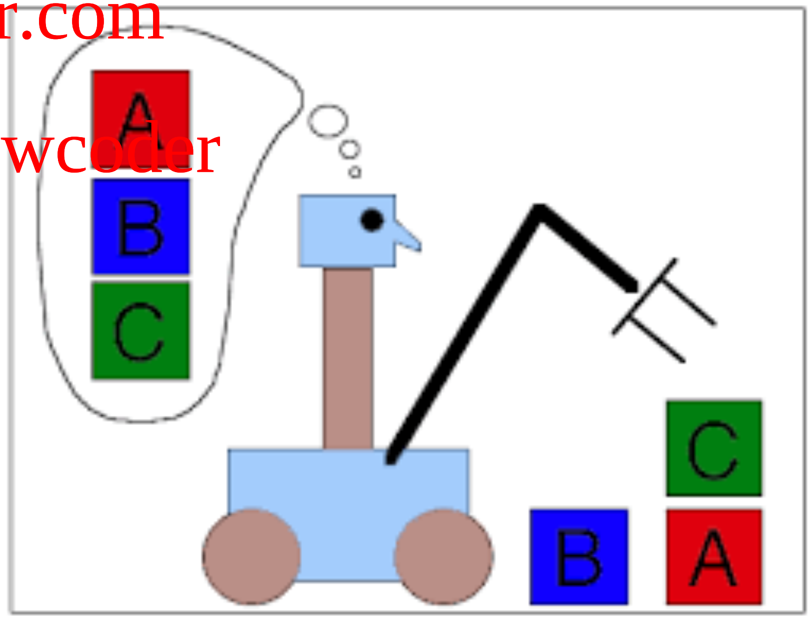
Introducing AI Planning

- Informal definition
- Formalising a planning problem
- Different types of planning
- Reasoning about events
- Abductive planning
- Examples



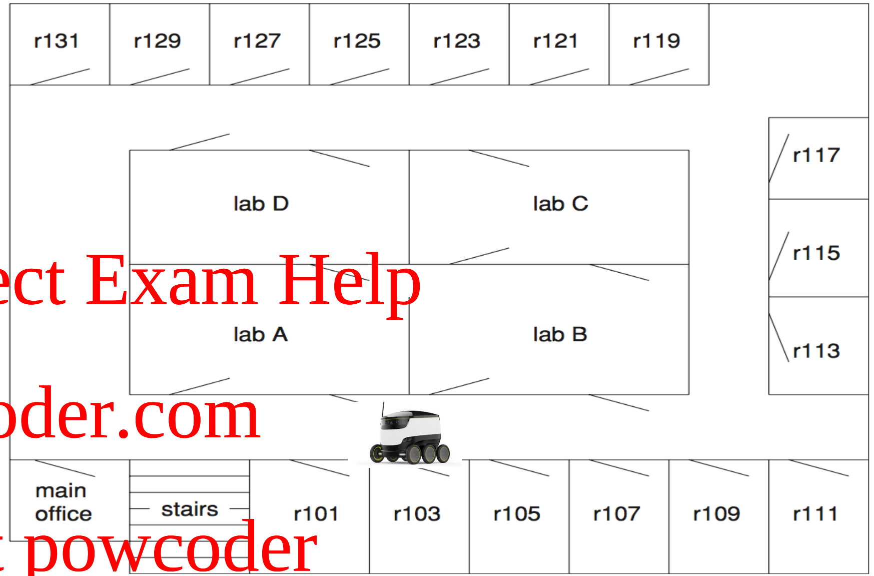
What is planning?

Planning is about how an agent achieves its goals.

Find a sequence of actions to solve a goal.

To reason about what to do, an agent must have:

- goals,
- model of the world,
- model of consequences of actions.

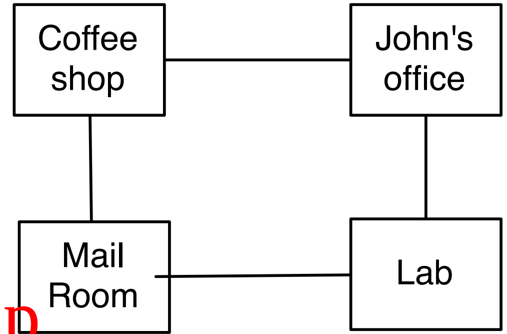


Delivery robot

Example: delivery robot

State features

rhc, rhm, mw, swc, cs, off, mr, lab



Actions

move clockwise (**mc**), deliver mail (**dm**),
pick up a coffee (**puc**), deliver the coffee (**dc**),
pick up mail (**pum**), move anticlockwise (**mcc**),

Action specification

Actions	*Precond*	*Postcond*
puc	not rhc $\wedge$ cs	rhc’
dc	rhc $\wedge$ off	not swc’
pum	wm $\wedge$ mr	rhm’
dm	rhm $\wedge$ off	

Explicit state space representation is a set of triples <s, a, s’>

But too many states and representation not very flexible

Feature-centric representation

For each action

precondition specifies when the action can be carried out.

For each feature:

- causal rules that specify when the feature gets a new value

e.g., $rhc' \leftarrow puc$

- frame rules that specify when the feature keeps its value.

e.g.

$cs' \leftarrow cs \wedge \text{not } mcc \wedge \text{not } mc$

$rhc' \leftarrow rhc \wedge \text{not } dc$

Action-centric representation

For each action

- *Preconditions* – features that must be true for an action to occur
- *Effects* – values of feature that change as result of the actions

STRIPS representation

- Underlying assumption:
  - Primitive features not mentioned in action descriptions are assumed to be unchanged by the action performance.
  - Derived features are defined, using definite clauses, in terms of primitive features in a given state.

Example:

action **puc**	action **dc**
Precondition: [cs, not rhc] Effect: [rhc]	Precondition: [off, rhc] Effect: [not rhc, not swc]

Planning with certainty

Main assumptions

- Deterministic actions.
  - agent capable of predicting the consequences of its actions.
- No exogenous events that change the state of the world.
- World state fully observable
- Time progresses discretely from one state to the next.
- Goals are predicates about a state that must be achieved or maintained.

Forward Planning

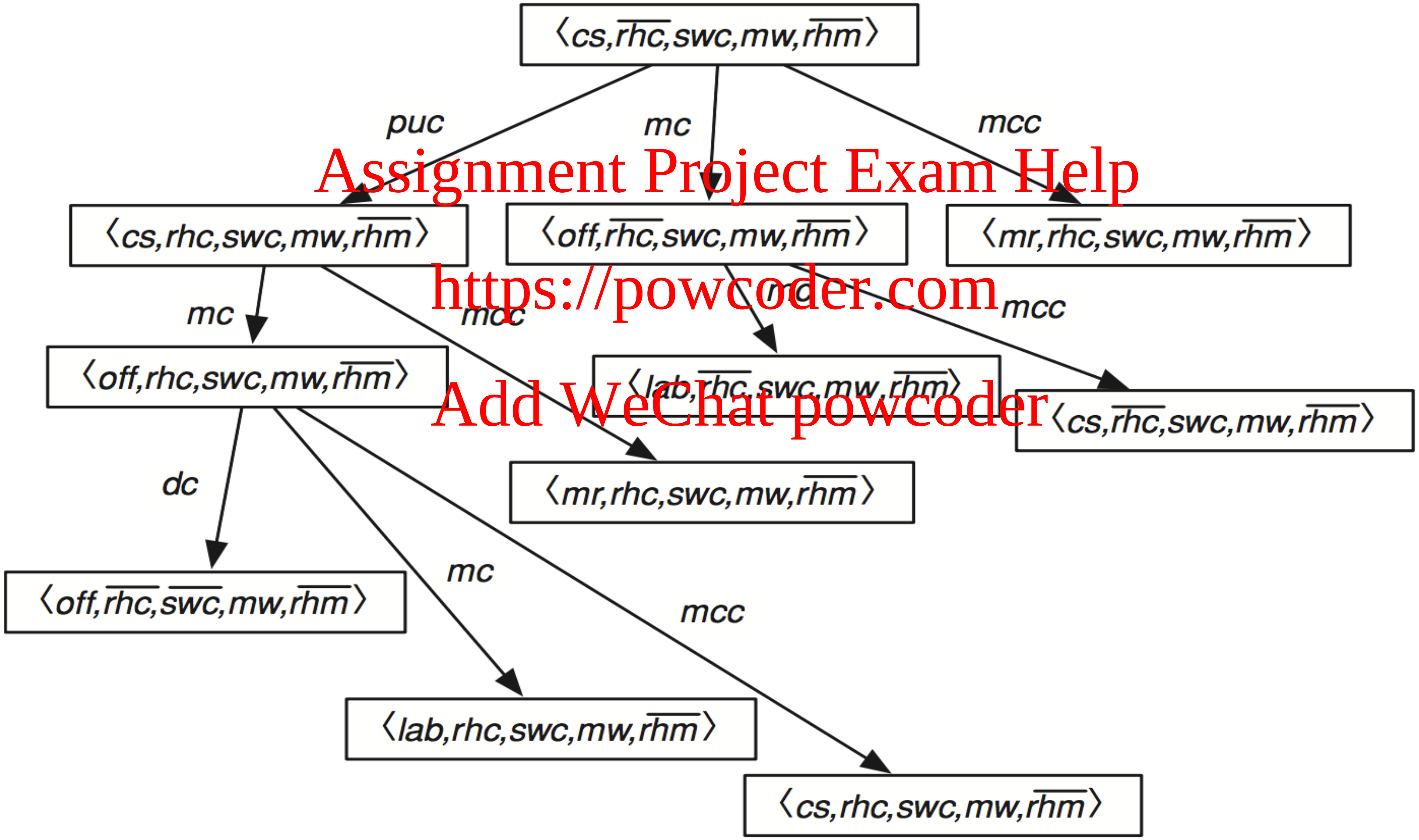
A deterministic planner that behaves as a problem solver for computing deterministic plans.

A deterministic plan is a sequence of actions that can achieve a **goal** from a given starting state.

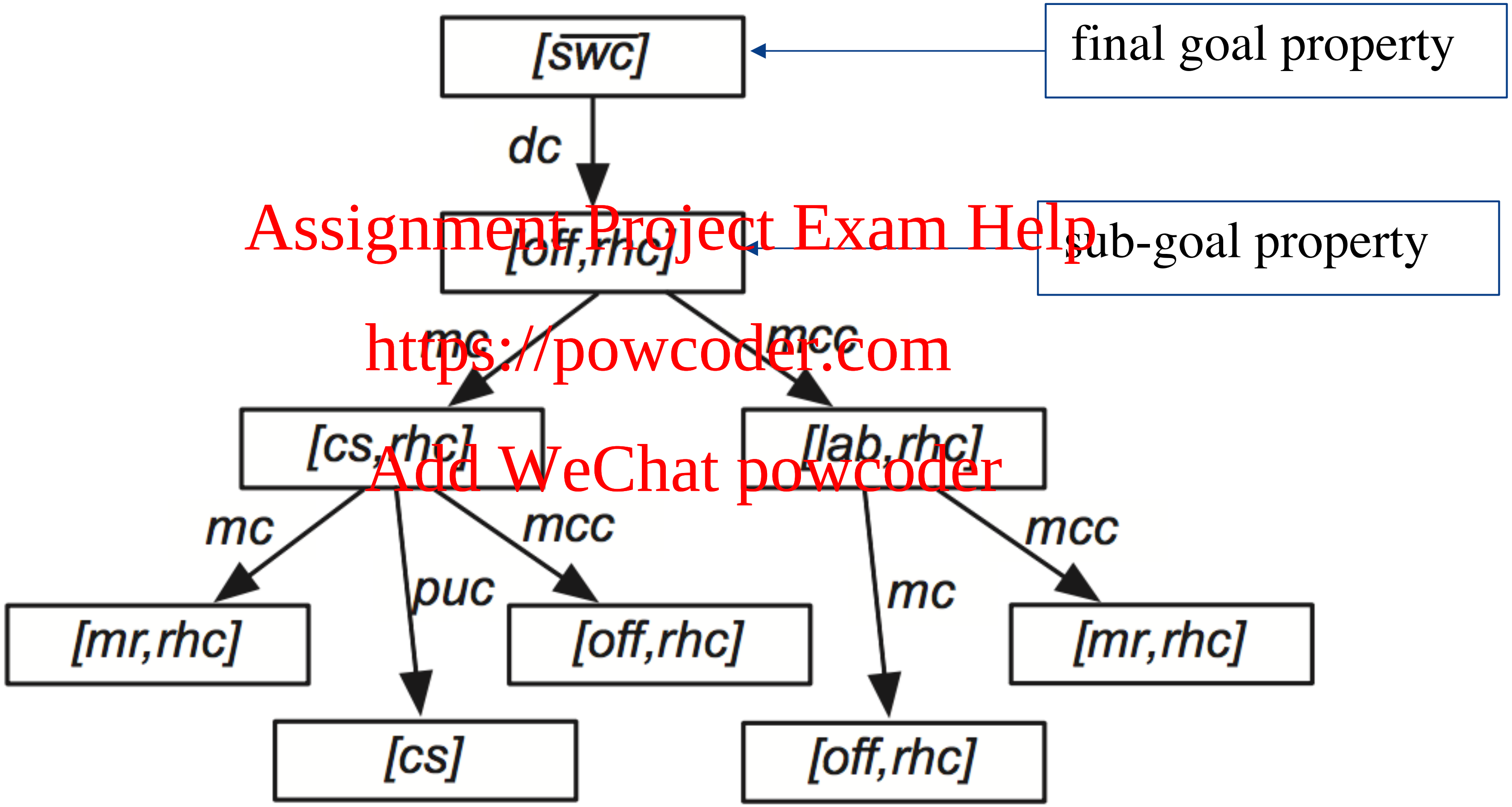
A forward planner treats the planning problem as a path finder in a state-space graph, searching the graph from the initial state to a state that satisfies a goal description.

A state-space graph is a graph whose nodes are states, and arcs are actions from one state to the other.

Example


$\langle cs, \overline{rhc}, swc, mw, \overline{rhm} \rangle$
puc
mc
mcc
$\langle cs, rhc, swc, mw, \overline{rhm} \rangle$
$\langle off, \overline{rhc}, swc, mw, \overline{rhm} \rangle$
$\langle mr, \overline{rhc}, swc, mw, \overline{rhm} \rangle$
mc
mcc
mc
mcc
$\langle off, rhc, swc, mw, \overline{rhm} \rangle$
$\langle lab, \overline{rhc}, swc, mw, \overline{rhm} \rangle$
$\langle cs, \overline{rhc}, swc, mw, \overline{rhm} \rangle$
$\langle mr, rhc, swc, mw, \overline{rhm} \rangle$
dc
mc
mcc
$\langle off, \overline{rhc}, \overline{swc}, mw, \overline{rhm} \rangle$
$\langle lab, rhc, swc, mw, \overline{rhm} \rangle$
$\langle cs, rhc, swc, mw, \overline{rhm} \rangle$

Regression Planning



Language for planning

Example of action schema:

Action(Fly(P, From, To),
PRECOND: At(P, From) $\wedge$ Plane(P) $\wedge$ Airport(From) $\wedge$ Airport(To)
EFFECT: ¬At(P, From) $\wedge$ At(P, To))

Action applicable in a state s:

There exists a substitution θ, such that s $\wedge$ PRECONDθ is satisfiable.

Result of action is a state s':

$$s' = s \setminus \{\varphi \mid \text{not } \varphi \in \text{EFFECT}\theta\} \cup \{\phi \mid \phi \in \text{EFFECT}\theta\}$$

Most common standardised language for planning is PDDL (Planning Domain Definition Language)

International Planning Competition
(http://www. icaps.conference.org/index.php/Main/Competition)

Reasoning about (effects of) events

Event Calculus

Logical framework for representing and reasoning about events and effects of events over time.

Ontology consists of events, fluents and time.

Signature includes the following sorted predicates:

Predicates	Meaning
initiates(e, f, t)	Fluent f starts to hold after event e at time t
terminates(e, f, t)	Fluent f stops to hold after event e at time t
initially(f)	Fluent f holds at the beginning (starting time 0)
happens(e,t)	Event e happens at time t
holdsAt(f, t)	Fluent f holds at time t
clipped(t1, f, t2)	Fluent f is terminated between times t1 and t2

Event Calculus Axiomatisation

Every Event Calculus description includes two theories:

- domain independent theory

  describes general principles for deciding when fluents hold or do not hold at particular time-points

- domain dependent theory

  describe particular effects of events or actions using the predicates initiates(.) and terminates(.). It may also include sentences stating the full initial state (using the predicate initially(.), and a narrative of instances of events occur at particular time-points, using predicate happens(.).

Domain-independent theory

Three general (commonsense) principles:

(i) fluents that were *initially tr*ue continue to hold until events occur that terminate them

$$\text{holdsAt}(f, t2) \leftarrow \text{initially}(f), \text{not clipped}(0, f, t2)$$

(ii) fluents that have been *initiated by event occurrences* continue to hold until events occur that terminate them

$$\text{holdsAt}(f,t2) \leftarrow \text{initiates}(a,f,t1), \text{happens}(a,t1), t1 < t2, \text{not clipped}(t1,f,t2)$$

(iii) fluents only change status via occurrences of terminating events

$$\text{clipped}(t1,f,t) \leftarrow \text{happens}(a, t2), \text{terminates}(e, f, t2), t1 < t2, t2 < t$$

Domain-dependent theory

Defines instead effects of actions and a given initial state.

It depends on the particular problem that is formalised.

Includes rules defining specific cases of:

initiates(e, f, t) ← BODY

terminates(e, f, t) ← BODY

initially(f)

The BODY conditions can include literals about "holdsAt" and "happens" as well as any other "static" domain specific predicate.

Integrity constraints could also be expressed to capture constraints on the occurrence of events. These are normally considered in addition to the domain-dependent axiomatisation.

Event Calculus Abductive Planning

Given:

DI domain independent EC theory

DD domain dependent EC theory

IC integrity constraints

G goal state

An EC plan P is a tuple <As, TC> where

As = {happens(e_i, t_i), happens(e_j, t_j),…}

TC = {ti < tj,…}, set of temporal constraints

such that

- $\overbrace{DI \cup DD}^{KB} \cup \overbrace{P}^{\Delta} \models G$
- $DI \cup DD \cup P \cup IC$ is consistent

- $KB \cup \Delta \models G$
- $KB \cup \Delta \cup IC \not\models \bot$

A small planning example

Consider a simple shopping trip planning, where an agent has to go to different places to buy different items.

We formulate the problem as an Event Calculus Abductive Planning problem:

DI (domain independent theory)

```
holds(F, T) :- time(T), initially(F), not clipped(0, F, T).

holds(F, T) :- time(T1), time(T),  0 < T1, T1 < T,
               happens(A, T1),  initiates(A, F, T1),
               \+ clipped(T1, F, T).

clipped(T1, F, T) :- time(T1), time(T), time(T2),
                     T1 < T2, T2 < T,
                     happens(A, T2), terminates(A, F, T2).
```

A small planning example

DD (domain dependent theory)

```
   initiates(goto(X), at(X), T) :- place(X).
terminates(goto(X), at(Y), T) :- place(X),
                                  place(Y),
                                  X =/= Y.
initiates(buy(Item, Place), have(Item), T) :-
                                  sells(Place, Item),
                                  item(Item),
                                  holds(at(Place), T).
time(T) :- T in 0..8.
```

fluents

- at(Place)
- have(Item)

actions

- goto(Place)
- buy(Item, Place)

DD (initial state and static facts)

```
initially(at(home)).   sells(sm, milk).     place(sm).     item(drill).
                       sells(sm, banana).   place(hws).    item(milk).
                       sells(hws, drill).   place(home).   item(banana).
```

A small planning example

IC (integrity constraints)

ic :- happens(E1, T), happens(E2, T), E1 =/= E2.

A (abducible)

abducible(happens(_,_)).

Instance of Event Calculus Abductive Planning

$<DI \cup DD, A, IC>$

Goal:
holds(have(banana), 5) $\wedge$
holds(have(drill), 5)

Plan:

happens(goto(hws), 1),
happens(buy(drill, hws), 2),
happens(goto(sm), 3),
happens(buy(banana, sm), 4).

EC Abductive Planning with unground goals

Unground Goals

Compute plans that take the agent to a state that satisfies the given goal.

General EC Abductive Proof Procedure

Observations are conjunction of unground literals. Abductive solutions are tuples (As, Cs, Ns), where

As: conjunction of (skolemised) abducibled

Cs: set of arithmetic constraints over time point variables

Ns: set of dynamic constraints (dynamic denials).

Forcing grounding of finite domain variables in solutions:

```
| ?- use_module(library(terms)).
| ?- use_module(library(clpfd)).
```

Abductive procedure grounds variables asap during inference:

```
| ?- enforce labeling(true)
```

Conclusion

- Notion of planning
- Forward planning
- Regression planning
- Event Calculus
- Event Calculus Abductive Planning