IA Planning
Course 4: Planning Under Uncertainty (PPDDL)

Halim Djerroud

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

revision: 0.1

Plan du cours

Planning Under Uncertainty

Introduction to Planning Under Uncertainty

Chapter Objectives:

1. Motivations
   - Uncertainty in action
   - Uncertainty in perception
2. Limitations of classical PDDL
3. Concrete examples (mobile robot, uncertain manipulation)

Context: Why Introduce Uncertainty?

Classical Deterministic Planning (STRIPS/PDDL)

- Perfectly known states
- Actions always succeed
- Deterministic effects

Limitations in the Real World

- **Unreliable actions**: a robot may slip, miss a grasp
- **Uncertain observations**: noisy sensors, imperfect vision
- **External changes**: dynamic environment
- **Risk and cost**: unforeseen consequences of a bad choice

Motivations: When Uncertainty Is Necessary

In mobile robotics:

- The robot deviates slightly during movement
- The laser sensor returns incomplete measurements
- Some surfaces cause slipping

In manipulation:

- The grasp may fail
- The object may fall or move
- Object properties are uncertain

Key idea: An action can lead to multiple possible outcomes.

Limitations of Classical PDDL

Assumes a Perfect World

- States observable without error
- Actions always executed correctly
- Unique and deterministic effects

Consequences

- **Unrealistic modeling** for robots and autonomous systems
- **No risk management**
- Impossible to express:
  - probability of failure,
  - perceptual uncertainty,
  - decisions based on rewards.

Concrete Examples

Mobile Robot

- Action: move forward 1 m
- Possible outcomes:
  - 0.8: moves correctly
  - 0.15: slips slightly
  - 0.05: hits obstacle and doesn't move

Object Manipulation

- Action: grasp the object
- Possible outcomes:
  - 0.9: successful grasp
  - 0.1: failed grasp or object falls

Problem: Classical PDDL cannot represent these probabilistic transitions.

Toward Probabilistic Planning

Objective

Model **non-deterministic** actions with multiple possible outcomes.

Fundamental Idea

- Each action is a **probabilistic draw**
- The plan must account for these uncertainties
- Decisions motivated by **expected reward**

Consequence

We need a formalism that allows:

- probabilistic transitions,
- rewards,
- robust policies.

Markov Chains

Markov Chains

Chapter Objectives:

1. Fundamental definitions
   - States
   - Transitions
   - Markov property
2. Representation: transition matrix
3. Illustrative example (slipping robot)

Markov Chains: Introduction

Why Introduce Markov Chains?

- Actions don't always have a unique outcome
- The system can evolve in multiple possible ways
- Transitions depend only on the current state

Objective

Understand the mathematical formalism for modeling:

- Possible states of a system
- Probabilistic transitions between these states
- Stochastic behavior of an agent or robot

Fundamental Definitions

State

- A possible configuration of the system
- Ex: robot position, orientation, sensor state...

Transition

- Moving from state s to state s'
- Associated with probability $P(s' \mid s)$

Markov Property

- The future depends only on the **current state**
- No need for history

$$P(s_{t+1} \mid s_t, s_{t-1}, \ldots) = P(s_{t+1} \mid s_t)$$

Representation: Transition Matrix

Definition

A matrix P such that:

$$P[i,j] = P(s_j \mid s_i)$$

Where each row represents a current state, and each column a future state.

Example: 3 states

$$P = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.3 & 0.3 & 0.4 \end{pmatrix}$$

- Row 1: Probabilities starting from state s_1
- Each row sums to 1
- Allows studying system evolution

The transition matrix is the **standard representation** of a Markov chain.

Illustrative Example: Slipping Robot

Situation: A robot wants to move forward one square, but the floor is slippery.

States

- s_1: current position
- s_2: moves correctly
- s_3: slips and deviates

Transition Matrix

$$P = \begin{pmatrix} 0 & 0.8 & 0.2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

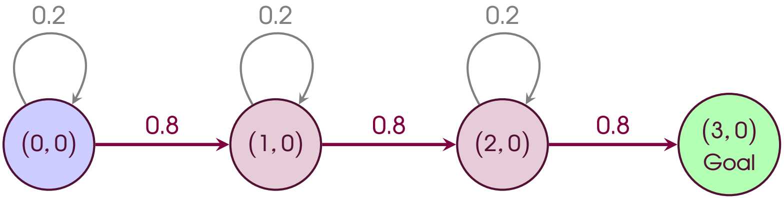
Transitions

- 0.8: moves correctly ($s_1 \longrightarrow s_2$)
- 0.2: slips ($s_1 \longrightarrow s_3$)

- s_2 and s_3 are absorbing states in this simple example

Conclusion: The same movement can have multiple outcomes -> essential concept for PPDDL.

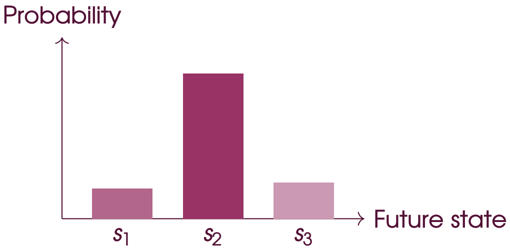
Markov Chain: Transition Diagram



- Each node represents a possible **state** of the robot.
- Each arrow represents a **transition** with an associated probability.
- Here: a single action (*move forward*) but multiple possible outcomes.

Probability Distribution Over Future States

After an action from s_1, the future state is not certain:



Consequence: The planner must reason about state *distributions*.

From Markov Chains to MDPs

Limitation of Markov Chains

- No notion of action or control
- Evolution is purely stochastic
- How can an agent influence the system?

Solution: Markov Decision Process (MDP)

- Adds **actions** chosen by the agent
- Transitions: $P(s' \mid s, a)$
- Adds **rewards** to guide behavior
- Foundation of reinforcement learning

Markov Decision Processes (MDP)

MDP: Formal Model of Decision-Making Under Uncertainty

Chapter Objectives:

1. Understand the components of an MDP
   - States and state space
   - Actions and their probabilistic effects
   - Transition function
   - Reward function
2. Define and compute an optimal policy
3. Markov property and implications
4. Application: autonomous navigation robot

Markov Decision Processes (MDP)

Why MDPs?

- Model uncertainty inherent to real actions
- Optimize sequential decisions under uncertainty
- Balance immediate gains and future consequences
- Mathematically formalize robotic decision-making

Formal Definition

An MDP is defined by a quadruple: $\mathcal{M} = (S, A, P, R)$ where:

- S: finite or countable set of states
- A: finite set of actions
- $P : S \times A \times S \rightarrow [0, 1]$: probabilistic transition function
- $R : S \times A \rightarrow \mathbb{R}$: reward function

Markov Property

Fundamental Assumption

The **Markov property** states that the future state depends only on the present state and chosen action, not on history:

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, \ldots, s_0) = P(s_{t+1} \mid s_t, a_t)$$

Practical Consequences

- The state must contain *all* necessary information
- Considerable algorithmic simplification
- Limitation: some problems require memory

"The future is independent of the past given the present"

MDP Components: States

States (S)

- Describe the complete system configuration at a given instant
- Must satisfy the Markov property
- Can be discrete or continuous (discretized in practice)

Concrete Examples

- **Mobile robot:** $(x, y, \partial, v, \text{battery})$
- **Manipulator arm:** $(\partial_1, \partial_2, \ldots, \partial_n, \text{object\_grasped})$
- **Drone:** $(x, y, z, \text{roll}, \text{pitch}, \text{yaw}, \text{velocities})$

Warning

A poorly designed state that omits critical information violates the Markov property and compromises solution optimality.

MDP Components: Actions

Actions (A or $A(s)$)

- Set of possible choices for the agent
- Can depend on the state: $A(s) \subseteq A$
- Represent control commands

Discrete Actions

- Up, Down, Left, Right
- Grasp, Release, Open
- Accelerate, Brake, Turn

Continuous Actions

- Velocity: $v \in [0, v_{\text{max}}]$
- Angle: $\theta \in [0, 2\pi]$
- Force: $F \in \mathbb{R}^3$

(discretized in practice)

MDP Components: Transition Function

Transition Probabilities ($P(s' \mid s, a)$)

- Model the **stochastic** effects of actions
- Capture real-world uncertainty
- Define a probability distribution: $\sum_{s' \in S} P(s' \mid s, a) = 1$

Example: Mobile Robot with Slipping

Action "move forward" from position $(0, 0)$:

- $P((1, 0) \mid (0, 0), \text{forward}) = 0.8$ ✓ success
- $P((0, 0) \mid (0, 0), \text{forward}) = 0.15$ ∼ slipping
- $P((0, 1) \mid (0, 0), \text{forward}) = 0.05$ × deviation

Unlike classical planning, an action does not guarantee its outcome!

MDP Components: Reward Function

Rewards ($R(s, a)$ or $R(s, a, s')$)

- Scalar signal measuring the "quality" of an action
- Encode the problem's objectives and constraints
- Can be positive (rewards) or negative (costs/penalties)

Examples of Reward Design

- **Goal reached:** $R(s_{goal}, \cdot) = +100$
- **Movement cost:** $R(s, \text{forward}) = -1$ (encourages efficiency)
- **Collision:** $R(s_{obstacle}, a) = -100$ (strong penalty)
- **Low battery:** $R(s_{battery<10\%}, a) = -50$

Critical Design : The reward function design determines learned behavior. A poorly defined reward can produce undesired behaviors!

Horizon and Discount Factor

Planning Horizon

- **Finite horizon:** planning over T steps
- **Infinite horizon:** planning without time limit

Discount Factor (γ)

For infinite horizons, we introduce $\gamma \in (0, 1]$:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s\right]$$

Interpretation:

- γ close to 1: long-term vision ($\gamma = 0.99$)
- γ close to 0: preference for immediate gains ($\gamma = 0.5$)
- Guarantees mathematical convergence ($\sum_{t=0}^{\infty} \gamma^t R_{max} < \infty$)

Policy and Value Function

Policy (π)

A policy is a decision strategy:

$$\pi : S \to A \quad \text{or} \quad \pi : S \times A \to [0, 1] \text{ (stochastic)}$$

Deterministic policy: $\pi(s) = a$ (one action per state)
Stochastic policy: $\pi(a \mid s)$ (distribution over actions)

Value Function: Measures the quality of a state under policy π:

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s\right]$$

Action-value function (Q-function):

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s')$$

Optimal Policy

Objective of Probabilistic Planning

Find an optimal policy π^* that maximizes expected value:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in S$$

Properties of the optimal policy:

- There always exists at least one deterministic optimal policy
- All optimal policies share the same value function V^*
- $V^*(s)$ satisfies the Bellman equation:

$$V^*(s) = \max_{a} \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right]$$

Final goal: act optimally by maximizing expected cumulative reward despite uncertainty.

Solution Algorithms

Main Methods

Dynamic programming: Value Iteration, Policy Iteration

Monte Carlo methods: trajectory sampling

Temporal difference learning: Q-Learning, SARSA

Value Iteration (overview): Iterate until convergence:

$$V_{k+1}(s) \leftarrow \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V_k(s') \right]$$

Then extract policy:

$$\pi^*(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right]$$

Complexity: $O(|S|^2|A|)$ per iteration

Example: MDP for Navigation Robot

States

- s_1: position A (start)
- s_2: position B (goal)
- s_3: obstacle/collision

Actions

- move forward
- turn
- move backward

Transitions

Action "move forward" from s_1:

$$P(s_2 \mid s_1, \text{forward}) = 0.7$$

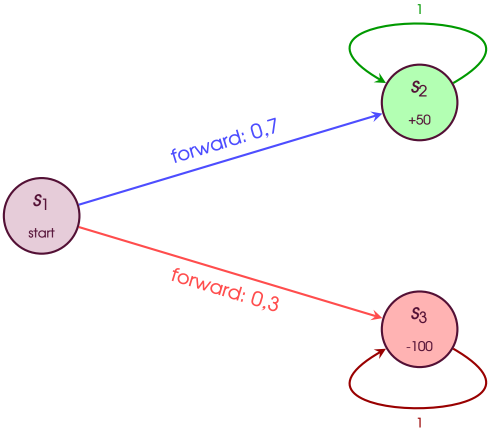
$$P(s_3 \mid s_1, \text{forward}) = 0.3$$

Rewards

- $R(s_2, \cdot) = +50$
- $R(s_1, \cdot) = -1$
- $R(s_3, \cdot) = -100$

Question: which action maximizes expected reward from s_1?

MDP Diagram: Navigation Robot



Expected value calculation (with $\gamma = 0.9$):

$$V(s_1) = -1 + 0.9 \times [0.7 \times 50 + 0.3 \times (-100)] = -1 + 0.9 \times 5 = \mathbf{3.5}$$

Extended Example: Action Selection

Complete Scenario

Let's add the "wait" action from s_1:

- $P(s_1 \mid s_1, \text{wait}) = 1.0$ (stays in place)
- $R(s_1, \text{wait}) = -2$ (waiting cost)

Comparison of Action Values

$$Q(s_1, \text{forward}) = -1 + 0.9 \times [0.7 \times 50 + 0.3 \times (-100)] = 3.5$$

$$Q(s_1, \text{wait}) = -2 + 0.9 \times V(s_1) = -2 + 0.9 \times 3.5 = 1.15$$

Optimal policy: $\pi^*(s_1) = \text{forward}$
because $Q(s_1, \text{forward}) > Q(s_1, \text{wait})$

MDP vs Classical Planning

Characteristic	Classical Planning	MDP
Determinism	Yes	No (stochastic)
Objective	Plan (sequence)	Policy (rule)
Uncertainty	Ignored	Explicitly modeled
Solution	Action sequence	Function $\pi : S \to A$
Optimization	Length/cost	Expected reward
Complexity	PSPACE-complete	P (fixed size)

When to Use MDPs?

- Actions with uncertain outcomes (real robotics)
- Dynamic and unpredictable environments
- Need to optimize over stochastic trajectories
- Availability of a probabilistic model of the world

Chapter Summary

Key Concepts

- **MDP =** (S, A, P, R): formalism for decision-making under uncertainty
- **Markov property**: the future depends only on the present
- **Policy** π: decision strategy for each state
- **Value** $V^\pi(s)$: expected cumulative reward
- **Optimality**: value maximization via Bellman equation

Key Takeaways

- MDPs generalize planning to stochastic environments
- The solution is a policy, not a fixed plan
- The exploration/exploitation tradeoff is crucial
- Applications: robotics, games, autonomous systems

Review of PDDL

Essential PDDL Review

Chapter Objectives:

1. Domains and problems
2. Predicates, types, objects
3. Deterministic actions
4. Limitations when facing uncertainty

Review: Domains and Problems in PDDL

Domain (describes the model)

- Defines the types, predicates and actions of the domain
- Specifies **general rules** applicable everywhere
- Ex.: robotics, logistics, navigation...

Problem (describes an instance)

- Lists the objects of the instance
- Describes the initial state
- Indicates the goal to reach

A **domain** contains general rules; a **problem** describes a particular situation.

Predicates, Types and Objects

Types

- Categories of objects
- Examples: `robot`, `location`, `object`

Objects

- Concrete elements of the problem
- Examples: `robot1`, `table1`, `roomA`

Predicates

- Describe properties of the world
- Examples:
  - `(at robot1 roomA)`
  - `(holding robot1 object1)`

Deterministic Actions in PDDL

Structure of an Action

- **Preconditions**: what must be true before
- **Effects**: what changes after the action

Example: Robot Movement

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

- Effects are **deterministic**: only one possible outcome
- The planner searches for a sequence of actions leading to the goal

Limitations of PDDL When Facing Uncertainty

Problems in Real Environments

- **Unreliable actions**: multiple possible outcomes
- **Noisy effects**: imperfect sensors
- **Unforeseen changes**: world dynamics
- **Costs/rewards** impossible to express

Consequences

- Overly simplistic modeling for robotics
- Inability to represent transition distributions
- No consideration of **risk**

Conclusion: PDDL is limited for uncertain worlds $\longrightarrow$ need for probabilistic PDDL: **PPDDL**.

Introduction to PPDDL

Why Probabilistic PDDL?

Chapter Objectives:

1. Relationship between PPDDL and MDP
2. Major extensions: probabilistic effects, conditions, dead-ends, rewards
3. General syntax overview

Why PPDDL?

Limitations of Classical PDDL

- Deterministic actions only
- No way to express uncertainty
- Impossible to represent transition probabilities
- No rewards or costs

Need

Model a world where multiple outcomes are possible for the same action.

Relationship Between PPDDL and MDP

MDP

An MDP is defined by:

$$\mathcal{M} = (S, A, P, R)$$

PPDDL ↔ MDP Correspondence

- States S: defined by predicates
- Actions A: PPDDL actions
- Transitions $P(s' \mid s, a)$: `(probabilistic ...)`
- Rewards R: `(:metric maximize (reward))`

Major PPDDL Extensions

- Probabilistic effects
- Probabilistic conditional effects
- Failure states (dead-ends)
- Rewards

> PPDDL extends PDDL to represent complete MDPs.

Probabilistic Effects

Idea

The same action can produce multiple possible effects, each with an associated probability.

Example

```
(probabilistic
0.8 (at robot room2)
0.2 (at robot room3))
```

MDP Interpretation of Probabilistic Effects

- $P(s_{\text{success}} \mid s, a) = 0.8$
- $P(s_{\text{slip}} \mid s, a) = 0.2$

The transition is a random draw among possible effects.

Probabilistic Conditional Effects

Principle

The probability depends on conditions true in the current state.

Example

```
(when (floor-wet)
  (probabilistic
    0.7 (slip)
0.3 (move-forward)))
```

Failure States (Dead-Ends)

Definition

A state from which no goal is reachable.

- collision
- broken object
- stuck robot

PPDDL planners must avoid these states.

Rewards

Motivation

PDDL does not allow minimizing a cost or maximizing a reward.

Declaration

```
(:metric maximize (reward))
```

- Action costs
- Goal bonus
- Dead-end penalties

General PPDDL Syntax

PPDDL Domain

- `(:requirements :probabilistic-effects :rewards)`
- Predicates, Probabilistic actions

PPDDL Action

- `:precondition`
- `:effect`
  - deterministic
  - `(probabilistic ...)`
  - `(when ...  (...))`

Problem

- initial state
- goal

Detailed PPDDL Syntax

Syntax and Probabilistic Constructs

Chapter Objectives:

1. PPDDL domains and requirements
2. Probabilistic actions:
   - `(probabilistic p1 eff1 p2 eff2 ...)`
   - Conditional effects
3. Rewards and metrics
4. Terminal states / dead-ends

Why PPDDL?

Link Between PPDDL and MDP

- PPDDL is an extension of PDDL that allows modeling **MDPs**.
- A PPDDL problem describes:
  - a set of states S (via predicates),
  - a set of actions A (as in PDDL),
  - probabilistic transitions $P(s' \mid s, a)$,
  - rewards $R(s, a)$.
- PPDDL planners seek an **optimal policy** rather than a simple plan.

PPDDL = PDDL + probabilities + rewards $\longrightarrow$ Complete MDP modeling.

PPDDL Extensions: Probabilistic Effects

Probabilistic Effects

- An action can lead to multiple possible outcomes
- Each effect is associated with a probability

Example

```
(probabilistic
0.8 (at robot roomB)
0.2 (at robot roomC))
```

- Allows representing uncertain actions (slipping, failure...)

PPDDL Extensions: Probabilistic Conditional Effects

Idea

Probabilities can depend on conditions in the current state.

Example

```
(when (battery-low)
    (probabilistic
      0.9 (failure)
     0.1 (success)))
```

- Useful for modeling sensor noise or robot wear

PPDDL Extensions: Failure States (Dead-Ends)

Definition

A **dead-end** state is a state from which no plan can reach the goal.

Utility

- Models irreversible situations
- Examples:
  - broken object,
  - robot breakdown,
  - fatal collision.

PPDDL planners optimize to avoid these costly states.

PPDDL Extensions: Rewards

Motivation

Classical PDDL only allows expressing Boolean goals. PPDDL introduces a **reward** notion to guide decision-making.

Example

```
(:metric maximize (reward))
```

- Costs/bonuses can be added to actions
- Enables **reward-oriented planning**, as in MDPs

General PPDDL Syntax

PPDDL Domain:

- `(:requirements :probabilistic-effects :rewards)`
- Predicates
- Probabilistic actions

PPDDL Action Structure:

- `:precondition` | conditions as in PDDL
- `:effect` | can contain:
  - deterministic effects,
  - `(probabilistic p1 eff1 p2 eff2 ...)`,
  - conditional effects.

PPDDL Problem:

- Describes the initial state
- Indicates the goal
- Can contain a reward objective

Probabilistic Planning with PPDDL

Planners for PPDDL

Chapter Objectives:

1. Understand the PPDDL planner ecosystem
2. Use Safe-Planner for non-deterministic planning
3. Interpret generated policies
4. Differentiate linear plan and policy

Important Reminder

PPDDL allows modeling uncertainty, but not all planners support all language features!

PPDDL Planner Ecosystem

Historical Planners (IPC-4, 2004) :

- **mGPT** | Value Iteration / LRTDP (Bonet & Geffner)
- **FF-Replan** | Probabilistic extension of FF
- **RFF** | Replanning with probabilistic effects

Modern Planners :

- **PROST** | Monte-Carlo Tree Search (IPC winner 2011, 2014)
- **Safe-Planner** | Compilation to classical planning
- **pyRDDLGym** | Modern framework (RDDL, not PPDDL)

PPDDL Advantages

- Established standard (IPC)
- Syntax close to PDDL
- Rich documentation

Limitations

- Aging tools
- Complex installation
- RDDL more modern

Syntax Differences: probabilistic vs oneof

Standard PPDDL Syntax (probabilistic)

```
(:action move
  :parameters (?from ?to - location)
  :precondition (and (at ?from)
                     (connected ?from ?to))
  :effect (and
    (not (at ?from))
    (probabilistic
      0.8 (at ?to)         ; 80% success
      0.2 (at ?from)))) ; 20% failure
```

Safe-Planner Syntax (non-deterministic)

```
(:action move
  :parameters (?from ?to - location)
  :precondition (and (at ?from)
                     (connected ?from ?to))
  :effect (and
    (not (at ?from))
    (oneof
      (at ?to)        ; outcome 1
      (at ?from)))) ; outcome 2
```

Important

`oneof` = non-determinism (issues équiprobables)

`probabilistic` = probabilités explicites (mGPT, PROST)

Why Safe-Planner for Teaching?

Pedagogical Advantages

- Simple installation (Python + classical planner)
- No complex C++ compilation
- Uses FF or Fast-Downward (already known)
- Generates visual graphs (.dot)
- Readable source code

Limitations

- No numerical probabilities
- Non-determinism only
- No rewards

Recommended Approach

1. **Lectures**: Present complete PPDDL with `probabilistic`
2. **Theoretical exercises**: Probability calculations, optimal policies
3. **Practical labs**: Safe-Planner with `oneof`

Installing Safe-Planner

Prerequisites

```
# Install FF (Fast-Forward)
sudo apt-get install ff
# Clone Safe-Planner
git clone https://github.com/mokhtarivahid/safe-planner.git
cd safe-planner
# Test installation
./sp --help
```

File Structure

Safe-Planner requires two separate files:

- `domain.ppddl` | domain definition
- `problem.ppddl` | problem instance

Minimal Example: Navigation Robot

domain.ppddl

```
(define (domain navigation)
  (:requirements :strips
                 :typing
                 :non-deterministic)

  (:types location)

  (:predicates
    (at ?l - location)
    (connected ?from ?to - location))

  (:action move
    :parameters (?from ?to - location)
    :precondition (and
      (at ?from)
      (connected ?from ?to))
    :effect (and
      (not (at ?from))
      (oneof
        (at ?to)        ; success
        (at ?from)))) ; failure
)
```

problem.ppddl

```
(define (problem nav-3locs)
  (:domain navigation)

  (:objects
    A B C - location)

  (:init
    (at A)
    (connected A B)
    (connected B C)
    (connected B A)
    (connected C B))

  (:goal (at C))
)
```

Execution

```
./sp -d domain.ppddl \
     -p problem.ppddl \
     -c ff
```

Understanding Safe-Planner Output

Main Plan (optimistic path) :

```
@ PLAN
0: move(A, B)
1: move(B, C)
```

Subpaths (failure handling) :

```
@ SUBPATHS
State s0: (at A) → move(A, B)
  Success → s1: (at B)
  Failure → s0: (at A)     [loop: retry]

State s1: (at B) → move(B, C)
  Success → s2: (at C)     [GOAL]
  Failure → s1: (at B)     [loop: retry]
```

Fundamental Difference

Classical plan: linear sequence of actions

Policy: state → action function (handles all cases)

Visualization with .dot Files

Graph Generation

```
# Safe-Planner creates a .dot
./sp -d domain.ppddl \
     -p problem.ppddl \
     -c ff
# Convert to image
dot -Tpng policy.dot \
    -o policy.png
# Display
xdg-open policy.png
```

.dot Structure

```
digraph Policy {
  n0 [label="move(A,B)"];
  n1 [label="move(B,C)"];
  n2 [label="GOAL"];

  n0 -> n1 [label="success"];
  n0 -> n0 [label="fail"];
  n1 -> n2 [label="success"];
  n1 -> n1 [label="fail"];
}
```

Reading the graph:

Node = state where an action is recommended

Edge = possible transition (success/failure)

Loop = retry on failure

Advanced Safe-Planner Options

Useful Commands

```
# Verbose mode (level 0-2)
./sp -d domain.ppddl -p problem.ppddl -c ff -v 2
# Use Fast-Downward instead of FF
./sp -d domain.ppddl -p problem.ppddl -c fd
# Use multiple planners
./sp -d domain.ppddl -p problem.ppddl -c ff fd
# All-outcome compilation (all results in one domain)
./sp -d domain.ppddl -p problem.ppddl -c ff -a
# Reverse ranking of compiled domains
./sp -d domain.ppddl -p problem.ppddl -c ff -r
```

Compatible Planners

FF, Fast-Downward, OPTIC, MADAGASCAR, PROBE, VHPOP, LPG-TD, LPG

More Complex Example: Delivery Robot

```
(define (domain delivery)
  (:requirements :strips :typing :non-deterministic)

  (:types location package)

  (:predicates
    (robot-at ?l - location)
    (package-at ?p - package ?l - location)
    (holding ?p - package)
    (delivered ?p - package)
    (connected ?from ?to - location)
    (empty-hand))

  (:action move
    :parameters (?from ?to - location)
    :precondition (and (robot-at ?from) (connected ?from ?to))
    :effect (and (not (robot-at ?from))
                 (oneof (robot-at ?to) (robot-at ?from))))

  (:action pick
    :parameters (?p - package ?l - location)
    :precondition (and (robot-at ?l) (package-at ?p ?l) (empty-hand))
    :effect (and (holding ?p) (not (package-at ?p ?l)) (not (empty-hand))))

  (:action drop
    :parameters (?p - package ?l - location)
    :precondition (and (robot-at ?l) (holding ?p))
    :effect (and (not (holding ?p)) (empty-hand)
                 (oneof (and (package-at ?p ?l) (delivered ?p))
                        (package-at ?p ?l))))
)
```

Analysis of Generated Policy

Analysis Questions for Students

1. How many different states in the policy?
2. What happens if `move` fails 3 times in a row?
3. What is the minimum/maximum plan length?
4. Is the policy strong cyclic? (does it guarantee success?)

Quality Metrics

- **Deterministic**: plan length
- **Probabilistic**: expected number of actions
- **Non-deterministic**: guarantee of goal achievement

Theoretical Calculation

With success probability $p = 0.8$ for `move`:

Expected attempts before success: $E = \frac{1}{\cdot} = 1.25$

Plan vs Policy: Summary

Plan (deterministic) :

- Linear sequence
- No branching
- Predictable environment
- Ex: `[move(A,B), move(B,C)]`

Policy (probabilistic) :

- State → action function
- Handles failures
- Uncertain environment
- Ex: decision table

State	Action
(at A)	move(A, B)
(at B)	move(B, C)
(at C)	GOAL

Properties of a Good Policy

- **Completeness**: defined for all reachable states
- **Optimality**: minimizes expected cost
- **Strong cyclic**: guarantees goal achievement

Going Further

Resources

- **Safe-Planner**: `https://github.com/mokhtarivahid/safe-planner`
- **PPDDL Specification**: Younes & Littman (2004)
- **IPC-4 benchmarks**: `https://ipc04.icaps-conference.org`
- **PDDL Tutorials**: `https://planning.wiki`

Modern Alternatives

- **RDDL + pyRDDLGym** | modern syntax, well maintained
- **PROST** | if explicit probabilities needed (RDDL)
- **MDPSim** | simulator to evaluate PPDDL policies