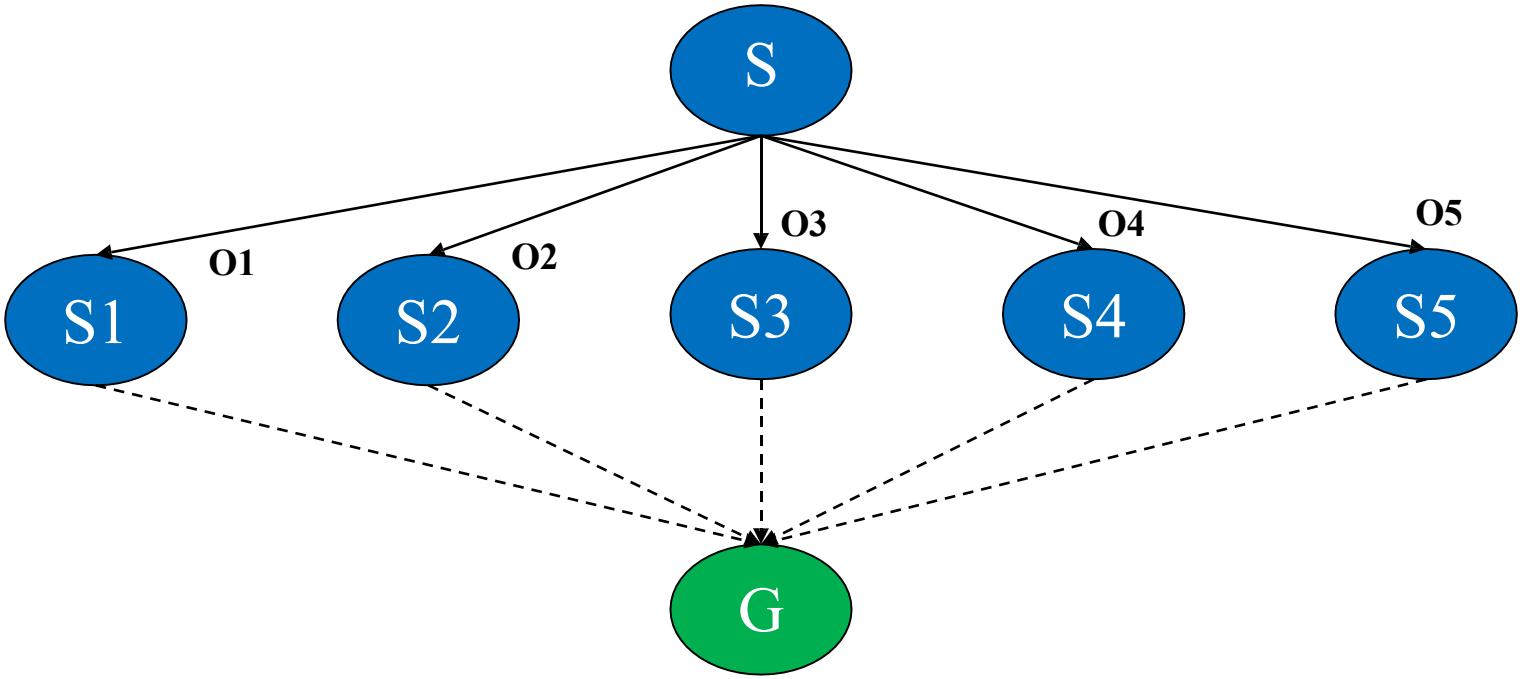
- Start from goal and apply operators backwards until a sequence of operators is found that reaches an initial state
- Advantage: may consider less irrelevant actions

Example: Backward Search (idealized)



Planning Heuristics

- search must be guided by good heuristics
- Heuristic: estimates the cost to from state to goal state
- This can be achieved by relaxing the problem to an easier problem and solving the relaxed problem

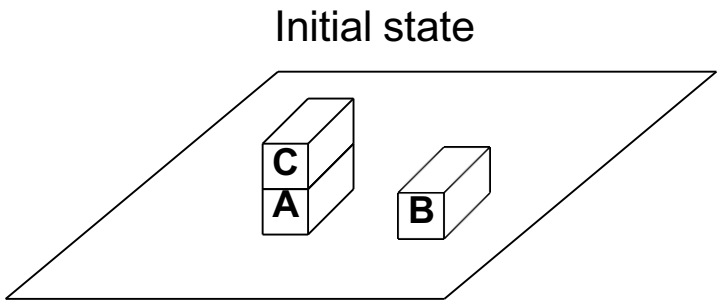


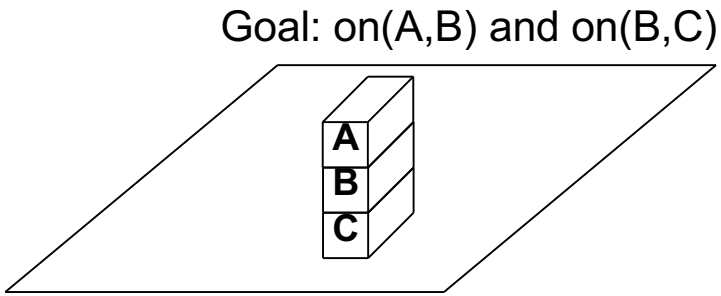
Ignore Pre-Del Heuristic

- Ignore Pre-Del Heuristic:
  - relax problem by ignoring pre- and del-list of operators
  - each operator then satisfies a number of goals (0, 1, 2, …)
  - given a state, compute minimal number of operators that must be applied to satisfy goal
  - this corresponds to the **set-cover** problem (that is still NP-hard)

  - we can do better by using an approximate greedy solution that can be computed in polynomial time (or apply local search)
  - however, the greedy solution may overestimate the cost
  - hence, the corresponding heuristic is not admissible
    (recall that A* guarantees optimality only for admissible heuristics)

Example: Blocksworld





- In Blocksworld, goal can be identified with
  U = {onTable(C), on (B,C), on(A,B)}

- Each operator can be identified with a subset of U
  - Put(A,B): {on(A,B)}
  - Put(A,C): {}
  - Put(B,C): {on(B,C)}
  - …

- Estimated cost comes down to number of unsatisfied literals (why?)

Example: Warehouse

- Again goal can be identified with set (e.g. order at location) U = {ItemAt(i1, x, y), ItemAt(i2, x, y), ItemAt(i3, x, y) }
- Intuitively, we can think of U as a set of items
- PSUs correspond to subsets of U (the items in U that they carry)
- The minimal set cover of U corresponds to PSUs that we have to carry to location (x,y)
- Estimated cost comes down to minimal number of PSUs that we have to carry

Ignore Del Heuristic

- Ignore Del Heuristic:
  - relax problem by ignoring only del-list of operators
  - no operator will undo goals in relaxed problem
  - relaxed problem is still NP-hard

  - approximate solution can again be computed in polynomial time
  - corresponding heuristic is again not admissible

Conclusions

- **STRIPS** is the basic example of State-Based Planning
- States are represented by sets of atoms
- Operators are described by 3 lists (pre, add, del)
- Frame problem is ‘solved’ by leaving everything unchanged that is not changed explicitly
- Planning comes down to classical search
- State-based planning can be more efficient than deductive planning, but planning remains hard in general

PDDL

Beyond STRIPS

- Planning Domain Definition Language (PDDL) extends STRIPS significantly
- Development is driven by the International Planning Competition (IPC) since 1998

PDDL supports in particular

- Numeric fluents
  - Contains(P, I, 100): PSU P contains 100 instances of item I
- Derived Predicates
  - PSUAt(P, X, Y) :- Carries(R, P), RobotAt(R, X, Y).
  - ItemAt(I, X, Y) :- Contains (P,I, N), N>0, PSUAt(P, X, Y).

PDDL Syntax basics

- Planning Domain Definition Language (PDDL) syntax is LISP-like and might look unfamiliar at first

- The syntax is very simple:
  - Instead of `f(x,y,z)` as in modern languages
  - you have `(f x y z)`

- A PDDL representation contains at least:
  - a domain definition (types, actions)
  - a problem definition (initial state, goal)

- PDDL is a declarative language that represents knowledge
- Can be used with many different search and planning algorithms

PDDL simple example

```
(define (domain blocksworld)
  (:requirements :strips)
  (:action move
     :parameters (?b ?t1 ?t2)
     :precondition (and (block ?b) (table ?t1) (table ?t2)
                        (on ?b ?t1) (not (on ?b ?t2))
     :effect (and (on ?b ?t2)) (not (on ?b ?t1))))
)
```

Name of our domain

Configure syntax and language options

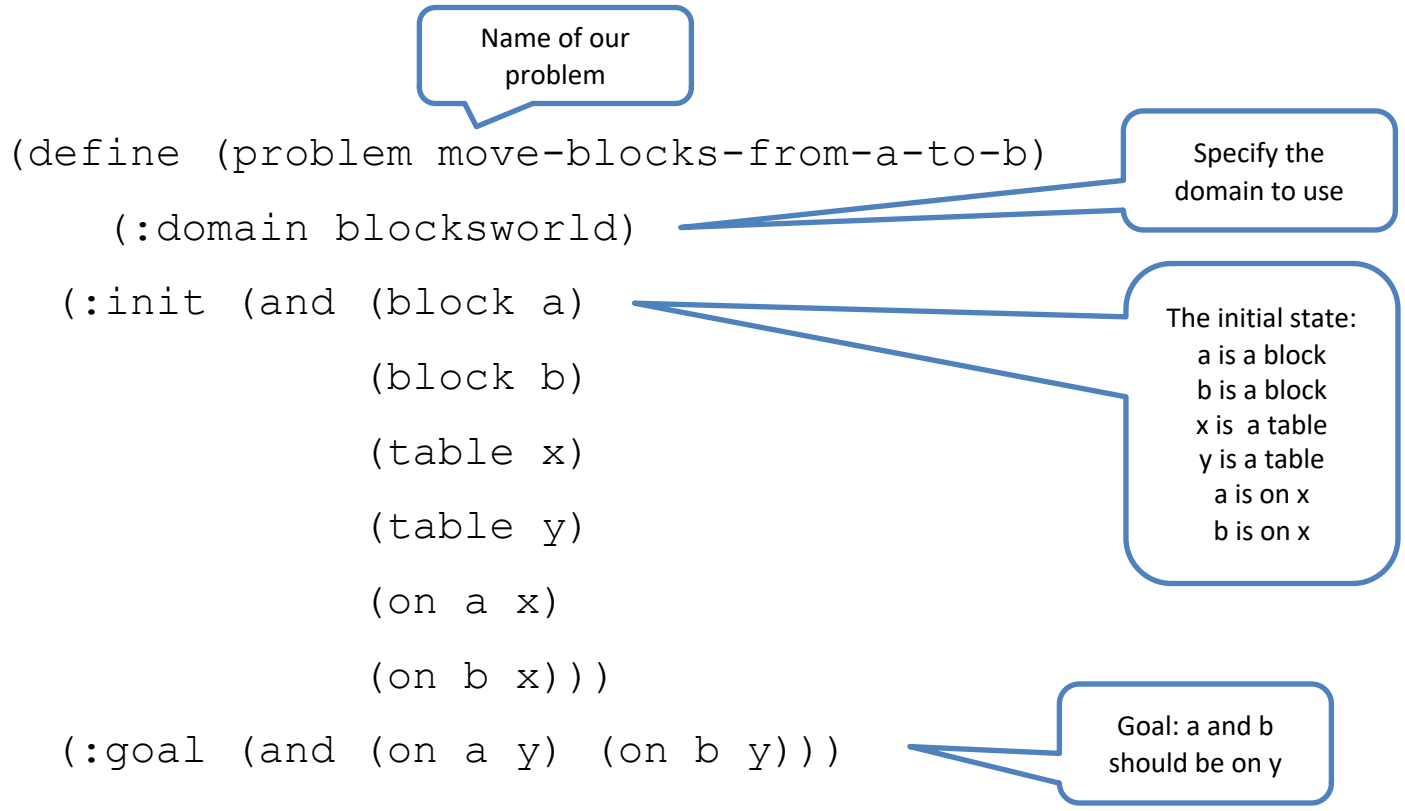
Define action „move“

with 3 parameters

List of 5 AND-connected preconditions

Combined ADD and DEL list

PDDL simple example



Demo: https://stripsfiddle.herokuapp.com/ (Blocks World 1)

PDDL example: Magic World

Use typed objects

```
(define (domain magic-world)
   (:requirements :strips :typing)
   (:types player location monster element chest)

   (:action move
      :parameters (?p - player ?l1 - location ?l2 - location)
      :precondition (and (at ?p ?l1) (border ?l1 ?l2) (not (guarded ?l2)))
      :effect (and (at ?p ?l2) (not (at ?p ?l1)))
   )
)
```

PDDL example: Magic World

```
(define (problem move-to-castle)
   (:domain magic-world)
   (:objects
      npc - player
      town field castle - location
   )
   (:init
      (border town field)
      (border field castle)
      (at npc town)
   )
   (:goal (and (at npc castle)))
)
```

Try it on
https://stripsfiddle.herokuapp.com/

Exercise:
Introduce fighting