Robotics II - Planning

Lecture 1/3

M.Sc. Oscar Lima

October, 21th 2019

Contents of this lecture

- Introduction
  - Important distinction
  - Resources
  - Motivation
  - Prerequisites
  - Objective of this lecture series
  - Domain specific vs domain independent planning
- Background
  - Decision making in robotics
  - State machines
  - Petri nets
  - Behavior trees
  - Reinforcement learning
  - Task Planning
  - Reasoning (KnowRob)
  - Propositional Logic
  - FOL
- Formalization
  - Assumptions
  - Grounding
  - Predicate - fact
  - Knowledge base - state space representation
  - Solution - plan
  - State, Initial state
  - Goal
  - Operator
  - Action
  - Applicability
  - Preconditions
  - Effects
  - State transition system
  - Close world assumption
  - Exercise
- Planning representations
  - State space search
  - Forward search (applicable)
  - Backward search (relevant)

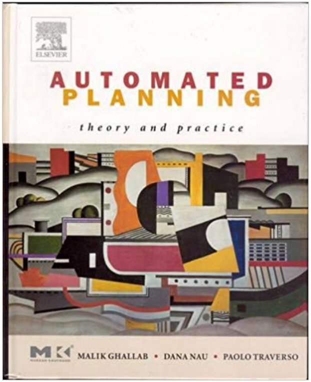
Introduction

Important distinction

- **Task planning ≠ Motion planning**
- Task planning : high level decision making at the symbolic level (e.g. move from A to B)
- Motion planning : how to move (geometrically) in the environment while satisfying some constraints (e.g. move from 0.5,0.5 to 10.0, 10.0, do not collide with environment)
- Until we reach the motion planning section (second half of 3rd lecture):
  - planning = task planning
- Motion planning:
  - Path planning (navigation 2D)
  - Motion planning (robot manipulators)
  - Grasp planning (robot end effector)

Resources (Literature)

- Ghallab et al. 2004,
  Automated Planning: theory and practice
  2004, Elsevier



This slide contains clickable links

- AI Planning course from University of Edinburgh
- Prof. Dana Nau lecture slides
- Prof. Joerg Hoffmann on heuristic search
- Prof. Dr. Amanda Coles, Introduction to AI Planning (Kings College London)
- Prof. Dr. Joerg Hoffman, Planning course material (Saarland University)
- ICAPS Conference - (Scientific articles) recorded presentations
- Uni Freiburg AI planning lecture slides
- PDDL online editor
- Visual Studio extension w/ tools to facilitate PDDL modeling

- What problem does it solve? / What is planning?
  - **R1:** Explicit offline deliberation process that chooses and organizes actions by anticipating their outcomes
  - **R2:** Useful to decompose a high level goal into a detailed sequence of symbolically parameterized actions that when followed correctly, solve a goal
  - e.g. (in robotics) bring me a coke would mean ...
- This however... represents only half of the answer: execution comes next and is usually interleaved[1]
- In a nutshell, task planning:
  - Useful for decision making in robotics
  - Intuitive for humans (based on logic)
  - Mature, well established research area

How do I grasp the coke??? mm I have no gripper...

[1] but we will cover both! (Planning and Execution)

Prerequisites (Task Planning)

For this lecture I will assume you are familiar with:

- Graph theory
- Basic search algorithms (BFS, DFS)
- Complexity analysis (Big O notation)
- First order logic (quick recap is provided)
- Propositional logic (quick recap is provided)
- General knowledge about robotics (or at least virtual agents...)

Objective of this lecture series

- Task planning can be an entire course on its own...
- Introduction / Big picture
- Get you interested in the topic
- Cover the basics
- Overview of the field from a research perspective
- Provide links for further reading (slides with plenty of links)

Domain specific vs domain independent planning

- Task Planning was not made for robotics!
- Tool for generic problem solving
- Used across multiple domains (logistics, white collar hacking, oil extraction, robotics, etc.)
- More to come once we cover heuristics...

Task Planning Background

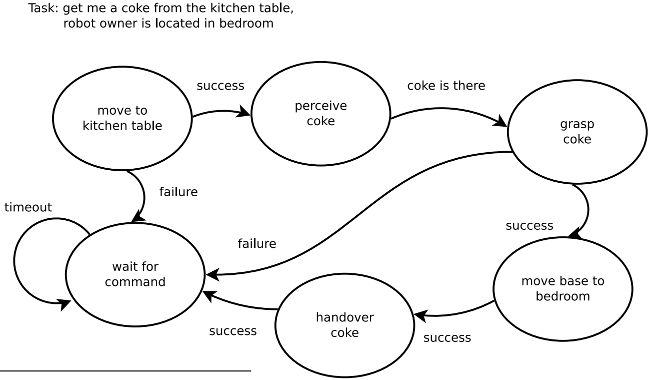
- What are our options for decision making in robotics?

Some options include: State machines, Petri nets, Behavior trees, RL[2], **Task Planning**, Reasoning (KnowRob), and many more...

[2] Reinforcement learning

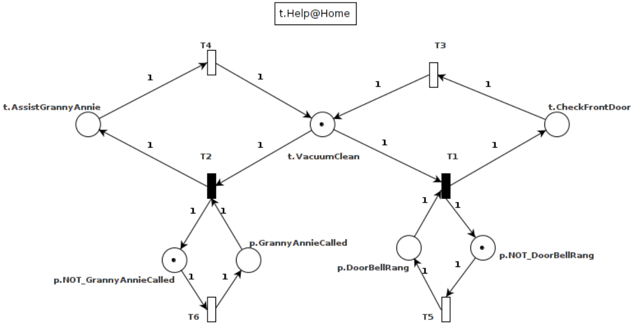
Finite State machine (aka Automata)

- Widely used in robotics
- Conditional task switch structure
- One way control transfer, "go to" considered harmful[3] [Colledanchise and Ogren 2017]
- Hard to maintain / extend
- Can be hierarchical
- e.g.



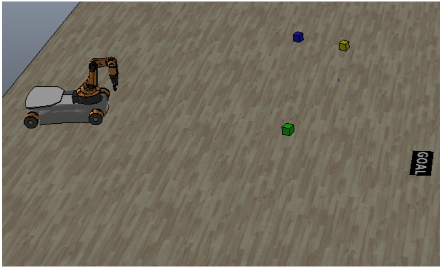
[3] Dijkstra: Too primitive, an invitation to make a mess of one's program

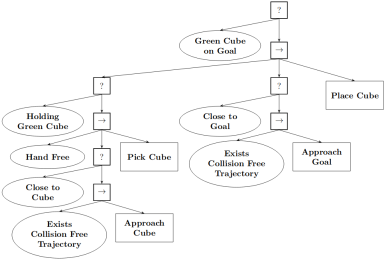
- default task: vacuum
- sub-task: attend door if rings
- sub-task: attend Granny Annie when she calls
- e.g.



Behavior trees

- Developed in the computer game industry
- Describe the desired behavior in modules
- Behavior is composed of a sequence of sub-behaviors
- Used extensively at CMU[4] for robotic manipulation
- e.g. move green cube to goal location[5]

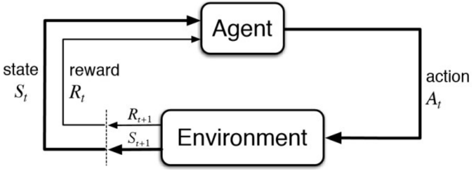




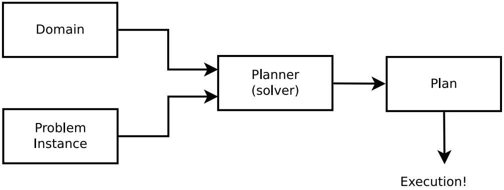
[4] Carnegie Mellon University
[5] https://btirai.github.io/youbot

Reinforcement learning

- Autonomously discover a behavior through trial and error [Kober et al. 2013]
- Formalization based on MDP framework
- A policy maps states to actions, (objective is to learn the policy)
- Deep reinforcement learning used in the context of machine learning and ANN (Alpha Go)
- Trade-off between exploration and exploitation

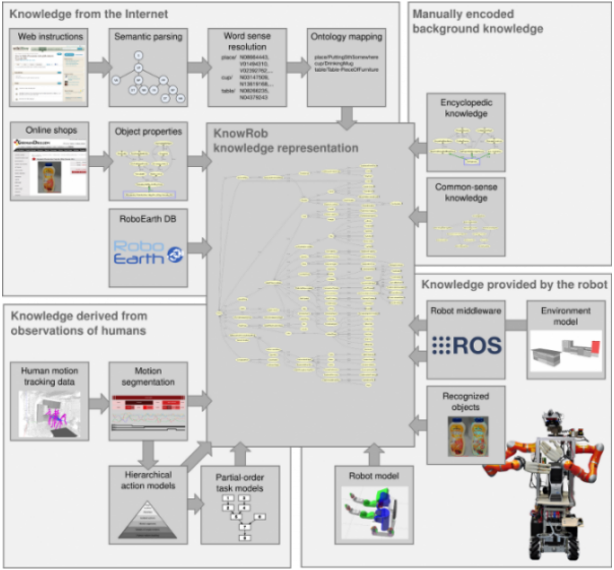


- AKA Automated Planning and Scheduling or AI planning or Activity Planning [6]
- A branch of AI that deals with decision making at a high level
- Solutions to planning problems based on **search**
- Classical planning does not involve ANN[7]
- Started in 1971 in Stanford Research Institute with Shakey robot and STRIPS planner



[6] https://en.wikipedia.org/wiki/Automated_planning_and_scheduling
[7] However some researchers are currently looking into it

Reasoning (KnowRob)



Symbols

$\neg$	not
$\wedge$	junction (and)
$\vee$	disjunction (or)
$\Rightarrow$	implication
$\iff$	equivalent (if and only if)

Syntax

- $a + b = c$
- $a\ b + = c$ *(Reverse Polish notation)*

Semantics

- $a + b = 4$ satisfies a **model** where a and b are 2.

Propositional logic (2)

1. If $p \in P_0$, then $p \in P$.
2. If $p \in P$, then $\neg p \in P$.
3. If $p \in P$ and $q \in P$, then $p \wedge q \in P$.
4. Nothing else is a propositional formula.

1. If $p \in P_0$, then $p \in P$.
2. If $p \in P$, then $\neg p \in P$.
3. If $p \in P$ and $q \in P$, then $p \wedge q \in P$.
4. Nothing else is a propositional formula.

Relationships

- logical connectives.
  example: "A cat is a mamal." and "A mamal is an animal". Therefore we can assume that "A cat is an animal"

Backus-Naur Form (BNF)

$$
\begin{array}{rcl}
\mathit{Sentence} & \Rightarrow & \mathit{AtomicSentence} | \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \Rightarrow & \mathit{True} | \mathit{False} | P | Q | R | \ldots \\
\mathit{ComplexSentence} & \Rightarrow & (\mathit{Sentence}) | [\mathit{Sentence}] \\
 & | & \neg \mathit{Sentence} \\
 & | & \mathit{Sentence} \wedge \mathit{Sentence} \\
 & | & \mathit{Sentence} \vee \mathit{Sentence} \\
 & | & \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
 & | & \mathit{Sentence} \iff \mathit{Sentence} \\
\text{Operator precedence} & : & \neg, \wedge, \vee, \Rightarrow, \iff
\end{array}
$$

BNF with operator precendences, from highest to lowest

Propositional logic
+ Predicates
+ Quantification

Predicates

- Instantiate variables
  example: "A cat is an animal" and "a dog is an animal" allows us to see dog and cat as instances of animals.

Quantifiers

- $\exists$ (exists) and $\forall$ (for all)

Formal languages and their ontological and epistemological commitments

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (Facts an agent believes)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Task Planning Formalization

Classical planning assumes[8]:

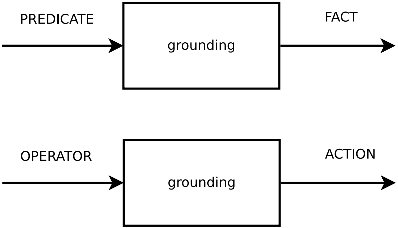
- Discrete representation (state space, plan space, task network) vs ~~Continuous~~
- Instantaneous actions vs ~~Temporal actions~~
- Sequential plans vs ~~Concurrent plans~~
- Fully observable vs ~~Partially observable~~
- Deterministic action outcome vs ~~Non deterministic (stochastic)~~
- Goal state condition vs ~~Temporal goals, Preferences, Rewards, etc.~~

NOTE: Classical planning is PSPACE-complete

[8] https://replay.csail.mit.edu/recordings/620

Grounding

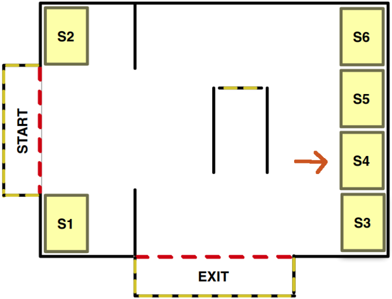
- In general it refers to the substitution of a variable[9] with an instance
- e.g. operator grounding, predicate grounding, etc.
- e.g. person? $\rightarrow$ Hans

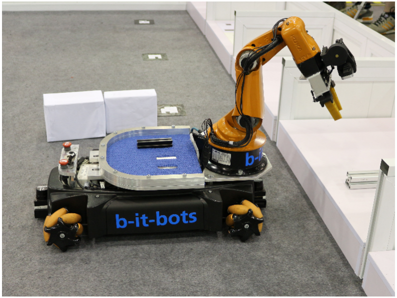


[9] or multiple variables

Predicate - Fact

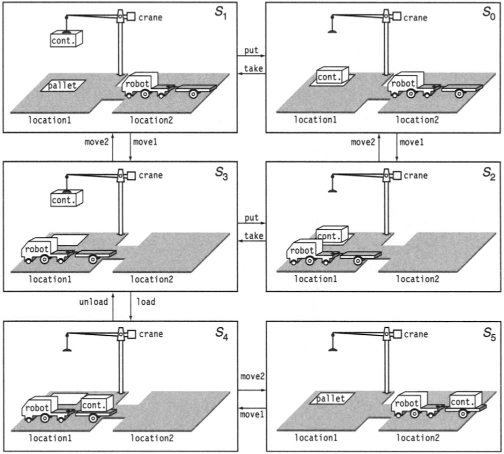
- Predicate: a vector of ungrounded literals, (placeholders for objects in the symbolic world)
- By fully grounding a predicate, we construct a fact
- e.g. robot_at(?robot ?location) $\rightarrow$ robot_at(youbot s4)





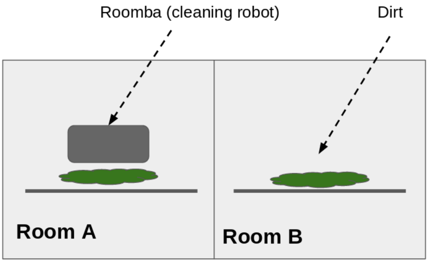
Knowledge base - state space representation

- State represented as a collection of facts
- From an implementation point of view those facts are stored in a knowledge base (KB)
- e.g. S1 = robot_at(loc1), holding(crane c1)



- Notation: π - plan, Π - set of plans
- (usually) a set of actions with some constraints[10]
- e.g. roomba (cleaning robot) domain

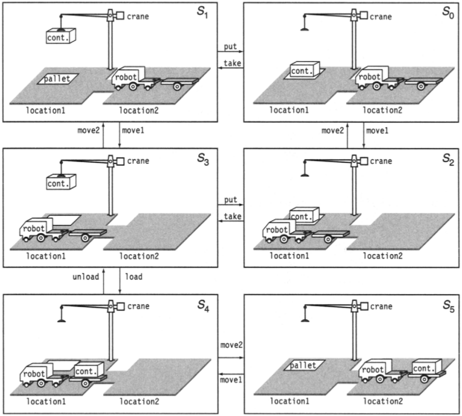
0: (CLEAN ROOMBA ROOMA)
1: (MOVE ROOMBA ROOMA ROOMB)
2: (CLEAN ROOMBA ROOMB)
3: (MOVE ROOMBA ROOMB ROOMA)



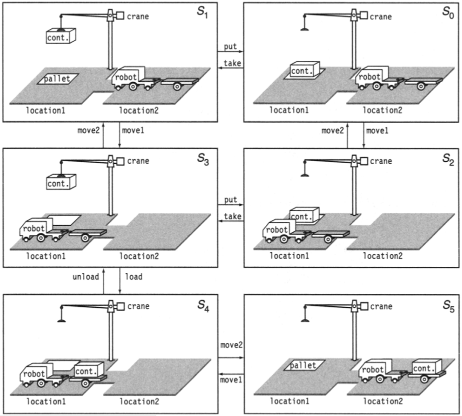
[10] **e.g. temporal constraints (deadlines)**; **ordering constraints**.

State, Initial state

- Notation: State $= s$, Initial state $= s_0$, Set of states $= S$
- A state is represented as a set of facts
- Relevant states: Initial state, goal state



- Notation: g
- A set of facts that we want to be true[11]
- The previous set can be used to build goal states



[11] Except for HTN, Hierarchical Task Networks. However, valid for state space and for plan space representations

- Notation: o operator, O set of operators
- A generic parameterized (ungrounded) action
- Has preconditions and effects
- e.g. move operator in PDDL[12] format for cleaning robot domain[13]

```
(:action move
  :parameters (?r - robot ?source ?destination - location)
  :precondition (at ?r ?source)
  :effect (and     ( not (at ?r ?source))
                   (at ?r ?destination))
)
```

[12] PDDL: Problem Domain Definition Language, will be covered later in the course...

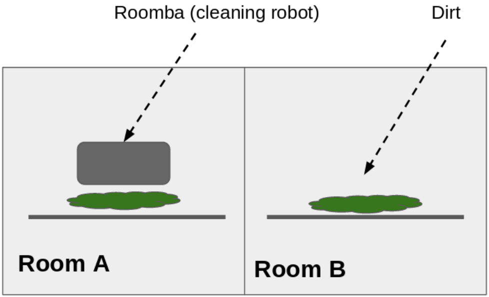
[13] https://github.com/oscar-lima/pddl_problems/blob/master/cleaning_robot/domain.pddl

- An action is a grounded[14] operator
- e.g.

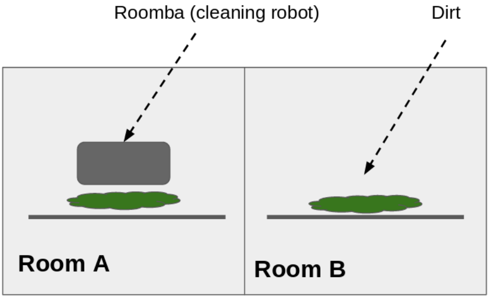
```
% operator (ungrounded action)
(:action move
        :parameters (?r - robot ?source ?destination - location)
        :precondition (at ?r ?source)
        :effect (and   ( not (at ?r ?source))
                        (at ?r ?destination))
)

% action (grounded operator)
(:action move
        :parameters (roomba roomA roomB)
        :precondition (at roomba roomA)
        :effect (and  ( not (at roomba roomA))
                           (at roomba roomB))
)
```

[14] or instantiated

- An action is applicable in state S_x iff all of its preconditions are met
- e.g.
- operators = move(l1? l2?), clean(?loc)
- actions = move(a b), move(b a), clean(a), clean(b)



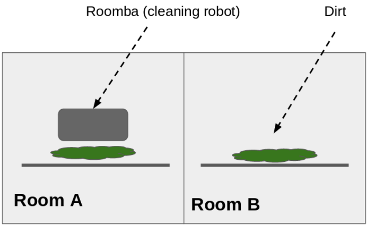
- An action is applicable in state S_x iff all of its preconditions are met
- e.g.
- operators = move(l1? l2?), clean(?loc)
- actions = move(a b), move(b a), clean(a), clean(b)
- applicable actions = move(a b), clean(a)



Preconditions

- Notation: $precond(o)$, $precond^{+}(o)$, $precond^{-}(o)$
- Positive preconditions: facts that must be true, present in KB, part of the state
- Negative preconditions: facts that must be false, not present in KB), not part of the state
- e.g. cleaning robot - clean operator

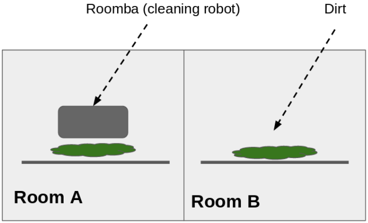
```
(:action clean
        :parameters (?r — robot ?l — location)
        :precondition (and  (at ?r ?l)
                            (not(clean ?l)))
        :effect (clean ?l)
)
```



Effects

- Notation: $effects(o)$, $effects^{+}(o)$, $effects^{-}(o)$
- Positive effects (AKA add list): facts that are added to KB
- Negative effects (AKA delete list): facts that are deleted from KB
- e.g. cleaning robot - clean operator

```
(:action clean
        :parameters (?r - robot ?l - location)
        :precondition (and (at ?r ?l)
          (not(clean ?l)))
        :effect (clean ?l)
)
```



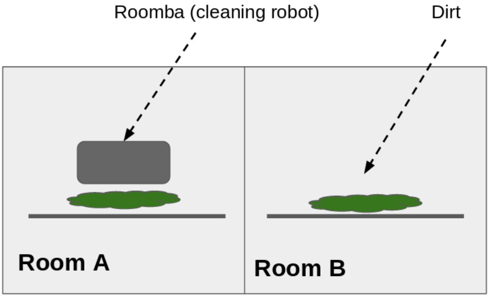
State transition system

- Notation: $\Sigma = (S, A, \gamma)$
- such that:
  - Σ - State transition system
  - S - Finite set of states, $S = \{s_0, s_1, s_2, ..., s_n\}$, NOTE: goal state(s) $\subset$ S
  - A - Finite set of actions, $A = \{a_1, a_2, a_3, ..., a_n\}$
  - γ - State transition function (AKA Progression), provides the state or set of states by applying "a" to "s" [15]
- γ - Key concept in planning and widely used in planning algorithms!
- e.g. 1. $\gamma(S_0, a_1) = S_1$
- e.g. 2. $\gamma(S_1, a_2) = \{S_1, S_2\}$

[15] Note: The produced transition is deterministic

Close world assumption

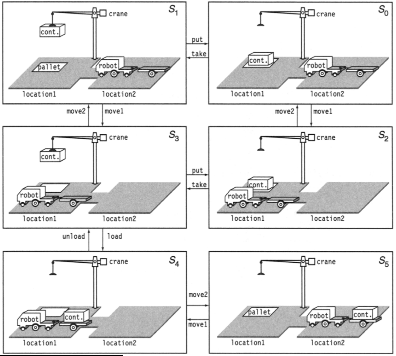
- All facts not present in KB[16] are assumed to be false
- e.g.
  - available_locations={room_a, room_b}
  - KB = {robot_at(roomba room_a)}
  - ¬ robot_at(roomba room_b) is assumed and hence does not belong to KB



[16] Knowledge Base, a set of facts that represents the state of the world

Exercise I, (1/2)

- Provided a Dock Worker Robot domain[17] and instance, use the classical planning formalism to:
  - 1. Formally describe any state of your preference, e.g. S_3
  - e.g.: $S_1 = \{adjacent(loc1\ loc2), attached(pile1\ loc1), belong(k1\ loc1), \ldots\}$
  - 2. Derive **some** of the preconditions and effects for any action of your choice
  - e.g.: $\gamma(S_3, put(k1, c1, pile1)) \rightarrow S_2$, "put" has precondition "$holding(k1\ c1)$"
- NOTE: Make use of the close world assumption



[17] see next slide or click : https://github.com/oscar-lima/pddl_problems/blob/master/dock_worker_robots/domain.pddl

Exercise II, (2/2) - DWR Domain description

- Types = { robot, location, crane, pile, container }
- Objects = { R1, loc1, loc2, k1, pile1, c2 }
- Actions = { move, load, unload, take, put }
- Predicates:

```
(adjacent ?l1  ?l2 - location)          ; location ?l1 is adjacent to ?l2
(attached ?p - pile ?l - location)      ; pile ?p attached to location ?l
(belong ?k - crane ?l - location)       ; crane ?k belongs to location ?l

(at ?r - robot ?l - location)           ; robot ?r is at location ?l
(occupied ?l - location)                ; there is a robot at location ?l
(loaded ?r - robot ?c - container )     ; robot ?r is loaded with container ?c
(unloaded ?r - robot)                   ; robot ?r is empty

(holding ?k - crane ?c - container)     ; crane ?k is holding a container ?c
(empty ?k - crane)                      ; crane ?k is empty

(in ?c - container ?p - pile)           ; container ?c is within pile ?p
(top ?c - container ?p - pile)          ; container ?c is on top of pile ?p
(on ?c1 - container ?c2 - container)    ; container ?c1 is on top of container ?c2
```

Task Planning Representations

STRIPS[18]

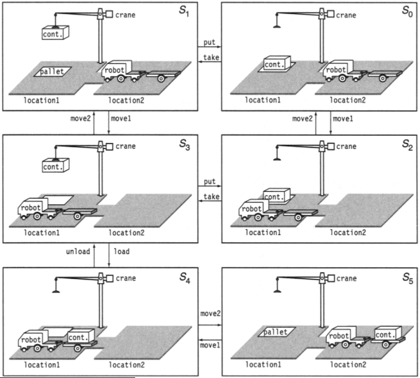
- There is complete knowledge about the initial state
- Actions are deterministic with exactly one outcome
- The solution is a linear plan (a sequence of actions)
- No exogenous events
- Search offline, then execute with eyes closed
- $STRIPS = (P, O, I, G)$
  - P - set of conditions
  - O - set of operators
  - I - initial state (s_0)
  - G - goal state
- Syntax e.g.

$Action(Move(robot, from, to),$
$PRECOND : At(robot, from),$
$EFFECT : \neg At(robot, source) \wedge At(robot, destination))$

[18] Stanford Research Institute Problem Solver

State Space search[19]

- Idea: Apply standard search algorithms (e.g. BFS, DFS, A*) to planning problem
- Nodes correspond to world states
- Arcs correspond to state transitions
- Path in the search space corresponds to plan



[19]Slide partially based on AIPLAN Edinburgh planning course

Forward Search pseudocode[20]

```
Forward-search(O, s0, g)
    s ← s0
    π ← the empty plan
    loop
        if s satisfies g then return π
        applicable ← {a | a is a ground instance of an operator in O,
                          and precond(a) is true in s}
        if applicable = ∅ then return failure
        nondeterministically choose an action a ∈ applicable
        s ← γ(s, a)
        π ← π.a
```

[20] Chapter 4, p 70 Dana Nau et al, Automated Planning: Theory & practice book

General idea:

- Start (backwards) from the goal state, $S \leftarrow g$
- Apply inverse of the planning operator (regression) to produce subgoals: $\gamma^{-}1(g, a)$
- Termination condition: $s_0 \subset S$, (goal is s_0)

Relevance:

- An action $a \in A$ is **relevant** iff:
  - $g \cap \mathit{effects}(a) \neq \{\}$
  - $g^{+} \cap \mathit{effects}^{-}(a) = \{\}$
  - $g^{-} \cap \mathit{effects}^{+}(a) = \{\}$

Regression:

- $\gamma^{-}1(g, a) = (g - \mathit{effects}(a) \cup \mathit{precond}(a))$

Backward Search pseudocode[21]

```
Backward-search(O, s_0, g)
    π ← the empty plan
    loop
        if s_0 satisfies g then return π
        relevant ← {a | a is a ground instance of an operator in O
                        that is relevant for g}
        if relevant = ∅ then return failure
        nondeterministically choose an action a ∈ applicable
        π ← a.π
        g ← γ^{-1}(g, a)
```

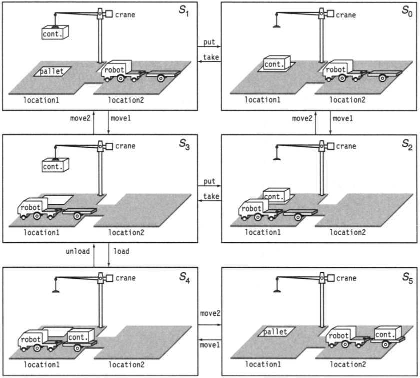
[21] Chapter 4, p 73 Dana Nau et al, Automated Planning: Theory & practice book

Assume:

- $g = S_4$, $s_0 = s_0$

Provide:

- The set of applicable actions in s_0 (Fwd search)
- The set of relevant actions in g
- Output of the regression function for g (choose one action randomly from previous step)



PDDL - Planning Domain Definition Language

- Do not write C++ code, just describe your domain!
- Standardize planners input → benchmark
- Created in 1998 by Drew McDermott and colleagues
- Inspired by STRIPS and ADL (Action Description Language)
- Used in International Planning Competition (IPC) 1998 / 2000

Syntax - Domain[22] (1/2)

```
(define (domain cleaning_robot)

(:requirements
        :typing
)

(:types
        location
        robot
)

(:predicates
        (at ?r - robot ?l - location)    ; robot r? is at location l?
        (clean ?l - location)            ; location ?l is clean
)
```

[22] Source code: https://github.com/oscar-lima/pddl_problems/tree/master/cleaning_robot

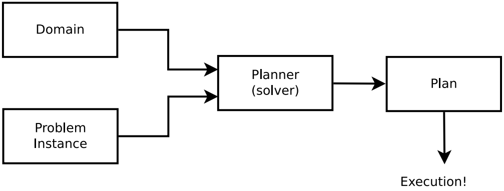
```
(:action move
        :parameters (?r - robot ?source ?destination - location)
        :precondition (at ?r ?source)
        :effect (and    ( not (at ?r ?source))
        (at ?r ?destination))
)

(:action clean
        :parameters (?r - robot ?l - location)
        :precondition (and (at ?r ?l) (not(clean ?l)))
        :effect (clean ?l))
)
```

Syntax - Problem

```
(define (problem p01)

(:domain cleaning_robot)

(:objects
        ghost - robot
        locA locB - location
)

(:init
        (at ghost locA)
        (not(clean locA))
        (not(clean locB))
)

(:goal
        (   and (at ghost locA)
                    (clean locB)
                    (clean locA)
        )
)
)
```

Elements

- domain.pddl
- problem.pddl
- try it! :
- http://editor.planning.domains



Examples

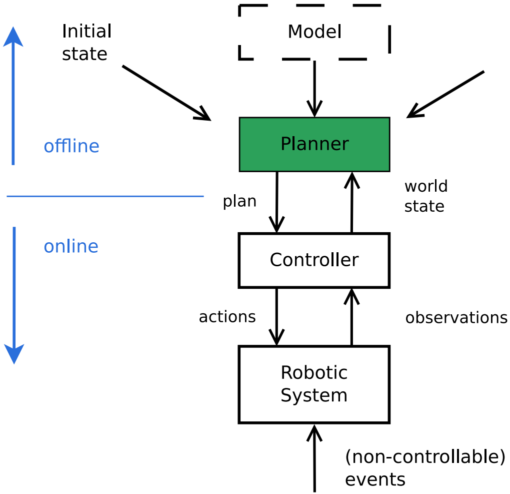
- Multiple examples available under:
- https://github.com/oscar-lima/pddl_problems

Evolution

- PDDL 1.2 : Classical planning
- PDDL 2.1 : Temporal planning
- PDDL + : Hybrid planning

Plan execution

Classic execution architecture



Scheduling

- Dispatch actions
- Encode plan as an STN
- e.g. Esterel dispatch (ROSPlan)

Environment representation

- Sense environment
- Transform into symbols
- Maintain KB (useful for re-planning)

ROSPlan

- Execution framework for Robotics
- Based on Robot Operating System (ROS)
- Maintained by KCL University
- Open source:
- https://github.com/kcl-planning/ROSPlan

Next lecture contents

- Plan Space representation
- Task networks (HTN)
- Planners
- Heuristics
- and more ...

Thank You for Your Attention.

Do You Have Questions?

[Gha+04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[CO17] Michele Colledanchise and Petter Ogren. “Behavior trees in robotics and AI: an introduction”. In: *arXiv preprint arXiv:1709.00084* (2017).

[Kob+13] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.