Planning

Shakey
Developed at SRI International
1966-1972
Retired at the Computer Science Museum

Review

- What is classical planning?
  - Environment: deterministic, static and fully observable
  - Factored representation of states: (lights on/off, door open/closed, window open/closed). With k variables, we have 2^k states.
- The frame problem
  - The frame problem is the challenge of representing the effects of an action without having to represent explicitly a large number of intuitively obvious non-effects.

This Week

Today:

- How do we represent a planning problem?
  - Planning Domain Definition Language (PDDL)

Wednesday:

- How do we solve a planning problem represented in PDDL?
  - State Space Search
    - Progression Search
    - Regression Search
    - Heuristics

From STRIPS to PDDL

STRIPS (**St**anford **R**esearch **I**nstitute **P**roblem **S**olver) was initially an automated planner developed for Shakey at SRI in 1971.

The same name was later used to refer to the formal language used for the inputs to this planner.

This language is the base for most of the languages used in planning today.

PDDL - Planning Domain Definition Language, developed in 1998 is derived from the STRIPS planning language.

PDDL

To formulate a planning problem, we need to define:

- Initial state
- Actions available
- Results of the actions
- Goal state

PDDL – State Representation

A state is represented with a conjunction of literals representing atomic facts (fluents):

- Logical Propositions: Cold $\wedge$ Wet
- FOL literals: At(Plane1, SFO) $\wedge$ At(Plane2, SJC)

The literals must be ground (no variables):

- At(x, SFO) is not allowed

The literals must be function-free:

- At(Brother(Anna), SFO) is not allowed

Names are unique:

- At(Plane1, SFO) $\wedge$ At(Plane2, SJC): Plane1 and Plane2 are distinct

PDDL – State Representation

Closed World Assumption:

- Whatever is not explicitly stated is assumed to be false.
- In PDDL (but not in STRIPS) negative literals are allowed. However with the closed world assumption, they are not needed.

PDDL – Action Representation

Actions (STRIPS operators) are described using action schemas.

An action schema includes:

- Action name followed by a list of variables
- Precondition
- Effect

Example:

Action(Fly(p, from, to),

PRECOND: At(p, from), Plane(p), Airport(from), Airport(to)

EFFECT: ¬At(p, from), At(p, to))

PDDL – Action Representation

Action(Fly(p, from, to),

PRECOND: At(p, from), Plane(p), Airport(from), Airport(to)

EFFECT: ¬At(p, from), At(p, to))

Each schema represents a set of actions.

We can think of the variables *p*, *from* and *to* as universally quantified.

We can substitute values for the variables and get one ground action: {p/Plane1, from/SFO, to/JFK}:

Action(Fly(Plane1, SFO, JFK),

PRECOND: At(Plane1, SFO), Plane(Plane1), Airport(SFO), Airport(JFK)

EFFECT: ¬At(Plane1, SFO), At(Plane1, JFK))

PDDL – Action Representation

Action(Fly(p, from, to),

PRECOND: At(p, from), Plane(p), Airport(from), Airport(to)

EFFECT: ¬At(p, from), At(p, to))

Any variable in the effect of an action schema must appear in the action's preconditions.

There is no explicit reference to time: the precondition implicitly refers to time t and the effect to time t+1.

PDDL and the Frame Problem

Classical planning deals with problems where most actions leave most things unchanged.

The frame problem is the challenge of representing the effects of an action without having to explicitly represent non-effects.

PDDL addresses the frame problem by specifying the result of an action in terms of what changes. Anything that stays the same is not mentioned.

PDDL – Applicable Actions

An action a is applicable in state s if a's preconditions are satisfied by s.

When an action schema contains variables, it may have multiple applicable instantiations.

Let's say we have the following initial state and action schema:

Init(At(P1, SFO) $\wedge$ At(P2, JFK) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(SFO) $\wedge$
Airport(JFK))

Action(Fly(p, from, to),
PRECOND: At(p, from), Plane(p), Airport(from), Airport(to)
EFFECT: ¬At(p, from), At(p, to))

PDDL – Applicable Actions

An action a is applicable in state s if a's preconditions are satisfied by s.

Init(**At(P1, SFO)** ∧ At(P2, JFK) ∧ **Plane(P1)** ∧ Plane(P2) ∧ **Airport(SFO)** ∧ **Airport(JFK)**)

Action(Fly(p, from, to),

PRECOND: **At(p, from), Plane(p), Airport(from), Airport(to)**

EFFECT: ¬At(p, from), At(p, to))

The Fly action can be instantiated as:

Action(Fly(P1, SFO, JFK),

PRECOND: At(P1, SFO), Plane(P1), Airport(SFO), Airport(JFK)

EFFECT: ¬At(P1, SFO), At(P1, JFK))

PDDL – Applicable Actions

Init(At(P1, SFO) ∧ **At(P2, JFK)** ∧ Plane(P1) ∧ **Plane(P2)** ∧ **Airport(SFO)** ∧ **Airport(JFK)**)

Action(Fly(p, from, to),

PRECOND: **At(p, from), Plane(p), Airport(from), Airport(to)**

EFFECT: ¬At(p, from), At(p, to))

The Fly action can also be instantiated as:

Action(Fly(P2, JFK, SFO),

PRECOND: At(P2, JFK), Plane(P2), Airport(JFK), Airport(SFO)

EFFECT: ¬At(P1, JFK), At(P1, SFO))

Both of these ground actions are applicable in the initial state since their preconditions are satisfied.

PDDL – Applying the Action

Starting from a state s, to apply action a:

- Remove the negative literals in the action's effect. This is the delete list Del(a)
- Add the positive literals in the action's effects. This is the add list or Add(a))
- Result(s,a)=(s−Del(a))∪Add(a)

s: At(P1, SFO) ∧ At(P2, JFK) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(SFO) ∧ Airport(JFK))

a: Action(Fly(P1, SFO, JFK),

PRECOND: At(P1, SFO), Plane(P1), Airport(SFO), Airport(JFK)

EFFECT: ¬At(P1, SFO), At(P1, JFK))

Delete list Del(a): At(P1, SFO)

Add list Add(a): At(P1, JFK)

Result(s, a): At(P2, JFK) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(SFO) ∧ Airport(JFK)) ∧ At(P1, JFK)

PDDL – Goal Representation

A goal is a conjunction of literals, positive or negative that may contain variables.

Variables are treated as existentially quantified.

A state is a goal state if it entails the goal.

Goal(At(x, SFO) $\wedge$ Plane(x))

Our goal here is to have at least one plane at SFO.

s:

At(Plane1, SFO) $\wedge$ Plane(Plane1) $\wedge$ At(Plane2, JFK) $\wedge$Plane(Plane2)

The state s entails the goal. It is a goal state.

PDDL – Problem Representation

A set of action schemas defines a planning domain.

A specific problem within the domain is defined with the addition of an initial state and a goal.

The problem is solved when we find a sequence of actions from the initial state to a state that entails the goal.

PDDL – Cargo Example

Planning Domain:

Action(Load(c, p, a),

PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

EFFECT: ¬ At(c, a) ∧ In(c, p))

Action(Unload(c, p, a),

PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

EFFECT: At(c, a) ∧ ¬ In(c, p))

Action(Fly(p, from, to),

PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)

EFFECT: ¬ At(p, from) ∧ At(p, to))

PDDL – Cargo Example

A specific problem = initial state + goal + planning domain

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1) ∧
Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))

Goal(At(C1, JFK) ∧ At(C2, SFO))

Action(Load(c, p, a),

…)

Action(Unload(c, p, a),

…)

Action(Fly(p, from, to),

…)

PDDL – Cargo Example

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧
Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))

Goal(At(C1, JFK) ∧ At(C2, SFO))

Action(Load(c, p, a),

PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

EFFECT: ¬ At(c, a) ∧ In(c, p))

Action(Unload(c, p, a),

PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

EFFECT: At(c, a) ∧ ¬ In(c, p))

Action(Fly(p, from, to),

PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)

EFFECT: ¬ At(p, from) ∧ At(p, to))

PDDL – Limitations

Less expressive than first order logic.

No way to say move all cargo to SFO (no universal quantifier in goal).

What about Fly(P1, SFO, SFO)? We get At(P1, SFO) and ¬ At(P1, SFO)

No way to concisely express global constraints: no more than 100 planes at a given airport.

No explicit time representation: Cargo must be at JFK by 5PM.

PDDL – Monkey and Bananas Example

A monkey is in a room with bananas hanging from the ceiling.

A box is in the room and the monkey can climb on it to reach the bananas.

Initially, the monkey is at A, the bananas at B and the box at C.

Initially the monkey and the box are at height Low and the bananas are at height High. If the monkey climbs onto the box, he will be at height High.

B

A

C

PDDL – Monkey and Bananas Example

Initial State:

Init(At(Monkey,A) ∧

At(Bananas,B) ∧

At(Box,C) ∧

Height(Monkey,Low) ∧

Height(Box,Low) ∧

Height(Bananas,High) ∧

Pushable(Box) ∧

Climbable(Box))

B

A

C

PDDL – Monkey and Bananas Example

Action(ACTION:Go(from, to),

PRECOND:At(Monkey, from),

∧ Height(Monkey,Low))

EFFECT:At(Monkey, to) ∧

¬(At(Monkey, from)))

B

A

C

PDDL – Monkey and Bananas Example

Action(ACTION:Push(b, from, to),

PRECOND:At(Monkey, from) ∧

Height(Monkey,Low)) ∧

At(b, from) ∧ Pushable(b))

EFFECT:At(b, to) ∧

At(Monkey, to) ∧

¬At(b, from) ∧

¬At(Monkey, from))

B

A

C

PDDL – Monkey and Bananas Example

Action(ACTION:ClimbUp(b),

PRECOND:At(Monkey, x) ∧

At(b, x) ∧ Climbable(b) ∧

∧ Height(Monkey,Low))

EFFECT:On(Monkey, b) ∧

¬ Height(Monkey,Low) ∧

Height(Monkey,High))

B

A

C

PDDL – Monkey and Bananas Example

Action(ACTION:ClimbDown(b),

PRECOND:On(Monkey, b) ∧

Height(Monkey,High),

EFFECT:¬On(Monkey, b) ∧

¬Height(Monkey,High)

∧ Height(Monkey,Low))

B

A

C

PDDL – Monkey and Bananas Example

Action(ACTION:Grasp(b),

PRECOND:Height(Monkey, h) ∧

Height(b, h) ∧

At(Monkey, x) ∧ At(b, x)

- EFFECT:Have(Monkey, b))

B

A

C

PDDL – Monkey and Bananas Example

Action(ACTION:UnGrasp(b),

PRECOND:Have(Monkey, b),

EFFECT:¬Have(Monkey, b))

B

A

C

PDDL – Monkey and Bananas Example

Goal?

Goal(Have(Monkey, Bananas))

B

A

C

PDDL – Monkey and Bananas Example

Solution?

[Go(Monkey, A, C)),

Push(Box, C, B),

ClimbUp(Box)

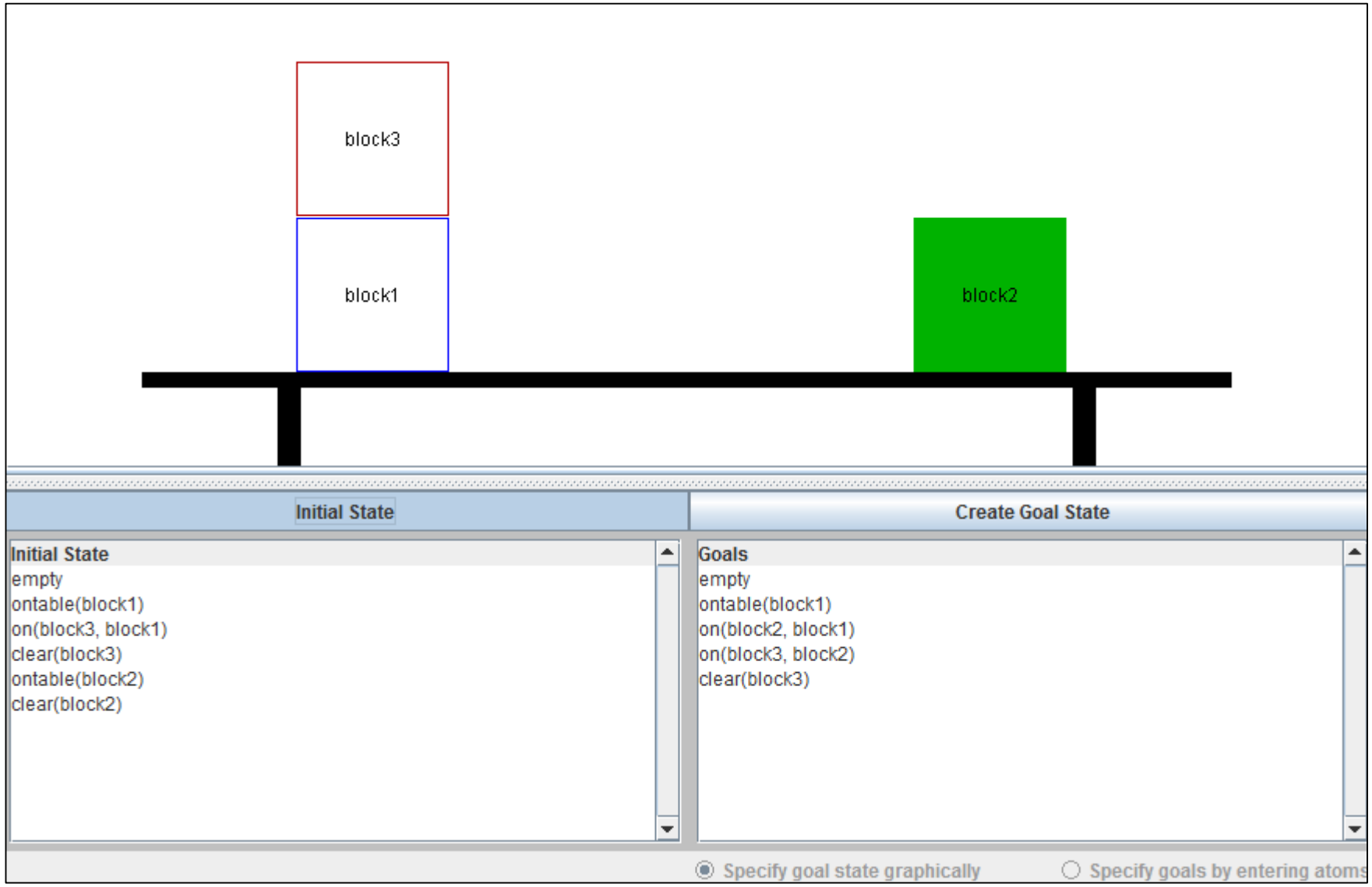
Grasp(Bananas)]

B

A

C

AI Space Planning Applet Demo



Complexity

PlanSAT: is there any plan that solves a given planning problem?

BoundedPlanSat: is there any plan that solves a given planning problem in k steps of less?

Both of these problems are decidable for classical planning.

There is an algorithm to find a plan if it exists.

Both problems are PSPACE hard (harder than NP).

Many real-world problems are much easier than the worst case.

To solve planning problems real-world planning problems, we'll need good heuristics.