AI Classical and Non-deterministic Planning: Model-based Autonomous Behavior

Sebastian Sardiña

School of Computing Technologies
RMIT University

Julio 28 - Agosto 1 2025

Part I

Non-deterministic Planning

Part 1: Non-deterministic Planning

Part 1: Non-deterministic Planning

Planning Models: Vanilla Model for Classical AI Planning

- finite and discrete state space S
- a **known initial state** $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a **deterministic transition function** $s' = f(a, s)$ for $a \in A(s)$
- positive **action costs** $c(a, s)$

A **solution**/**plan** is seq. of applicable actions $\pi = a_0, \ldots, a_n$ that maps s_0 into S_G.

Plan is **optimal** if it minimizes the **sum of action costs**.

ℹ️ Different **models** obtained by **relaxing** assumptions in **bold**.

Planning Models: Vanilla Model for Classical AI Planning

- finite and discrete state space S
- a **known initial state** $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a **deterministic transition function** $s' = f(a, s)$ for $a \in A(s)$ 👈
- positive **action costs** $c(a, s)$

A **solution/plan** is seq. of applicable actions $\pi = a_0, \ldots, a_n$ that maps s_0 into S_G.

Plan is **optimal** if it minimizes the **sum of action costs**.

ℹ️ Different **models** obtained by **relaxing** assumptions in **bold**.

Planning with non-deterministic actions

What if an action may yield different effect outcomes?

- **Slipery floor:** you may slip and fall (and maybe hurt yourself).
- **Slipery blocksworld:**
  if you stack or unstack a block, it may fall down to the table.
- **Dice rolling:** if you roll a die, it may yield different outcomes: 1,2,3,4,5 or 6.
- **Robot operation:** when using the gripper, it may succeed or fail to pick an object (and may need to retry).

Planning with non-deterministic actions

What if an action may yield different effect outcomes?

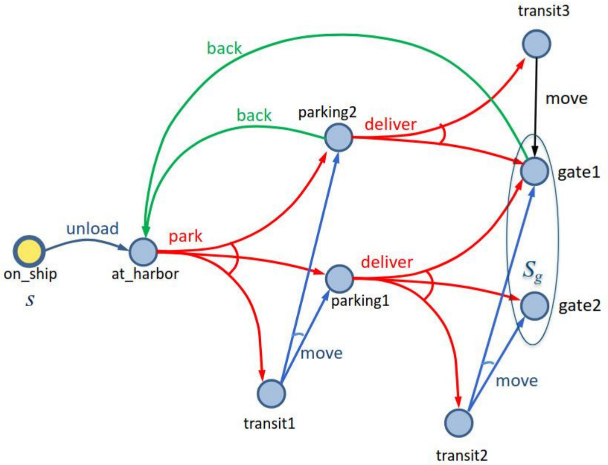
- **Slipery floor:** you may slip and fall (and maybe hurt yourself).
- **Slipery blocksworld:**
  if you stack or unstack a block, it may fall down to the table.
- **Dice rolling:** if you roll a die, it may yield different outcomes: 1,2,3,4,5 or 6.
- **Robot operation:** when using the gripper, it may succeed or fail to pick an object (and may need to retry).
- **Finding parking:** when visiting a block you may or may not find parking space (if not, keep going around the block).
- **Walking on beam:** if you do a step on a beam, you may advance or fall down.
- **Walking on corridor:** if you do a step you may or may not be at the end of the corridor.

Example: Harbor Management FOND Problem

Very simple harbor management domain:

1. Unload a single item from a ship.
2. Park the item in a storage facility.
3. Deliver it to gates (to be loaded into tracks).

Storage and gates may be unavailable, but we can always wait and move containers around.



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Planning with Markov Decision Processes

Goal MDPs are **fully observable, probabilistic** state models:

1. a state space S
2. initial state $s_0 \in S$
3. a set $G \subseteq S$ of goal states
4. actions $A(s) \subseteq A$ applicable in each state $s \in S$
5. **transition probabilities** $P_a(s' \mid s)$ for $s \in S$ and $a \in A(s)$ ☜
6. action costs $c(a, s) > 0$

- **Solutions** are **functions (called "policies")** mapping states into actions; $\pi : S \mapsto A$
  - $\pi(s)$ *states what action to do in state* s
- **Optimal** solutions minimize **expected cost** to goal.
- **Reward-based** MDPs involve **rewards** instead of costs, and **discount factor** $\gamma \in [0, 1)$ in place of goals. They underlie theory of RL. 😉

FOND Planning: Fully-observable Non-Deterministic Planning

A **FOND state model** is like the "logical" counterpart of Goal MDPs:

1. a state space S
2. initial state $s_0 \in S$
3. a set $G \subseteq S$ of goal states
4. actions $A(s) \subseteq A$ applicable in each state $s \in S$
5. **non-det state transition function** F: successors $s' \in F(a, s)$, $s \in S$, $a \in A(s)$ ☜
6. action costs $c(a, s) = 1$

- **Main change** from Classical Planning: $F(a, s)$ maps to set of possible states (not to one unique state).
  - Nature decides what next state is reached after action a is applied in state s — non-determinism.
  - ... but agent will observe the state reached after a is applied.

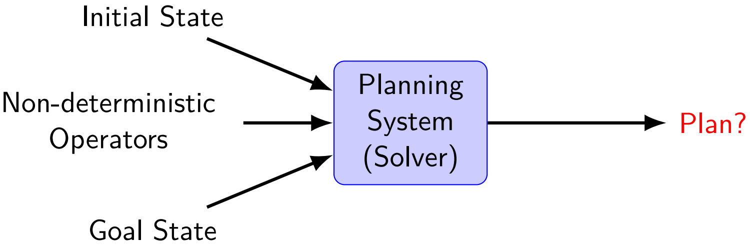
FOND Planning: Fully-observable Non-Deterministic Planning

A **FOND state model** is like the "logical" counterpart of Goal MDPs:

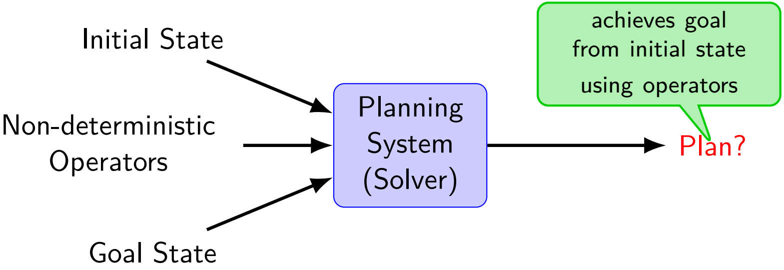
1. a state space S
2. initial state $s_0 \in S$
3. a set $G \subseteq S$ of goal states
4. actions $A(s) \subseteq A$ applicable in each state $s \in S$
5. **non-det state transition function** F: successors $s' \in F(a, s)$, $s \in S$, $a \in A(s)$ ☜
6. action costs $c(a, s) = 1$

- **Main change** from Classical Planning: $F(a, s)$ maps to set of possible states (not to one unique state).
  - Nature decides what next state is reached after action a is applied in state s — non-determinism.
  - ... but agent will observe the state reached after a is applied.
- **Main change** from MDPs: possible transitions $s \in F(a, s)$ not weighted by probabilities.

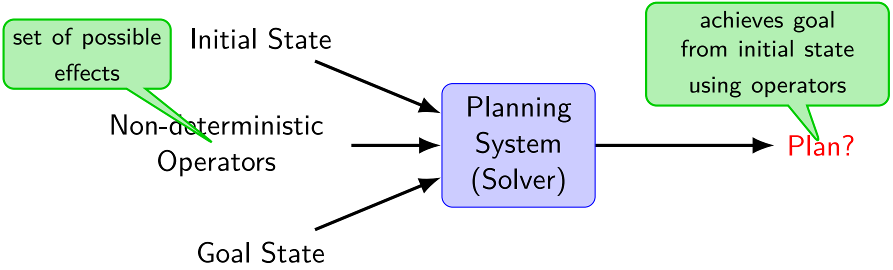
Fully Observable Non-Deterministic Planning (FOND)



Fully Observable Non-Deterministic Planning (FOND)

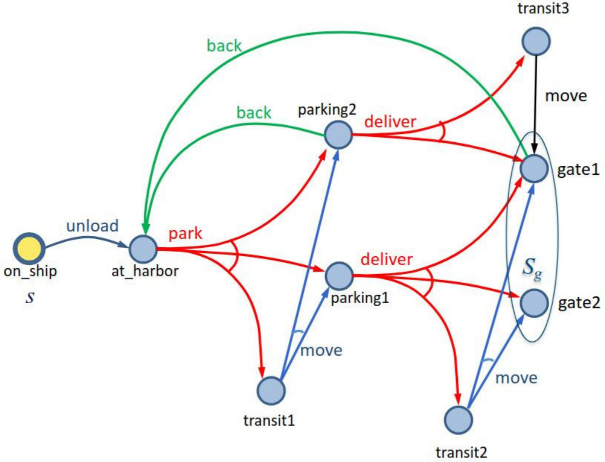


Fully Observable Non-Deterministic Planning (FOND)



Example: Does it have a solution?

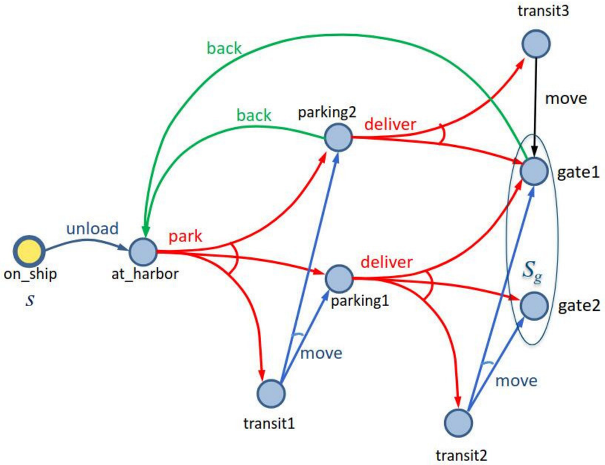
- Is it possible to always deliver the containers to the gates?
- If so, what is the sequence of actions?



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: Does it have a solution?

- Is it possible to always deliver the containers to the gates? ❓
- If so, what is the sequence of actions?

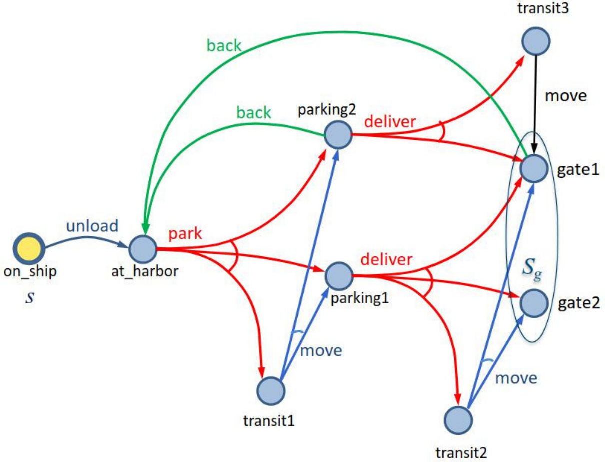


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: Does it have a solution?

- Is it possible to always deliver the containers to the gates? ❓
- If so, what is the sequence of actions? ❌

Need to know what to do in each state!



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: Does it have a solution?

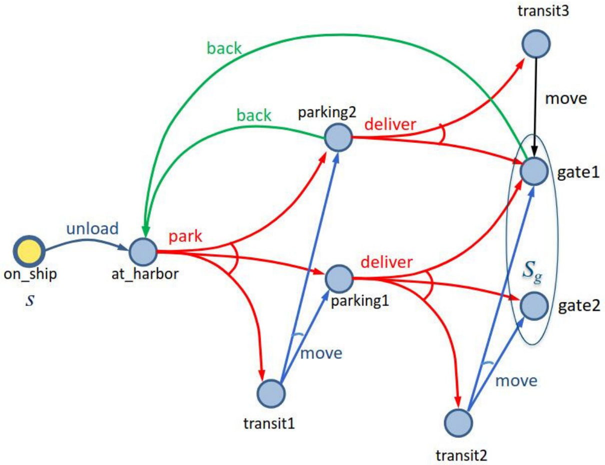
- Is it possible to always deliver the containers to the gates? ❓
- If so, what is the sequence of actions? ❌

Need to know what to do in each state!

Policy

A **policy** π is a partial function from states s into actions a; that is, $\pi : S \mapsto A$.

(when undefined, agent stops acting)



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: Does it have a solution?

- Is it possible to always deliver the containers to the gates? ❓
- If so, what is the sequence of actions? ❌

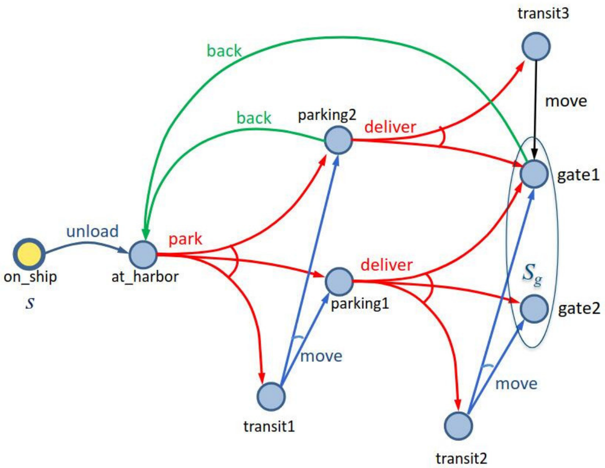
Need to know what to do in each state!

Policy

A **policy** π is a partial function from states s into actions a; that is, $\pi : S \mapsto A$.

(when undefined, agent stops acting)

🤔 Is there a "good" policy π?

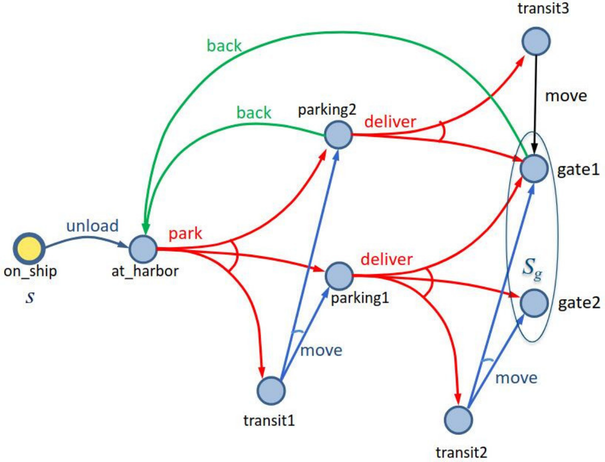


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: Does π_1 solve the task?

Policy π_1

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

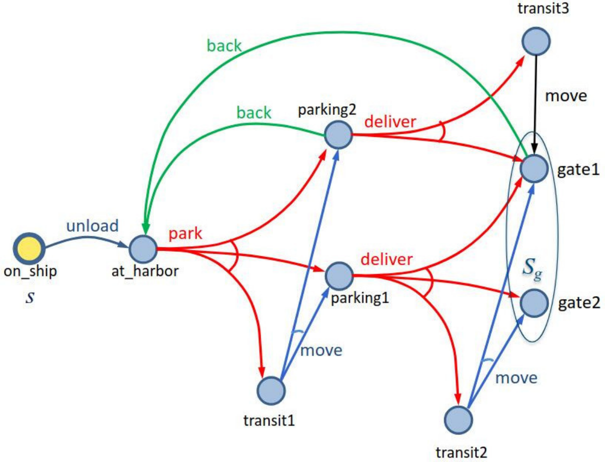


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: Does π_1 solve the task?

Policy π_1

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

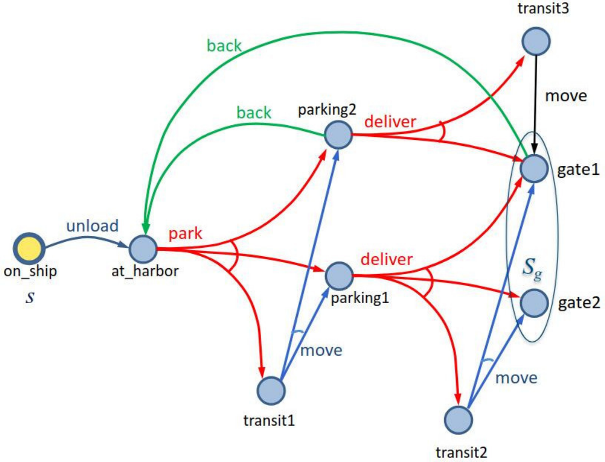


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: Does π_1 solve the task?

Policy π_1 ❌

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

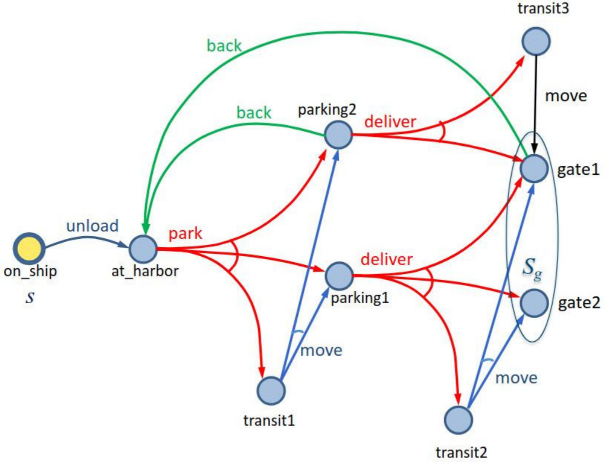


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: What about π_2?

Policy π_2

S	$\pi_2(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	deliver
transit1	move
transit2	move
transit3	move

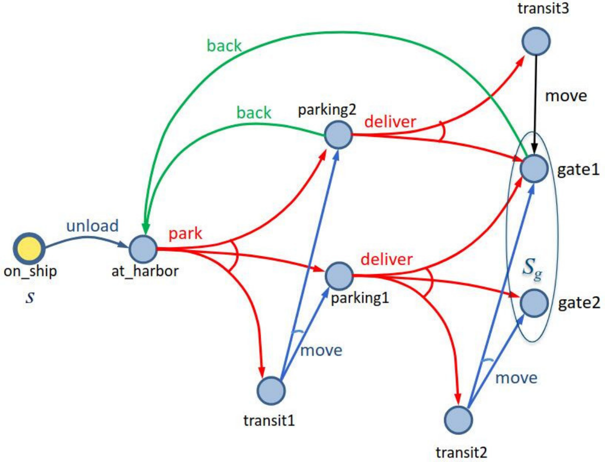


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: What about π_2?

Policy π_2 ✅

S	$\pi_2(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	deliver
transit1	move
transit2	move
transit3	move



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

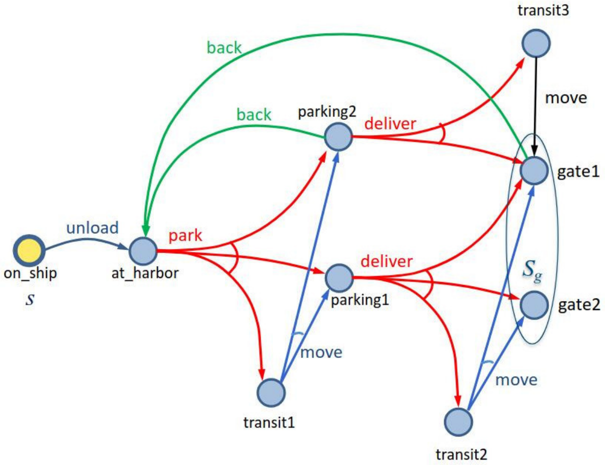
Example: Which one is better?

Policy π_2

S	$\pi(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

Policy π_4

S	$\pi(s)$
on_ship	unload
at_harbor	park



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

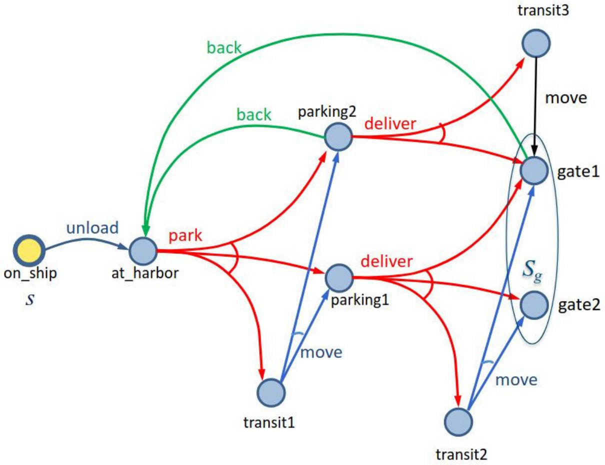
Example: Which one is better?

Policy π_2 ✅

S	$\pi(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

Policy π_4 ❌

S	$\pi(s)$
on_ship	unload
at_harbor	park

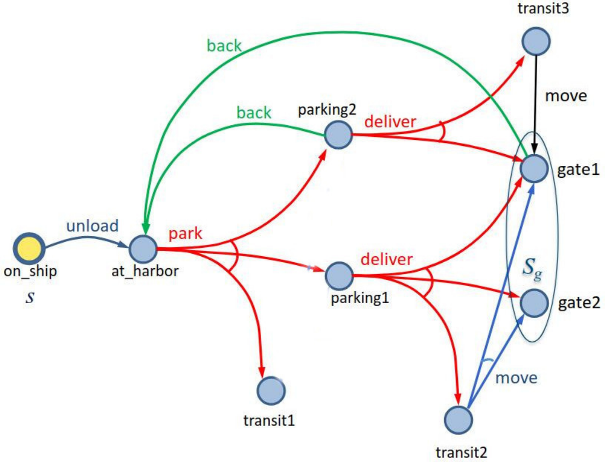


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: What if transit1 is a dead-end?

Policy π_2

S	$\pi_2(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	deliver
transit2	move
transit3	move



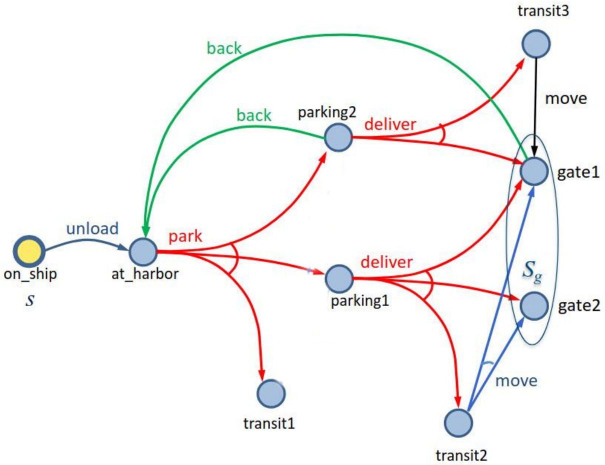
(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: What if transit1 is a dead-end?

Policy π_2 ❌

S	$\pi_2(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	deliver
transit2	move
transit3	move

But could π_2 succeed (sometimes)? 🤔

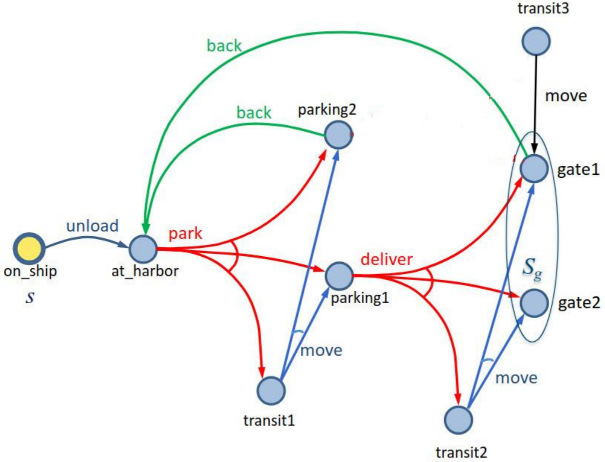


(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: What if parking2 is not connected to gates?

Policy π_1

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit2	move
transit3	move



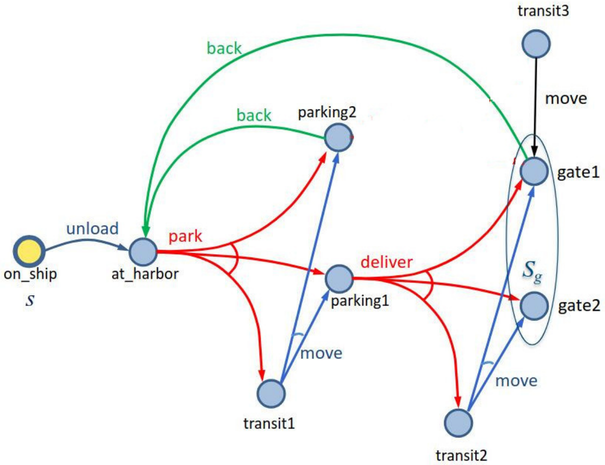
(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

Example: What if parking2 is not connected to gates?

Policy π_1 ❌

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit2	move
transit3	move

Storage parking1 may never be available!



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

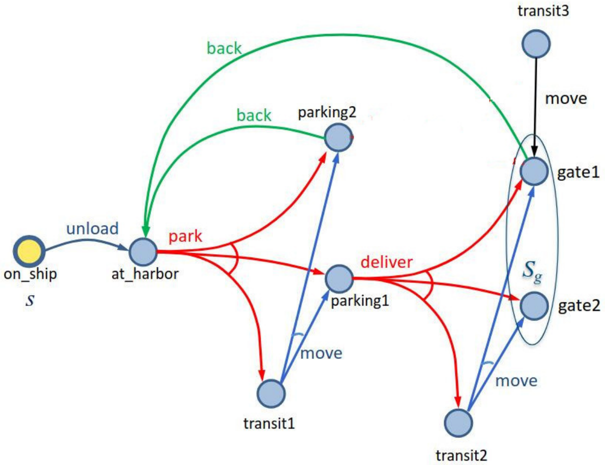
Example: What if parking2 is not connected to gates?

Policy π_1 ❌ ✅

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit2	move
transit3	move

Storage parking1 may never be available!

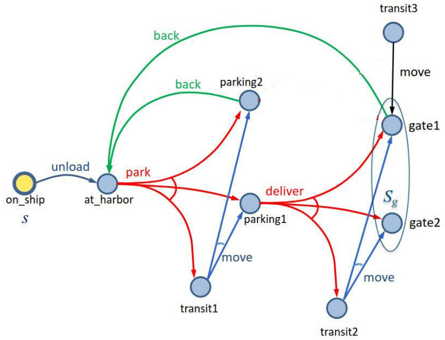
But, what if we know parking1 would eventually becomes available? 🤔



(Example 11.1 in *Acting, Planning, and Learning* Ghallab, Nau, Traverso 2025)

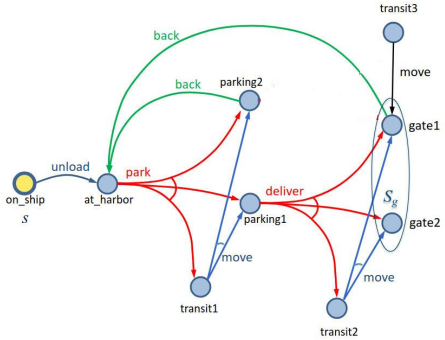
So, some lessons...

- Classical plans as sequences of actions are **not enough** to solve FOND problems.
- We need to use a **policy** that maps states into actions.
  - *More like "programs" with conditionals and loops!*
- Some (bad) policies are better than others.
- Some policies ***may* achieve** the goal, but not always.
- Some policies will achieve the goal *if* environment is **not too adversarial** — not unfair.



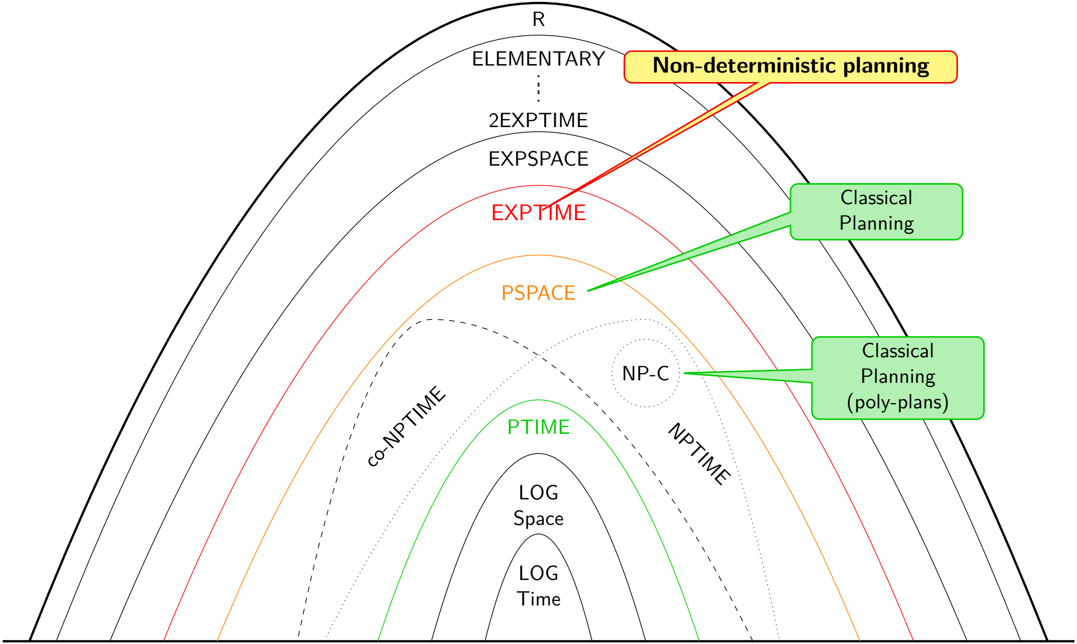
So, some lessons...

- Classical plans as sequences of actions are **not enough** to solve FOND problems.
- We need to use a **policy** that maps states into actions.
  - *More like "programs" with conditionals and loops!*
- Some (bad) policies are better than others.
- Some policies ***may* achieve** the goal, but not always.
- Some policies will achieve the goal *if* environment is **not too adversarial** — not unfair.

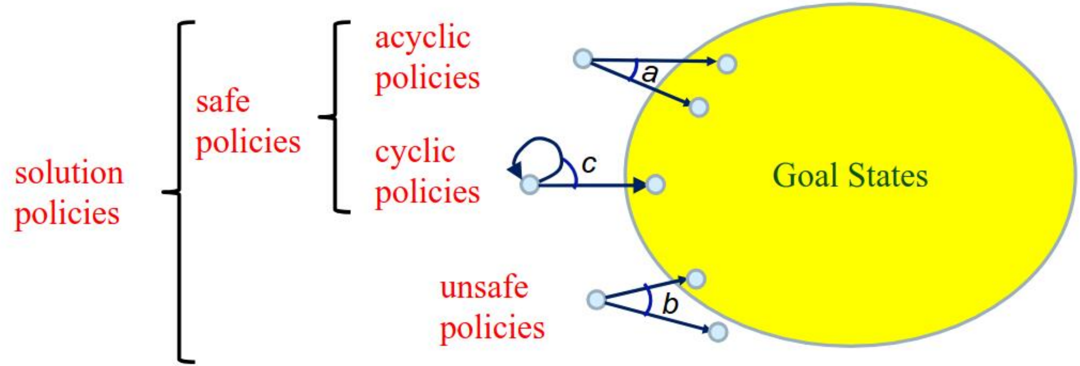


This seems way more complex planning! 😟

Planning is hard!



Kinds of Solution Policies



Acting, Planning, and Learning Ghallab, Nau, Traverso 2025

Part 1: Non-deterministic Planning

Part 1: Non-deterministic Planning

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \bot$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \bot$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \perp$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*
2. π is **strong solution** if all max π-trajectories from s_0 reach the goal.

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \bot$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*
2. π is **strong solution** if all max π-trajectories from s_0 reach the goal.
   - All executions are guaranteed to reach the goal (in a known bounded number of actions!).

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \perp$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*
2. π is **strong solution** if all max π-trajectories from s_0 reach the goal.
   - All executions are guaranteed to reach the goal (in a known bounded number of actions!).
   - Plans may have conditionals (but no loops!)

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \bot$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*
2. π is **strong solution** if all max π-trajectories from s_0 reach the goal.
   - All executions are guaranteed to reach the goal (in a known bounded number of actions!).
   - Plans may have conditionals (but no loops!)
3. π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \bot$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*
2. π is **strong solution** if all max π-trajectories from s_0 reach the goal.
   - All executions are guaranteed to reach the goal (in a known bounded number of actions!).
   - Plans may have conditionals (but no loops!)
3. π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.
   - Always a *possibility* to reach the goal.

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \bot$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*
2. π is **strong solution** if all max π-trajectories from s_0 reach the goal.
   - All executions are guaranteed to reach the goal (in a known bounded number of actions!).
   - Plans may have conditionals (but no loops!)
3. π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.
   - Always a *possibility* to reach the goal.
   - Goal will be achieved if environment is not “adversarial”

FOND Planning: Solution Concepts

Running policy π from state s yields trajectories runs:

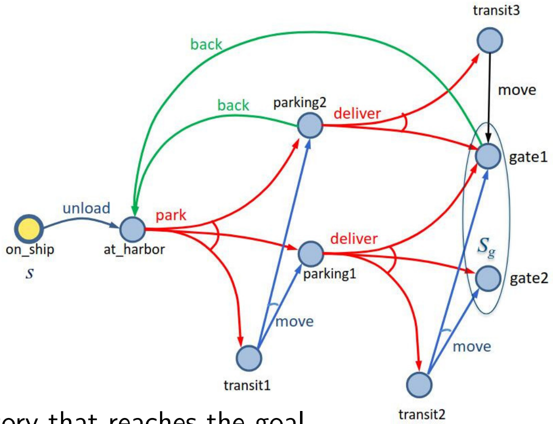
- π-**trajectories** $s_0, \ldots, s_n$, such that $s_{i+1} \in F(a_i, s_i)$, $a_i = \pi(s_i)$, for $i \in [0, n-1]$.
- π-trajectory **maximal** if 1) s_n is goal state, 2) $\pi(s_n) = \bot$, or 3) $n = \infty$ (π is infinite)

FOND Planning Solution Concepts

1. π is a **weak solution** if there is a π-trajectory from s_0 that reaches goal.
   - *At least one execution of the plan reaches the goal.*
2. π is **strong solution** if all max π-trajectories from s_0 reach the goal.
   - All executions are guaranteed to reach the goal (in a known bounded number of actions!).
   - Plans may have conditionals (but no loops!)
3. π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.
   - Always a *possibility* to reach the goal.
   - Goal will be achieved if environment is not "adversarial"
   - Plans may have conditionals & loops!

Weak Plans

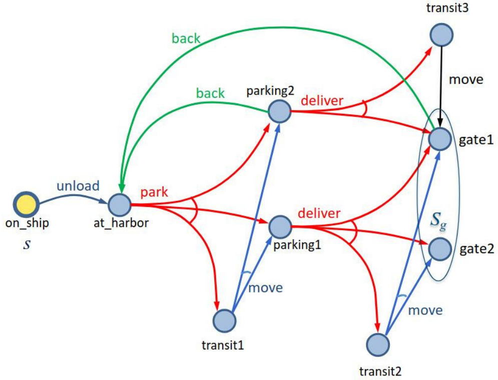
S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit2	move
transit3	move



- ✓ Policy π is a weak plan as there is a trajectory that reaches the goal.
  - *{on_ship}, {at_harbor}, {parking1}, {gate1}*
- ✖ But π is *not* a strong plan.
  - *{on_ship}, {at_harbor}, {parking2}, {at_harbor}, {parking2}, {at_harbor}, ...*

What about strong cyclic?

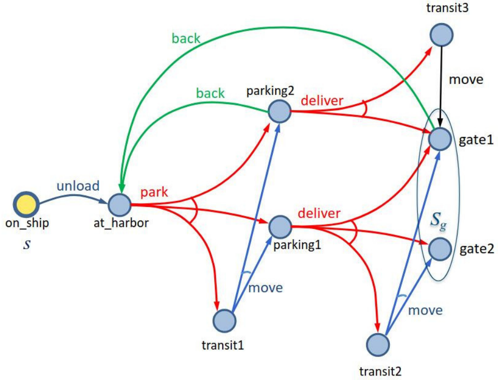
S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move



Policy π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.

What about strong cyclic?

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

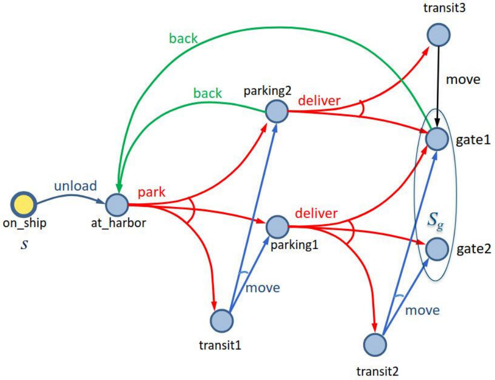


Policy π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.

- **Yes!**, policy never "loses" the possibility to get the goal 👍

What about strong cyclic?

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

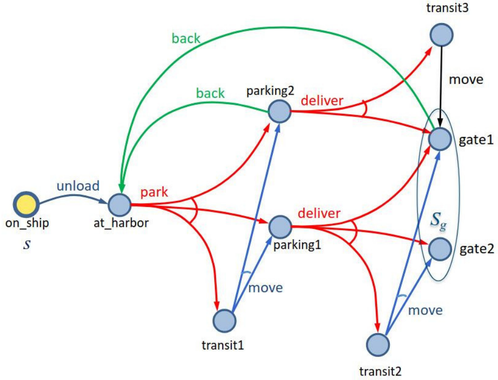


Policy π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.

- **Yes!**, policy never "loses" the possibility to get the goal 👍
- But, it may **loop** "forever" in some states.

What about strong cyclic?

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move

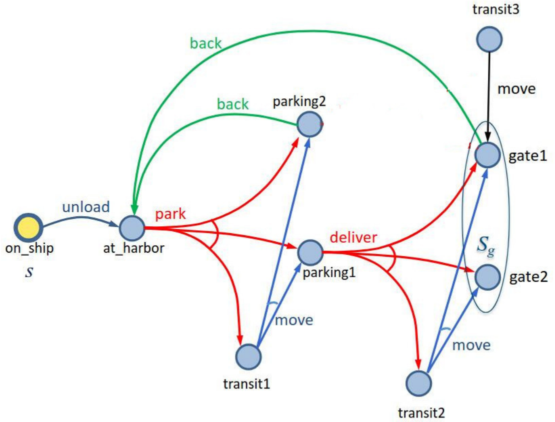


Policy π is **strong cyclic solution** if for each state s reachable from s_0 with a π-trajectory, there is a π-trajectory from s to goal.

- **Yes!**, policy never "loses" the possibility to get the goal 👍
- But, it may **loop** "forever" in some states.
- We can make π strong by changing it to $\pi_1(\text{parking2}) = \text{deliver}$.

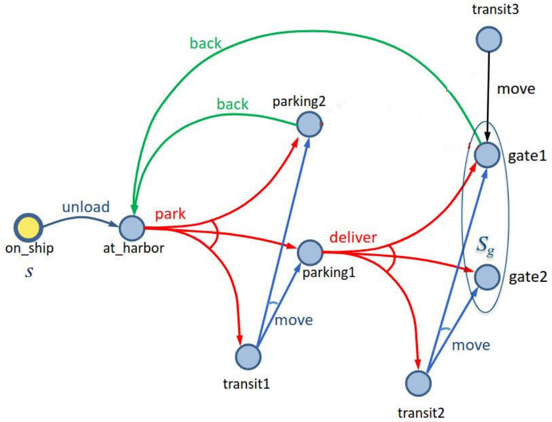
Strong cyclic policies: when do they work?

❓ Is there a strong plan?



Strong cyclic policies: when do they work?

❓ Is there a strong plan? **No!**

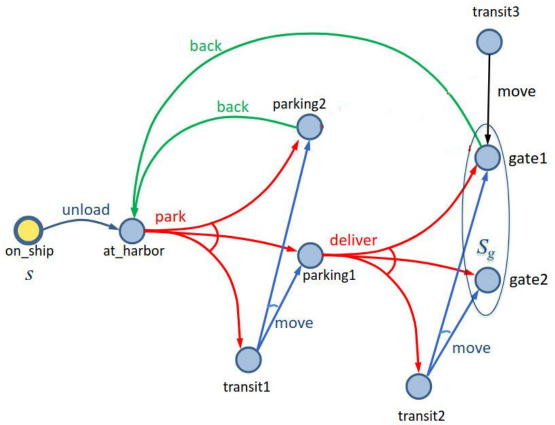


Strong cyclic policies: when do they work?

❓ Is there a strong plan? **No!**

Best we can do is:

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move



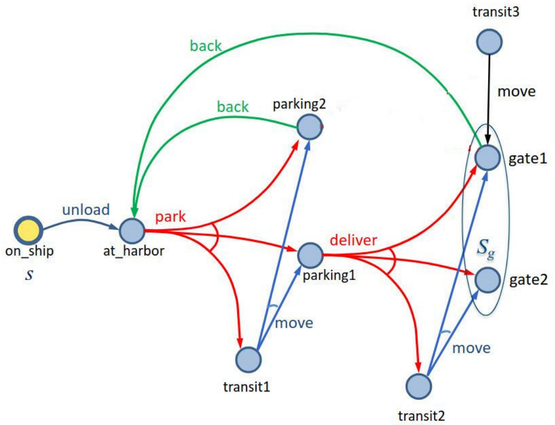
❓ **When will this policy reach the goal?**

Strong cyclic policies: when do they work?

❓ Is there a strong plan? **No!**

Best we can do is:

S	$\pi_1(s)$
on_ship	unload
at_harbor	park
parking1	deliver
parking2	back
transit1	move
transit2	move
transit3	move



❓ **When will this policy reach the goal?**

When executed in "**fair**" environments!

Fairness Environments

A trajectory σ is an **unfair** execution of π if a state s appears infinitely often in σ but some outcome state $s' \in F(\pi(a), s)$ only appears a finite number of times in σ.

Non-determinism behavior under fairness assumption

A strong cyclic policy eventually reaches the goal in every **fair** trajectory.

Fairness Environments

A trajectory σ is an **unfair** execution of π if a state s appears infinitely often in σ but some outcome state $s' \in F(\pi(a), s)$ only appears a finite number of times in σ.

Non-determinism behavior under fairness assumption

A strong cyclic policy eventually reaches the goal in every **fair** trajectory.

❓ What type of environments?

Fairness Environments

A trajectory σ is an **unfair** execution of π if a state s appears infinitely often in σ but some outcome state $s' \in F(\pi(a), s)$ only appears a finite number of times in σ.

Non-determinism behavior under fairness assumption

A strong cyclic policy eventually reaches the goal in every **fair** trajectory.

❓ What type of environments?

- Where each effect listed has indeed **non-zero probability**.
- **Re-trying** is an effective strategy.
  - rolling a die until it shows a 6.
  - driving around the block until a parking space is available.
  - pour into cup until full.

Fairness Environments

A trajectory σ is an **unfair** execution of π if a state s appears infinitely often in σ but some outcome state $s' \in F(\pi(a), s)$ only appears a finite number of times in σ.

Recap: Solution plans for FOND planning

- Classical sequential plans are not enough to solve FOND problems.
  - *We need more flexible behavior description (controllller) for agents*
- We use **policies** mapping states into actions.
  - *Allow conditional and loops.*

Recap: Solution plans for FOND planning

- Classical sequential plans are not enough to solve FOND problems.
  - *We need more flexible behavior description (controllller) for agents*
- We use **policies** mapping states into actions.
  - *Allow conditional and loops.*
- **Weak plans** may get the goal if we are lucky — not really adequate.
- **Strong plans** are very demanding: they require that all possible executions of the plan reach the goal. Often there is no strong plan! 😯

Recap: Solution plans for FOND planning

- Classical sequential plans are not enough to solve FOND problems.
  - *We need more flexible behavior description (controlller) for agents*
- We use **policies** mapping states into actions.
  - *Allow conditional and loops.*
- **Weak plans** may get the goal if we are lucky — not really adequate.
- **Strong plans** are very demanding: they require that all possible executions of the plan reach the goal. Often there is no strong plan! 😯
- **Strong-cyclic plans** are more flexible: they allow loops and conditionals, and they guarantee that the goal will be reached if the environment is **fair**.
- Many environments are fair: **retrying** is an effective strategy. 💪

Recap: Solution plans for FOND planning

- Classical sequential plans are not enough to solve FOND problems.
  - *We need more flexible behavior description (controlller) for agents*
- We use **policies** mapping states into actions.
  - *Allow conditional and loops.*
- **Weak plans** may get the goal if we are lucky — not really adequate.
- **Strong plans** are very demanding: they require that all possible executions of the plan reach the goal. Often there is no strong plan! 😯
- **Strong-cyclic plans** are more flexible: they allow loops and conditionals, and they guarantee that the goal will be reached if the environment is **fair**.
- Many environments are fair: **retrying** is an effective strategy. 💪

Question

How can we compute these plans with loops? How to compute strong-cyclic plans policies?

Part 1: Non-deterministic Planning

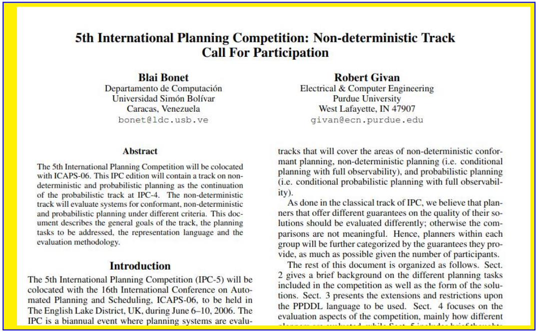
Part 1: Non-deterministic Planning

Non-determinism in PDDL

- Non-deterministic effects added to PDDL for the 5th IPC in 2006.
- Action effect can have a **one-of** effect:

```
(oneof e1 e2 ... en)
```

- To support uncertainty track in IPC-5.



5th International Planning Competition: Non-deterministic Track Call For Participation

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Robert Givan
Electrical & Computer Engineering
Purdue University
West Lafayette, IN 47907
givan@ecn.purdue.edu

Abstract

The 5th International Planning Competition will be colocated with ICAPS-06. This IPC edition will contain a track on non-deterministic and probabilistic planning as the continuation of the probabilistic track at IPC-4. The non-deterministic track will evaluate systems for conformant, non-deterministic and probabilistic planning under different criteria. This document describes the general goals of the track, the planning tasks to be addressed, the representation language and the evaluation methodology.

Introduction

The 5th International Planning Competition (IPC-5) will be colocated with the 16th International Conference on Automated Planning and Scheduling, ICAPS-06, to be held in The English Lake District, UK, during June 6–10, 2006. The IPC is a biannual event where planning systems are evalu-

tracks that will cover the areas of non-deterministic conformant planning, non-deterministic planning (i.e. conditional planning with full observability), and probabilistic planning (i.e. conditional probabilistic planning with full observability).

As done in the classical track of IPC, we believe that planners that offer different guarantees on the quality of their solutions should be evaluated differently; otherwise the comparisons are not meaningful. Hence, planners within each group will be further categorized by the guarantees they provide, as much as possible given the number of participants.

The rest of this document is organized as follows. Sect. 2 gives a brief background on the different planning tasks included in the competition as well as the form of the solutions. Sect. 3 presents the extensions and restrictions upon the PPDDL language to be used. Sect. 4 focuses on the evaluation aspects of the competition, mainly how different

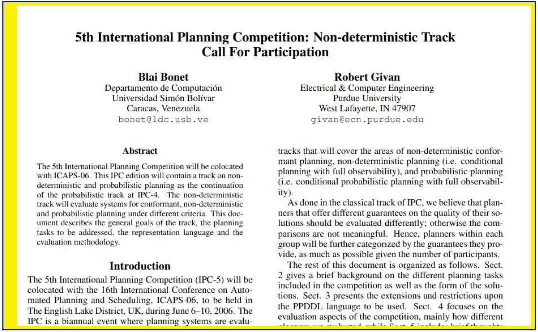
```
(:action unstack
    :parameters (?b1 ?b2 - block)
    :precondition (and (not (= ?b1 ?b2)) (emptyhand) (clear ?b1) (on ?b1 ?b2))
    :effect (oneof
      (and (holding ?b1) (clear ?b2) (not (emptyhand)) (not (clear ?b1)) (not (on ?b1 ?b2)))
      (and (clear ?b2) (on-table ?b1) (not (on ?b1 ?b2)))))
            ;; second effect: fail to grab; ?b1 ends on table
```

Non-determinism in PDDL

- Non-deterministic effects added to PDDL for the 5th IPC in 2006.
- Action effect can have a **one-of** effect:

  `(oneof e1 e2 ... en)`

- To support uncertainty track in IPC-5.


5th International Planning Competition: Non-deterministic Track
Call For Participation

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Robert Givan
Electrical & Computer Engineering
Purdue University
West Lafayette, IN 47907
givan@ecn.purdue.edu

Abstract

The 5th International Planning Competition will be colocated with ICAPS-06. This IPC edition will contain a track on non-deterministic and probabilistic planning as the continuation of the probabilistic track at IPC-4. The non-deterministic track will evaluate systems for conformant, non-deterministic and probabilistic planning under different criteria. This document describes the general goals of the track, the planning tasks to be addressed, the representation language and the evaluation methodology.

Introduction

The 5th International Planning Competition (IPC-5) will be colocated with the 16th International Conference on Automated Planning and Scheduling, ICAPS-06, to be held in The English Lake District, UK, during June 6–10, 2006. The IPC is a biannual event where planning systems are evalu-

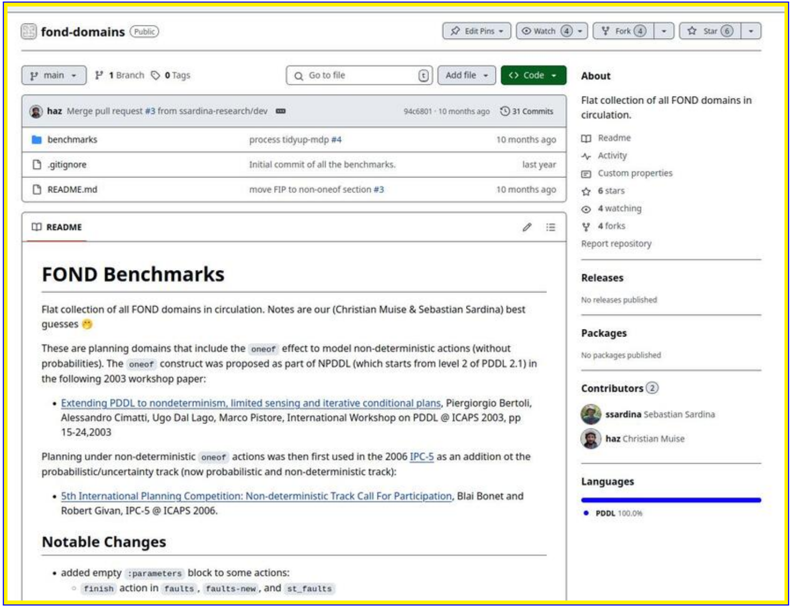
tracks that will cover the areas of non-deterministic conformant planning, non-deterministic planning (i.e. conditional planning with full observability), and probabilistic planning (i.e. conditional probabilistic planning with full observability).

As done in the classical track of IPC, we believe that planners that offer different guarantees on the quality of their solutions should be evaluated differently; otherwise the comparisons are not meaningful. Hence, planners within each group will be further categorized by the guarantees they provide, as much as possible given the number of participants.

The rest of this document is organized as follows. Sect. 2 gives a brief background on the different planning tasks included in the competition as well as the form of the solutions. Sect. 3 presents the extensions and restrictions upon the PPDDL language to be used. Sect. 4 focuses on the evaluation aspects of the competition, mainly how different

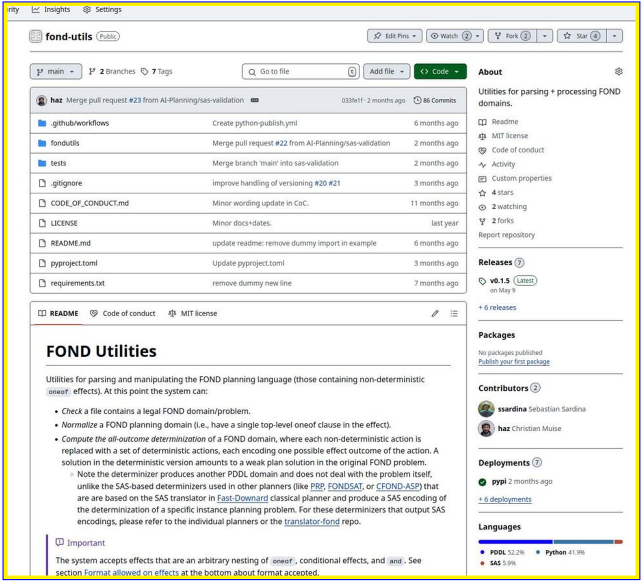
```
(:action pick-up-from-table
    :parameters (?b - block)
    :precondition (and (emptyhand) (clear ?b) (on-table ?b))
    :effect (oneof
        (and)    ;; no effect - things stay the same!
        (and (holding ?b) (not (emptyhand)) (not (on-table ?b)))))
```

AI-Planning/fond-domains @ GH: Benchmark for FOND



https://github.com/AI-Planning/fond-domains

AI-Planning/fond-utils @ GH: Utilities for FOND



https://github.com/AI-Planning/fond-utils

FOND Planning using Classical Planners

✅ One of the most effective ways to solve FOND planning problems is to use **classical planners**! Weird...? 😕

FOND Planning using Classical Planners

✅ One of the most effective ways to solve FOND planning problems is to use **classical planners**! Weird...? 😕

They all use a **deterministic relaxation** of the FOND problem:

All-outcome determinization

Deterministic relaxation P_D of FOND P obtained by substituing **non-det** actions a with effects $\{e_1, \ldots, e_n\}$ by **deterministic** actions $a^1, \ldots, a^n$, where a^i's effect is e_i, for $i \in [1, n]$.

- P_D is a deterministic classical planning problem.
- Under reasonable assumptions, P_D is polynomially larger than P.
- There are tools to do the determinization: https://github.com/AI-Planning/fond-utils

Week and Online Solutions for FOND Planning

⚙️ **Weak (open loop) solution** for P

From any **classical plan** ρ for P_D:

- If ρ generates trajectory $s_0, \ldots, s_N$ in P_D, set $\pi(s_i) = a$ if $\rho_i \in a$.
- Run π and hope for the best! 🤞

Week and Online Solutions for FOND Planning

⚙️ **Weak (open loop) solution** for P

From any **classical plan** ρ for P_D:

- If ρ generates trajectory $s_0, \ldots, s_N$ in P_D, set $\pi(s_i) = a$ if $\rho_i \in a$.
- Run π and hope for the best! 🤞

⚙️ **Online (closed loop) solution method for** P

Reach goal by interacting with FOND "system" that returns **observation** $s' \in F(a, s)$:

1. From **current state** s, initially s_0, compute **plan** $\rho = \rho_1, \ldots, \rho_N$ for $P_D[s]$.
2. Execute **prefix** $a_1, \ldots, a_i$ for $\rho_i \in a_i$ until state s_i **observed** is goal or **different** than state s'_i **predicted** in P_D.
3. If s_i is **goal**, exit; else set $s := s_i$ and go back to 1

Week and Online Solutions for FOND Planning

⚙️ **Weak (open loop) solution** for P

From any **classical plan** ρ for P_D:

- If ρ generates trajectory $s_0, \ldots, s_N$ in P_D, set $\pi(s_i) = a$ if $\rho_i \in a$.
- Run π and hope for the best! 🤞

⚙️ **Online (closed loop) solution method for** P

Reach goal by interacting with FOND "system" that returns **observation** $s' \in F(a, s)$:

1. From **current state** s, initially s_0, compute **plan** $\rho = \rho_1, \ldots, \rho_N$ for $P_D[s]$.
2. Execute **prefix** $a_1, \ldots, a_i$ for $\rho_i \in a_i$ until state s_i **observed** is goal or **different** than state s'_i **predicted** in P_D.
3. If s_i is **goal**, exit; else set $s := s_i$ and go back to 1

☀️ **Properties:** If **no dead-end states** reachable in P, under mild assumptions, goal state eventually reached. Else, method is **incomplete**.

PRP: Strong Cyclic Policies using Classical Planners

More powerful off-line method, can compute **strong cyclic policies**:

PRP: Planning for Relevant Policies (Muise, McIllraith, Beck ICAPS'12)

1. Run **simulated on-line** method not just from s_0 but from every possible sucessor s' of a (simulated) **observed** state s; i.e., $s' \in F(a,s)$ for a executed in s.
2. If state $s' \in F(a,s)$ is reached from which **no classical plan** for $P_D(s)$; **remove** a from $A(s)$, and **start all over again**.
3. Keep policy to $\pi(s) = a$ where deterministic version a_i is head of **shortest classical prefix** found from s to goal.

Properties:

- Method is **sound and complete**: returns strong cyclic policy if one exists. 👍
- More **scalable** than other methods as it uses **classical planners**.
- Can be made more efficient by **generalizing plans** using **regression**.
- Struggles if there are many "risky" nondeterminism leading to dead-ends.

Regression to Generalize Policies

Consider the following situation:

1. Goal is $G = \{g\}$.
2. Classical plan $\rho = a_1, \ldots, a_n$ optimally achieves G from state s_0 in P_D.
3. So, ρ yields trajectory $s_0, s_1, \ldots, s_n$ in P_D such that $g \in s_n$.
   - *The last action of ρ has $g \in \text{Add}(a_n)$ — a_n achieves the goal.*
4. The precondition of a_n is $\text{Pre}(a_n) = \{p, q\}$.
   - *Clearly, $p, q \in s_{n-1}$ – a_n's precondition hold just before the goal.*

So, we can set our FOND policy to $\pi(s_{n-1}) = a_n$.

Regression to Generalize Policies

Consider the following situation:

1. Goal is $G = \{g\}$.
2. Classical plan $\rho = a_1, \ldots, a_n$ optimally achieves G from state s_0 in P_D.
3. So, ρ yields trajectory $s_0, s_1, \ldots, s_n$ in P_D such that $g \in s_n$.
   - *The last action of ρ has $g \in \text{Add}(a_n)$ — a_n achieves the goal.*
4. The precondition of a_n is $\text{Pre}(a_n) = \{p, q\}$.
   - *Clearly, $p, q \in s_{n-1}$ – a_n's precondition hold just before the goal.*

So, we can set our FOND policy to $\pi(s_{n-1}) = a_n$. **Is that the best we can do?** 🤔

Regression to Generalize Policies

Consider the following situation:

1. Goal is $G = \{g\}$.
2. Classical plan $\rho = a_1, \ldots, a_n$ optimally achieves G from state s_0 in P_D.
3. So, ρ yields trajectory $s_0, s_1, \ldots, s_n$ in P_D such that $g \in s_n$.
   - *The last action of ρ has $g \in \text{Add}(a_n)$ — a_n achieves the goal.*
4. The precondition of a_n is $\text{Pre}(a_n) = \{p, q\}$.
   - *Clearly, $p, q \in s_{n-1}$ – a_n's precondition hold just before the goal.*

So, we can set our FOND policy to $\pi(s_{n-1}) = a_n$. **Is that the best we can do?** 🤔

What about any other state s' such that $p, q \in s'$?

Regression to Generalize Policies

Consider the following situation:

1. Goal is $G = \{g\}$.
2. Classical plan $\rho = a_1, \ldots, a_n$ optimally achieves G from state s_0 in P_D.
3. So, ρ yields trajectory $s_0, s_1, \ldots, s_n$ in P_D such that $g \in s_n$.
   - *The last action of ρ has $g \in \text{Add}(a_n)$ — a_n achieves the goal.*
4. The precondition of a_n is $\text{Pre}(a_n) = \{p, q\}$.
   - *Clearly, $p, q \in s_{n-1}$ – a_n's precondition hold just before the goal.*

So, we can set our FOND policy to $\pi(s_{n-1}) = a_n$. **Is that the best we can do?** 🤔

What about any other state s' such that $p, q \in s'$? **Can we also set $\pi(s') = a_n$?** 🤔

Regression to Generalize Policies

Consider the following situation:

1. Goal is $G = \{g\}$.
2. Classical plan $\rho = a_1, \ldots, a_n$ optimally achieves G from state s_0 in P_D.
3. So, ρ yields trajectory $s_0, s_1, \ldots, s_n$ in P_D such that $g \in s_n$.
   - *The last action of ρ has $g \in$ Add(a_n) — a_n achieves the goal.*
4. The precondition of a_n is Pre$(a_n) = \{p, q\}$.
   - *Clearly, $p, q \in s_{n-1}$ – a_n's precondition hold just before the goal.*

So, we can set our FOND policy to $\pi(s_{n-1}) = a_n$. **Is that the best we can do?** 🤔

What about any other state s' such that $p, q \in s'$? **Can we also set $\pi(s') = a_n$?** 🤔
YES! — $\{p, q\}$ is the *regression* of goal w.r.t. action a_n

Regression to Generalize Policies

Consider the following situation:

1. Goal is $G = \{g\}$.
2. Classical plan $\rho = a_1, \ldots, a_n$ optimally achieves G from state s_0 in P_D.
3. So, ρ yields trajectory $s_0, s_1, \ldots, s_n$ in P_D such that $g \in s_n$.
   - *The last action of ρ has $g \in$* Add(a_n) — *a_n achieves the goal.*
4. The precondition of a_n is Pre$(a_n) = \{p, q\}$.
   - *Clearly, $p, q \in s_{n-1}$ – a_n's precondition hold just before the goal.*

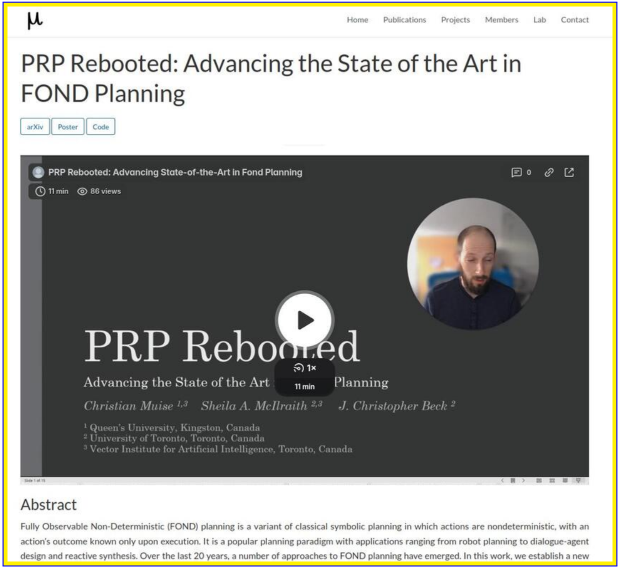
So, we can set our FOND policy to $\pi(s_{n-1}) = a_n$. **Is that the best we can do?** 🤔

What about any other state s' such that $p, q \in s'$? **Can we also set $\pi(s') = a_n$?** 🤔
YES! — $\{p, q\}$ is the *regression* of goal w.r.t. action a_n

Question

If Add$(a_{n-1}) = \{p\}$ and Pre$(a_{n-1}) = \{w\}$, what states s' can we set $\pi(s') = a_{n-1}$?

PRP Rebooted: AAAI'24



https://mulab.ai/project/pr2/

Shortcomings of Classical Planners for FOND

PRP scales wellas it uses **classical planners** + **regression**. However:

- Codebase is highly **sophisticated**; thousands of lines.
- Uses a lot of **tricks**: regression, dead-end detection, regression, loop closing, strong-cyclic check, etc.
- Struggle from "risky nondeterminism", where previous search choices need to be thrown and restarted.
  - *non-deterministic actions whose other effects will eventually lead to dead-ends.*
- May output very large policies — no guarantees of "compactness".
- Unable to handle **mixed fairness** environments.
  - *some actions are fair, others are unfair.*

Shortcomings of Classical Planners for FOND

PRP scales wellas it uses **classical planners** + **regression**. However:

- Codebase is highly **sophisticated**; thousands of lines.
- Uses a lot of **tricks**: regression, dead-end detection, regression, loop closing, strong-cyclic check, etc.
- Struggle from "risky nondeterminism", where previous search choices need to be thrown and restarted.
  - *non-deterministic actions whose other effects will eventually lead to dead-ends.*
- May output very large policies — no guarantees of "compactness".
- Unable to handle **mixed fairness** environments.
  - *some actions are fair, others are unfair.*

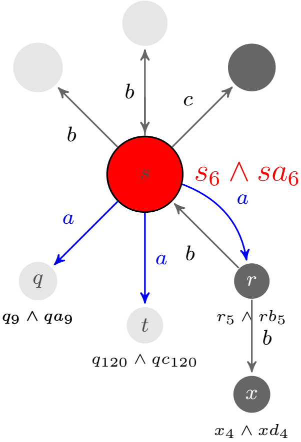
? What can we do about these issues? Can we get a simpler, declarative solver for FOND?

Recall Theory $C(P, n)$ for Classical Problem $P = \langle F, A, I, G \rangle$

- **Init:** p_0 for $p \in I$, $\neg q_0$ for $q \in F$ and $q \notin I$
- **Goal:** p_n for $p \in G$
- **Actions:** For $i = 0, 1, \ldots, n-1$, and each action $a \in A$
  - $a_i \supset p_i$ for $p \in Prec(a)$
  - $a_i \supset p_{i+1}$ for each $p \in Add(a)$
  - $a_i \supset \neg p_{i+1}$ for each $p \in Del(a)$
- **Persistence:** For $i = 0, \ldots, n-1$, and each atom $p \in F$, where $O(p^+)$ and $O(p^-)$ stand for the actions that add and delete p resp.
  - $p_i \wedge \bigwedge_{a \in O(p^-)} \neg a_i \supset p_{i+1}$
  - $\neg p_i \wedge \bigwedge_{a \in O(p^+)} \neg a_i \supset \neg p_{i+1}$
- **Seriality:** For each $i = 0, \ldots, n-1$, if $a \neq a'$, $\neg(a_i \wedge a'_i)$

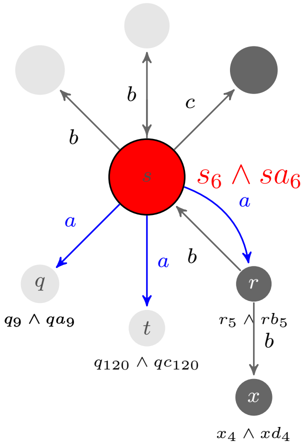
Strong Cyclic Planning as SAT

💡 **Key idea:** label each state with action and distance to goal.



Strong Cyclic Planning as SAT

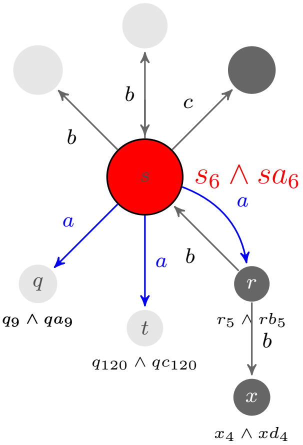
- **Key idea:** label each state with action and distance to goal.
  - **Variables** of SAT encoding (i is not time index!)
    - s_i: min "distance" from s to goal in policy is at most i
    - sa_i: s_i and $\pi(s) = a$



Strong Cyclic Planning as SAT

💡 **Key idea:** label each state with action and distance to goal.

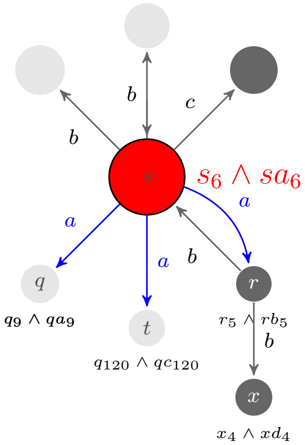
- **Variables** of SAT encoding (i is not time index!)
  - s_i: min "distance" from s to goal in policy is <u>at most</u> i
  - sa_i: s_i and $\pi(s) = a$
- **Formulas** $C(M)$; here $M = \mathcal{S}(P)$ and $max = |S| - 1$:
  1. s_{max} for initial state s_I ; max dist I to goal of length $\leq max$
  2. s_0 for $s \in S_G$ and $\neg s_0$ for $s \notin S_G$



Strong Cyclic Planning as SAT

Key idea: label each state with action and distance to goal.

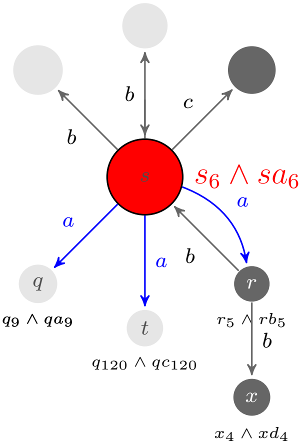
- **Variables** of SAT encoding (i is not time index!)
  - s_i: min "distance" from s to goal in policy is at most i
  - sa_i: s_i and $\pi(s) = a$
- **Formulas** $C(M)$; here $M = \mathcal{S}(P)$ and $max = |S| - 1$:
  1. s_{max} for initial state s_I ; max dist I to goal of length $\leq max$
  2. s_0 for $s \in S_G$ and $\neg s_0$ for $s \notin S_G$
  3. $s_i \supset \bigvee_{a \in A(s)} sa_i$; choose action in s, preserve distance



Strong Cyclic Planning as SAT

💡 **Key idea:** label each state with action and distance to goal.

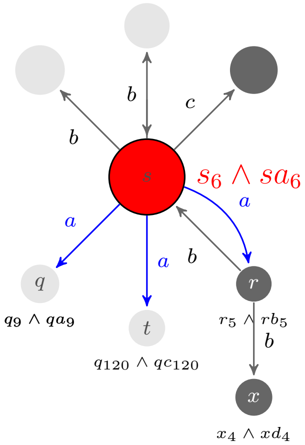
- **Variables** of SAT encoding (i is not time index!)
  - s_i: min "distance" from s to goal in policy is at most i
  - sa_i: s_i and $\pi(s) = a$
- **Formulas** $C(M)$; here $M = \mathcal{S}(P)$ and $max = |S| - 1$:
  1. s_{max} for initial state s_I ; max dist I to goal of length $\leq max$
  2. s_0 for $s \in S_G$ and $\neg s_0$ for $s \notin S_G$
  3. $s_i \supset \bigvee_{a \in A(s)} sa_i$; choose action in s, preserve distance
  4. $sa_i \supset \bigvee_{s' \in f(a,s)} s'_{i-1}$; some successor gets closer to goal



Strong Cyclic Planning as SAT

💡 **Key idea:** label each state with action and distance to goal.

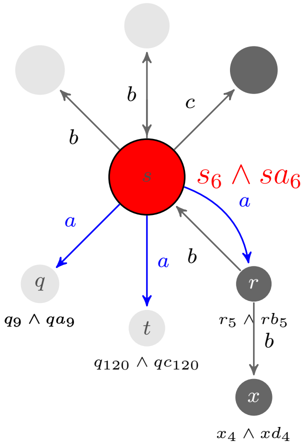
- **Variables** of SAT encoding (i is not time index!)
  - s_i: min "distance" from s to goal in policy is at most i
  - sa_i: s_i and $\pi(s) = a$
- **Formulas** $C(M)$; here $M = \mathcal{S}(P)$ and $max = |S| - 1$:
  1. s_{max} for initial state s_I ; max dist I to goal of length $\leq max$
  2. s_0 for $s \in S_G$ and $\neg s_0$ for $s \notin S_G$
  3. $s_i \supset \bigvee_{a \in A(s)} sa_i$; choose action in s, preserve distance
  4. $sa_i \supset \bigvee_{s' \in f(a,s)} s'_{i-1}$; some successor gets closer to goal
  5. $s_{i-1} \supset s_i$; if distance $\leq i-1$, then $\leq i$
  6. $sa_{i-1} \supset sa_i$; if distance $\leq i-1$, then $\leq i$



Strong Cyclic Planning as SAT

Key idea: label each state with action and distance to goal.

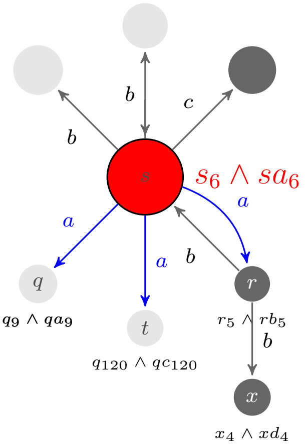
- **Variables** of SAT encoding (i is not time index!)
  - s_i: min "distance" from s to goal in policy is at most i
  - sa_i: s_i and $\pi(s) = a$
- **Formulas** $C(M)$; here $M = \mathcal{S}(P)$ and $max = |S| - 1$:
  1. s_{max} for initial state s_I ; max dist I to goal of length $\leq max$
  2. s_0 for $s \in S_G$ and $\neg s_0$ for $s \notin S_G$
  3. $s_i \supset \bigvee_{a \in A(s)} sa_i$; choose action in s, preserve distance
  4. $sa_i \supset \bigvee_{s' \in f(a,s)} s'_{i-1}$; some successor gets closer to goal
  5. $s_{i-1} \supset s_i$; if distance $\leq i-1$, then $\leq i$
  6. $sa_{i-1} \supset sa_i$; if distance $\leq i-1$, then $\leq i$
  7. $sa_{max} \supset s'_{max}$; if $\pi(s) = a$, all $s' \in f(a,s)$, must reach goal
  8. $sa_{max} \supset \neg sa'_{max}$; if $\pi(s) = a$, then $\pi(s) \neq a'$, $a \neq a'$.



Strong Cyclic Planning as SAT

Key idea: label each state with action and distance to goal.

- **Variables** of SAT encoding (i is not time index!)
  - s_i: min "distance" from s to goal in policy is at most i
  - sa_i: s_i and $\pi(s) = a$
- **Formulas** $C(M)$; here $M = \mathcal{S}(P)$ and $max = |S| - 1$:
  1. s_{max} for initial state s_I ; max dist I to goal of length $\leq max$
  2. s_0 for $s \in S_G$ and $\neg s_0$ for $s \notin S_G$
  3. $s_i \supset \bigvee_{a \in A(s)} sa_i$; choose action in s, preserve distance
  4. $sa_i \supset \bigvee_{s' \in f(a,s)} s'_{i-1}$; some successor gets closer to goal
  5. $s_{i-1} \supset s_i$; if distance $\leq i-1$, then $\leq i$
  6. $sa_{i-1} \supset sa_i$; if distance $\leq i-1$, then $\leq i$
  7. $sa_{max} \supset s'_{max}$; if $\pi(s) = a$, all $s' \in f(a,s)$, must reach goal
  8. $sa_{max} \supset \neg sa'_{max}$; if $\pi(s) = a$, then $\pi(s) \neq a'$, $a \neq a'$.



Theorem

1. *Model M has a **strong-cyclic policy** π iff $C(M)$ is satisfiable.*
2. *If σ satisfies $C(M)$, $\pi(s) = a$ for sa_{max} true in σ is a **strong-cyclic policy** that solves M*

Too large encoding: Towards Compact Polocies

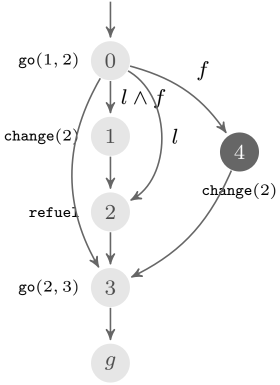
- Encodings are **exhaustive**, all states s represented! ✖
- (Geffner & Geffner 2018) proposed an encoding in SAT computing **compact policies**.
  - *of course, not in worst case*
- Can also be adjusted to compute **strong policies**.
- Can also handle **dual FOND**: fair and unfair actions!
- (Yadav & Sardina 2023): alternative encoding in a Answer Set Programming (ASP):
  - More compact — exploits ASP first-order language.
  - More readable — uses a more declarative style.
  - Integrates regression ideas from PRP.
  - Exploits ASP technology.

Compact Controllers via ASP (Yadav & Sardina 2023)

💡 **Key idea:** devise a finite state controller with n states - (Geffner & Geffner 2018)

Encoding in ASP for FOND problem $P = \langle A, I, G \rangle$:

- `atom(P)`: for each predicate `P` $\in A$.
- `action(A)`: for each action `A` $\in A$. In addition, to define an action's precondition and effects we use the following terms:
  - `prec(A, P)`: atom `P` is in precondition of action `A`.
  - `effect(A, E)`: associates an action with its `E`-th effect (one per oneoff effect).
  - `add(A, E, P)`: `E`-th effect of action `A` adds atom `P`.
  - `del(A, E, P)`: `E`-th effect of action `A` deletes atom `P`.
- `init(P)`: predicate `P` $\in I$ is true in the initial state.
- `goal(P)`: predicate `P` $\in G$ is in the goal condition.



Define Controllers States and Transitions

Solver to decide:

1. `policy(S, A)`: action `A` executed in controller state `S`.
2. `next(S1, E, S2)`: `S2` is the next controler state if the `E`-th effect of prescribed action in `S1` ocurrs.

Define Controllers States and Transitions

Solver to decide:

1. `policy(S, A)`: action `A` executed in controller state `S`.
2. `next(S1, E, S2)`: `S2` is the next controler state if the `E`-th effect of prescribed action in `S1` ocurrs.

```
state(0..k).  % states of the controller
{policy(S, A): action(A)} = 1:- state(S), S != k.
{next(S1, E, S2): state(S2)} = 1 :- policy(S1, A), effect(A, E).
```

1. Defines controller $k+1$ states. State k is goal state.
2. Select one action per controller state (except goal state k).
3. Defines a transition for each action's effect to a next controller state.

Define Controllers States and Transitions

```
1 holds(S, P) :- policy(S, A), prec(A, P).
2 holds(S1, P) :-
3    next(S1, E, S2), holds(S2, P), policy(S1, A), not add(A, E, P).
4 -holds(S2, P) :- next(S1, E, S2), policy(S1, A), del(A, E, P).
5 -holds(0, P) :- atom(P), not init(P).
6 holds(k, P) :- goal(P).
```

1. Preconditions must hold where action is prescribed.
2.
3. Regression: `P` must have been true in the previous controller state.
4. Progression: `P` must be false next if action deleted it.
5. Initial state negative atoms.
6. What must be true at goal controller state `k`

Define Solution Concept: Strong Cyclic

```
reachableG(S):- state(S), S = k.
reachableG(S):- next(S, _, S1), reachableG(S1).
:- not reachableG(S), state(S).
```

1. Goal controller state is reachable from itself.
2. Transitive clousure: Any (previous) controller state connected to a controller state that reaches the goal state, also reaches the controller goal state.
3. **Constraint:** No controller state does not reach the goal state.

Full FOND-ASP Code

```
state(0..k).  % states of the controller
{policy(S, A): action(A)} = 1:- state(S), S != k.
{next(S1, E, S2): state(S2)} = 1 :- policy(S1, A), effect(A, E).

holds(S, P) :- policy(S, A), prec(A, P).
holds(S1, P) :-
   next(S1, E, S2), holds(S2, P), policy(S1, A), not add(A, E, P).
-holds(S2, P) :- next(S1, E, S2), policy(S1, A), del(A, E, P).
-holds(0, P) :- atom(P), not init(P).
holds(k, P) :- goal(P).

reachableG(S):- state(S), S = k.
reachableG(S):- next(S, _, S1), reachableG(S1).
:- not reachableG(S), state(S).
```

If a model is returned, controller defined in predicates `policy/2` and `next/3`.

Experimental Results vs. PRP and FOND-SAT

☀ Better in risky non-determinism domains — first five. PRP better in the rest.

Recap SAT/ASP for FOND Planning

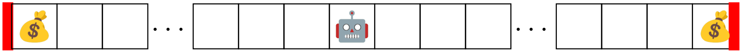
- Declarative elegant solver for FOND planning problems via SAT or ASP.
- Compact controllers: finite state controller with $k+1$ states.
- Increase the size when no solution found, and repeat.
- Faster than classical planning based approaches in domains with meaningful non-determinism ("risky").
- Can incorporate domain control knowledge (e.g., "do not executre a after b").
- Still struggles with large domains with "easy" non-determinism.

Part 1: Non-deterministic Planning

Part 1: Non-deterministic Planning

Can the robot get the money?

Consider an robot in a corridor:

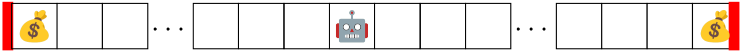


- Robot can move $left$ or $right$ (up to the walls). Unknown size of steps, but ≥ 1
- A price is at some of the end of the corridor.
- Robot doesn't know its cell, but can sense if there is a wall on left/right after moving.

❓ **Can the robot get the money? How to model the setting?**

Can the robot get the money?

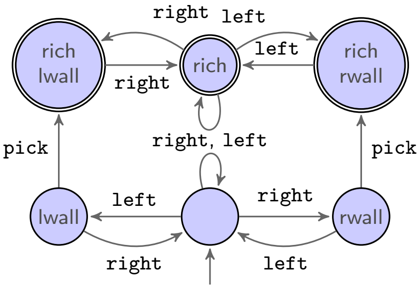
Consider an robot in a corridor:



- Robot can move *left* or *right* (up to the walls). Unknown size of steps, but ≥ 1
- A price is at some of the end of the corridor.
- Robot doesn't know its cell, but can sense if there is a wall on left/right after moving.

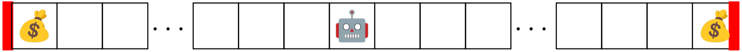
Can the robot get the money? How to model the setting?

```
(define (domain tile)
  (:predicates (leftWall) (rightWall))
  (:action right
        :parameters ()
        :precondition (not rightWall)
        :effect (oneof () (rightWall)))
  (:action left
        :parameters ()
        :precondition (not leftWall)
        :effect (oneof () (leftWall)))
  (:action pick
        :parameters ()
        :precondition (or leftWall rightWall)
        :effect (rich)))
```

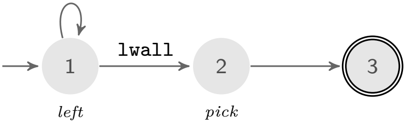


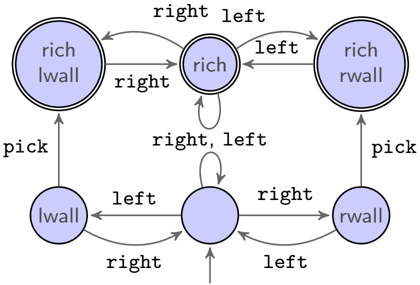
Can the robot get the money?

Consider an robot in a corridor:



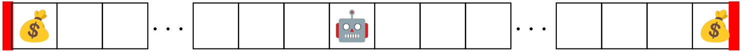
Would this controller work?

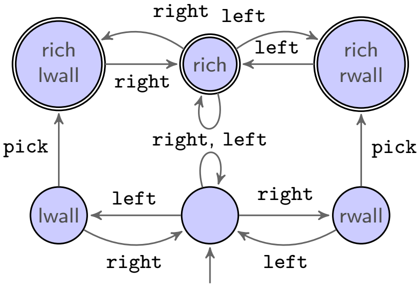




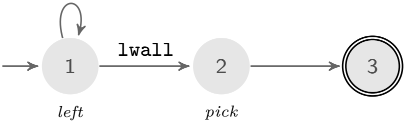
Can the robot get the money?

Consider an robot in a corridor:





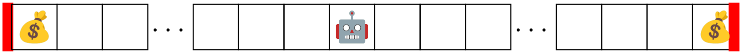
❓ Would this controller work? YES!



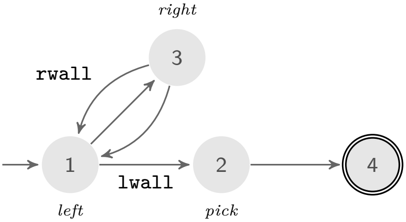
Strong-cyclic policy: Retrying *left* works!

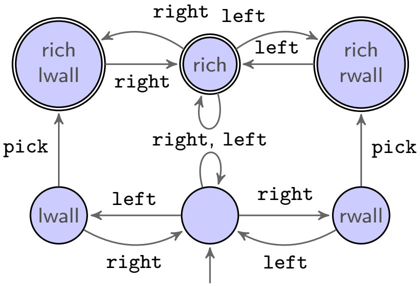
Can the robot get the money?

Consider an robot in a corridor:



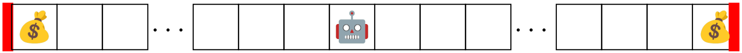
What about this one?



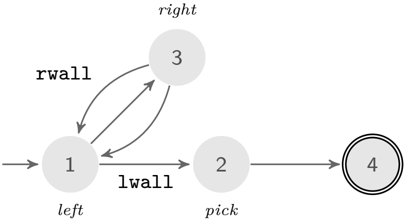


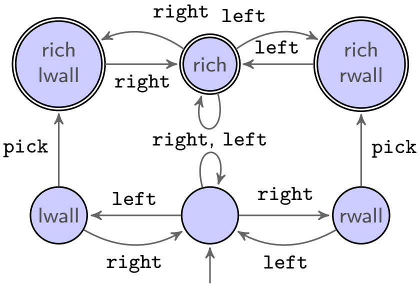
Can the robot get the money?

Consider an robot in a corridor:



What about this one? NO!





How come? It is also a **strong-cyclic policy!**
States where rich true are always reachable..
left action done infinitely many times in initial state

Conditional Fairness (Rodriguez et al. 2021)

- Standard fairness assumption is **not enough**:
  - trying *left* is not sufficient!
  - must not move *right* while trying... 😉
- We need conditional fairness: *left* is fair as long as *right* is not executed.
  - *Same for right: fair provided left is not executed.*
- Standard FOND planners cannot handle this: they assume that **all actions are fair**.
- (Rodriguez et al. 2021)'s FOND$^+$ in ASP can handle:
  - Strong-cyclic policies with conditional fairness.
  - Mixed fairness: some actions are fair, others not.


Journal of Artificial Intelligence Research 74 (2022) 887–916 Submitted 12/2021; published 06/2022

FOND Planning with Explicit Fairness Assumptions

Ivan D. Rodriguez IVANDANIELRA@GMAIL.COM
Blai Bonet BONETBLAI@GMAIL.COM
Universitat Pompeu Fabra, Barcelona, Spain

Sebastian Sardina SEBASTIAN.SARDINA@RMIT.EDU.AU
RMIT University, Melbourne, Australia

Hector Geffner HECTOR.GEFFNER@UPF.EDU
Universitat Pompeu Fabra, Barcelona, Spain
Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain
Linköping University, Linköping, Sweden

Abstract

We consider the problem of reaching a propositional goal condition in fully-observable non-deterministic (FOND) planning under a general class of fairness assumptions that are given explicitly. The fairness assumptions are of the form A/B and say that state trajectories that contain infinite occurrences of an action a from A in a state s and finite occurrence of actions from B, must also contain infinite occurrences of action a in s followed by each one of its possible outcomes. The infinite trajectories that violate this condition are deemed as *unfair*, and the solutions are policies for which *all the fair trajectories reach a goal state.* We show that strong and strong-cyclic FOND planning, as well as QNP planning, a planning model introduced recently for generalized planning, are all special cases of FOND planning with fairness assumptions of this form which can also be combined. FOND$^+$ planning, as this form of planning is called, combines the syntax of FOND planning with some of the versatility of LTL for expressing fairness constraints. A sound and complete FOND$^+$ planner is implemented by reducing FOND$^+$ planning to answer set programs, and its performance is evaluated in comparison with FOND and QNP planners, and LTL synthesis tools. Two other FOND$^+$ planners are introduced as well which are more scalable but are not complete.

(Best Paper Award ICAPS'21)

FOND$^+$

Let's generalize FOND:

FOND$^+$ Problem

A FOND$^+$ problem $P_c = \langle P, C \rangle$ is a FOND problem P extended with a set C of **(conditional) fairness assumptions** of the form A_i/B_i, $i = 1, \ldots, n$ and where each A_i is a set of **non-deterministic actions** in P, and each B_i is a set of actions in P disjoint from A_i.

Meaning of $A/B \in C$: If a state trajectory contains infinite occurrences of actions $a \in A$ in a state s, and *finite* occurrences of actions from B, then s must be immediately followed by each $s' \in F(\pi(s), s)$ an infinite number of times.

- ✏ *if $left$ is executed infinitely many times in s, but $right$ is executed, say, 10 times, then eventually we will see the left wall.*

FOND Solutions as FOND$^+$ Solutions

FOND$^+$ Problem

A FOND$^+$ problem $P_c = \langle P, C \rangle$ is a FOND problem P extended with a set C of **(conditional) fairness assumptions** of the form A_i/B_i, $i = 1, \ldots, n$ and where each A_i is a set of **non-deterministic actions** in P, and each B_i is a set of actions in P disjoint from A_i.

🔅 Strong and strong cyclic planning all have solutions defined by the implicit fairness assumptions particular to each one of them.

FOND Solutions as FOND$^+$ Solutions

FOND$^+$ Problem

A FOND$^+$ problem $P_c = \langle P, C \rangle$ is a FOND problem P extended with a set C of **(conditional) fairness assumptions** of the form A_i/B_i, $i = 1, \ldots, n$ and where each A_i is a set of **non-deterministic actions** in P, and each B_i is a set of actions in P disjoint from A_i.

🔅 Strong and strong cyclic planning all have solutions defined by the implicit fairness assumptions particular to each one of them.

Theorem

The ***strong-cyclic solutions*** *of a FOND problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \{A/\emptyset\} \rangle$, where A is the set of all the non-deterministic actions in P.*

FOND Solutions as FOND$^+$ Solutions

FOND$^+$ Problem

A FOND$^+$ problem $P_c = \langle P, C \rangle$ is a FOND problem P extended with a set C of **(conditional) fairness assumptions** of the form A_i/B_i, $i = 1, \dots, n$ and where each A_i is a set of **non-deterministic actions** in P, and each B_i is a set of actions in P disjoint from A_i.

🔅 Strong and strong cyclic planning all have solutions defined by the implicit fairness assumptions particular to each one of them.

Theorem

The ***strong-cyclic solutions*** *of a FOND problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \{A/\emptyset\} \rangle$, where A is the set of all the non-deterministic actions in P.*

Theorem

The ***strong solutions*** *of a FOND problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \emptyset \rangle$.*

FOND$^+$-ASP: An ASP-based Planner

```
% policy, edges, and connectedness
{ pi(S,A) : ACTION(A) } = 1 :- STATE(S), not GOAL(S).
successor(S,T) :- pi(S,A), TRANSITION(S,A,T).

connected(S,T) :- successor(S,T).
connected(S,T) :- connected(S,X), successor(X,T), S != X.

blocked(S,T) :- STATE(S), STATE(T), not connected(S,T).
blocked(S,T) :- connected(S,T), terminate(S).
blocked(S,T) :- connected(S,T), terminate(T).
blocked(S,T) :- connected(S,T),
                blocked(X,T) : successor(S,X), connected(X,T).

fair(S) :- pi(S,A), ASET(I,A), blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).

% terminating states
terminate(S) :- GOAL(S).
terminate(S) :- fair(S), successor(S,T), terminate(T).
terminate(S) :- not fair(S), successor(S,_), terminate(T) : successor(S,T).

% reachable states must terminate
:- reachable(S), not terminate(S).
reachable(S) :- INITIAL(S).
reachable(S) :- reachable(X), not GOAL(X), successor(X,S).
```

```
STATE(S)
INITIAL(S)
GOAL(S)
ACTION(A)
TRANSITION(S,A,T)
ASET(A,I)
BSET(B,I)
```

just 24 lines!

figs/fondplus.lp

FOND$^+$-ASP: Graphical Intuition...

figure of a transition system, with two states looping, the first selects action A and the second B. draw successors of each..

FOND^{+}-ASP: Solution Policy

```
1  % policy, edges, and connectedness
2  { pi(S,A) : ACTION(A) } = 1 :- STATE(S), not GOAL(S).
3  successor(S,T) :- pi(S,A), TRANSITION(S,A,T).
4
5  % reachable states must terminate
6  :- reachable(S), not terminate(S).
7  reachable(S) :- INITIAL(S).
8  reachable(S) :- reachable(X), not GOAL(X), successor(X,S).
```

2 Select an action per domain state.

3 Edges are transitions of the action selected.

FOND$^+$-ASP: Solution Policy

```
1 % policy, edges, and connectedness
2 { pi(S,A) : ACTION(A) } = 1 :- STATE(S), not GOAL(S).
3 successor(S,T) :- pi(S,A), TRANSITION(S,A,T).
4
5 % reachable states must terminate
6 :- reachable(S), not terminate(S).
7 reachable(S) :- INITIAL(S).
8 reachable(S) :- reachable(X), not GOAL(X), successor(X,S).
```

2 Select an action per domain state.

3 Edges are transitions of the action selected.

6 **Constraint:** every reachable state via the policy needs to eventually terminate.

7-8 Define reachable states via the policy.

FOND^{+}-ASP: State Termination

Defines when a state will eventually lead to termination and not get “sucked” in a loop..

```
1  % terminating states
2  terminate(S) :- GOAL(S).
3  terminate(S) :- fair(S), successor(S,T), terminate(T).
4  terminate(S) :- not fair(S), successor(S,_),
5                  terminate(T) : successor(S,T).
```

2 If the state is a goal state.

3 If state will *behave **fairly*** (wrt effects of prescribed action) and one successor state will terminate.

4 If state may *not* behave **fairly**, and all successors will terminate.

FOND$^+$-ASP: Fairness

```
1   connected(S,T) :- successor(S,T).
2   connected(S,T) :- connected(S,X), successor(X,T), S != X.
3
4   % terminating states
5   terminate(S) :- GOAL(S).
6   terminate(S) :- fair(S), successor(S,T), terminate(T).
7   terminate(S) :- not fair(S), successor(S,_),
8                   terminate(T) : successor(S,T).
9
10  fair(S) :- pi(S,A), ASET(I,A),
11             blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).
```

1-2 States connected by the policy.

4-7 Every path from `S` to `T` will terminate somewhere.

10 Fair if any loop that includes actions in a fairness pair A/B (e.g., $left$ and $right$), will terminate somewhere else.

FOND$^+$-ASP: Strong Cyclic

Theorem

*The **strong-cyclic solutions** of a FOND problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \{A/\emptyset\}\rangle$, where A is the set of all the non-deterministic actions in P.*

```
1  % terminating states
2  terminate(S) :- GOAL(S).
3  terminate(S) :- fair(S), successor(S,T), terminate(T).
4  terminate(S) :- not fair(S), successor(S,_),
5                  terminate(T) : successor(S,T).
6
7  fair(S) :- pi(S,A), ASET(I,A),
8             blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).
```

always false

Line 3 always applies!

FOND^{+}-ASP: Strong Cyclic

Theorem

*The **strong-cyclic solutions** of a FOND problem P are the solutions of the FOND^{+} problem $P_c = \langle P, \{A/\emptyset\}\rangle$, where A is the set of all the non-deterministic actions in P.*

```
1  % terminating states
2  terminate(S) :- GOAL(S).
3  terminate(S) :- fair(S), successor(S,T), terminate(T).
4  terminate(S) :- not fair(S), successor(S,_),
5                  terminate(T) : successor(S,T).
6
7  fair(S) :- pi(S,A), ASET(I,A),
8             blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).
```

always true

always false

Line 3 always applies!

FOND$^+$-ASP: Strong Cyclic

Theorem

*The **strong-cyclic solutions** of a FOND problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \{A/\emptyset\}\rangle$, where A is the set of all the non-deterministic actions in P.*

```
1  % terminating states
2  terminate(S) :- GOAL(S).
3  terminate(S) :- fair(S), successor(S,T), terminate(T).
4  terminate(S) :- not fair(S), successor(S,_),
5                  terminate(T) : successor(S,T).
6
7  fair(S) :- pi(S,A), ASET(I,A),
8             blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).
```

always applies

always true

always false

Line 3 always applies!

FOND^{+}-ASP: Strong

Theorem

*The **strong solutions** of a FOND problem P are the solutionsof the FOND^{+} problem $P_c = \langle P, \emptyset \rangle$.*

```
1  % terminating states
2  terminate(S) :- GOAL(S).
3  terminate(S) :- fair(S), successor(S,T), terminate(T).
4  terminate(S) :- not fair(S), successor(S,_),
5                  terminate(T) : successor(S,T).
6
7  fair(S) :- pi(S,A), ASET(I,A);
8             blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).
```

always false

Line 4 always applies!

FOND$^+$-ASP: Strong

Theorem

The ***strong solutions*** *of a FOND problem P are the solutionsof the FOND$^+$ problem $P_c = \langle P, \emptyset \rangle$.*

```
1  % terminating states
2  terminate(S) :- GOAL(S).
3  terminate(S) :- fair(S), successor(S,T), terminate(T).
4  terminate(S) :- not fair(S), successor(S,_),
5                  terminate(T) : successor(S,T).
6
7  fair(S) :- pi(S,A), ASET(I,A),
8             blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).
```

always false

always false

☼ Line 4 always applies!

FOND$^+$-ASP: Strong

Theorem

*The **strong solutions** of a FOND problem P are the solutionsof the FOND$^+$ problem $P_c = \langle P, \emptyset \rangle$.*

```
1   % terminating states
2   terminate(S) :- GOAL(S).
3   terminate(S) :- fair(S), successor(S,T),
4   terminate(S) :- not fair(S), successor(S,_),
5                   terminate(T) : successor(S,T).
6
7   fair(S) :- pi(S,A), ASET(I,A),
8             blocked(X,S) : pi(X,B), BSET(I,B), not blocked(S,X).
```

always applies

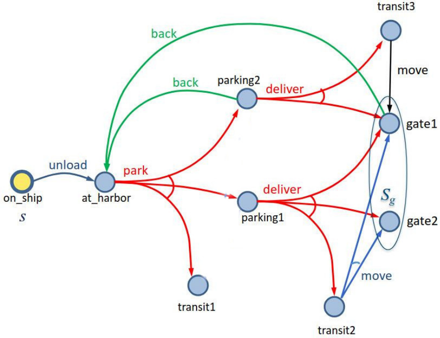
always false

always false

Line 4 always applies!

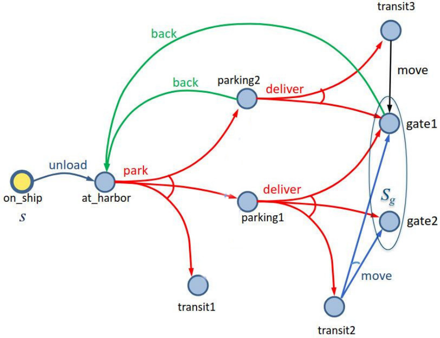
Discussion

- We tested FOND$^+$-ASP experimentally:
  - Only planner that can solve FOND+ problems!
  - Performs better than FOND-SAT and LTL synthesis tool STRIX.
  - PRP scales up better for FOND tasks.
  - **Limitation:** state space grounding.



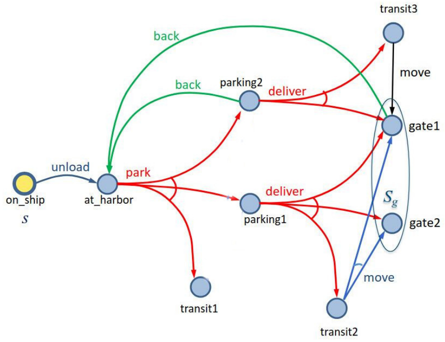
Discussion

- We tested FOND$^+$-ASP experimentally:
  - Only planner that can solve FOND+ problems!
  - Performs better than FOND-SAT and LTL synthesis tool STRIX.
  - PRP scales up better for FOND tasks.
  - **Limitation:** state space grounding.
- FOND = simple extension of classical planning
  - *Just add* `oneof` *in effects!*
- But brings radical changes:
  - Complexity up to EXPTIME-complete.
  - Builds plans with loops!
  - Can model scenarios with "re-tries"
  - Can deal with adversarial domains.



Discussion

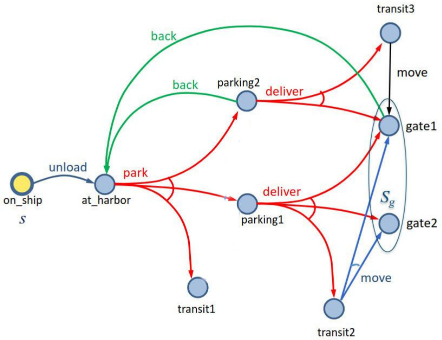
- We tested FOND$^+$-ASP experimentally:
  - Only planner that can solve FOND+ problems!
  - Performs better than FOND-SAT and LTL synthesis tool STRIX.
  - PRP scales up better for FOND tasks.
  - **Limitation:** state space grounding.
- FOND = simple extension of classical planning
  - *Just add* `oneof` *in effects!*
- But brings radical changes:
  - Complexity up to EXPTIME-complete.
  - Builds plans with loops!
  - Can model scenarios with "re-tries"
  - Can deal with adversarial domains.
- FOND$^+$ and domains with "qualitative" numbers?
  - *e.g., distance to the wall*



Discussion

- We tested FOND$^+$-ASP experimentally:
  - Only planner that can solve FOND+ problems!
  - Performs better than FOND-SAT and LTL synthesis tool STRIX.
  - PRP scales up better for FOND tasks.
  - **Limitation:** state space grounding.
- FOND = simple extension of classical planning
  - *Just add* `oneof` *in effects!*
- But brings radical changes:
  - Complexity up to EXPTIME-complete.
  - Builds plans with loops!
  - Can model scenarios with "re-tries"
  - Can deal with adversarial domains.
- FOND$^+$ and domains with "qualitative" numbers?
  - *e.g., distance to the wall*

Qualitative Numeric Planning (QNP)



Que vimos? 👀

1. **Busqueda** as a general problem solving method:
   - Representación: state model (a graph!).
   - Uninformed methods: BrFS, DFS, IDS, UCS.
   - Informed methods: A* and heuristics.
   - Heuristics as problem relaxation.

Que vimos? 👀

1. **Busqueda** as a general problem solving method:
   - Representación: state model (a graph!).
   - Uninformed methods: BrFS, DFS, IDS, UCS.
   - Informed methods: A* and heuristics.
   - Heuristics as problem relaxation.
2. **Classical Planning** = AI Search + AI KR
   - Model-based approach to autonomous behavior.
   - Languages: STRIP and PDDL.
   - Heuristic extraction by relaxing the representation.
   - Delete-relaxation heuristic: h^+
   - Approximations: h_{add}, $h_{\max}$, h_{FF}.
   - Planning graphs.

Que vimos? 👀

1. **Busqueda** as a general problem solving method:
   - Representación: state model (a graph!).
   - Uninformed methods: BrFS, DFS, IDS, UCS.
   - Informed methods: A* and heuristics.
   - Heuristics as problem relaxation.
2. **Classical Planning** = AI Search + AI KR
   - Model-based approach to autonomous behavior.
   - Languages: STRIP and PDDL.
   - Heuristic extraction by relaxing the representation.
   - Delete-relaxation heuristic: h^+
   - Approximations: h_{add}, $h_{\max}$, h_{FF}.
   - Planning graphs.
3. **FOND Planning**: Non-determinism
   - Non-deterministic state models (no probabilities!)
   - PDDL with `one-of` effects + Policies.
   - Solution concepts: weak, strong, strong-cyclic.
   - Fairness assumption on environment.
   - Computing policies.

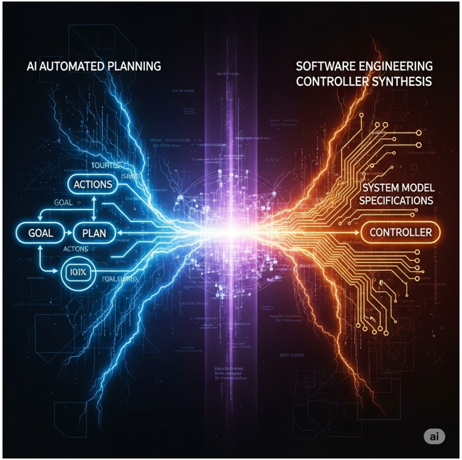
AI Planning and Control Synthesis in SE 🤝

- What if we want to plan for **more complex goals**?
  ➠ **Elevator controller:** every passenger floor requests needs to be *eventually* fulfilled, but **never** have more than 10 passengers on board.

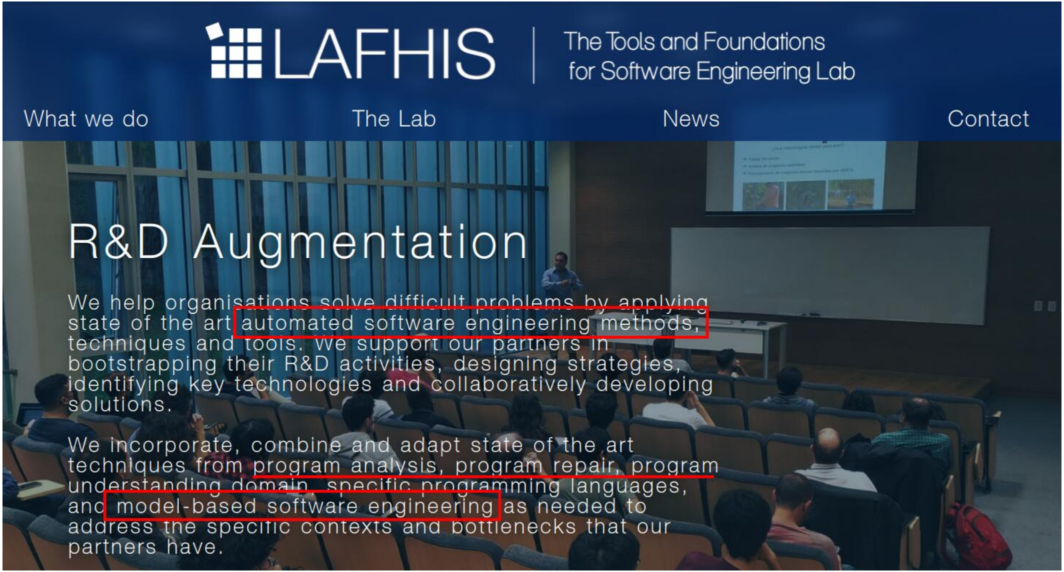
- **Event-driven systems**: some events cannot be planned/controlled (e.g., user aborts transaction)

- **Infinite behavior:** continuous operation, never stop.
  ➠ What are the goals if we never finish? Infinite games vs. finite games

- **Compositional planning/synthesis**: software components described separately
  ➠ Plan on different PDDLs and the combine.



LaFHIS - Laboratory on Fundamentals and Tools for Software Engineering



LAFHIS | The Tools and Foundations for Software Engineering Lab

What we do The Lab News Contact

R&D Augmentation

We help organisations solve difficult problems by applying state of the art automated software engineering methods, techniques and tools. We support our partners in bootstrapping their R&D activities, designing strategies, identifying key technologies and collaboratively developing solutions.

We incorporate, combine and adapt state of the art techniques from program analysis, program repair, program understanding domain, specific programming languages, and model-based software engineering as needed to address the specific contexts and bottlenecks that our partners have.

Contact sebastian.sardina@rmit.edu.au - https://ssardina.github.io/