Feature-based Generalized Policies and Guarantees

IJCAI/GenPlan 2022

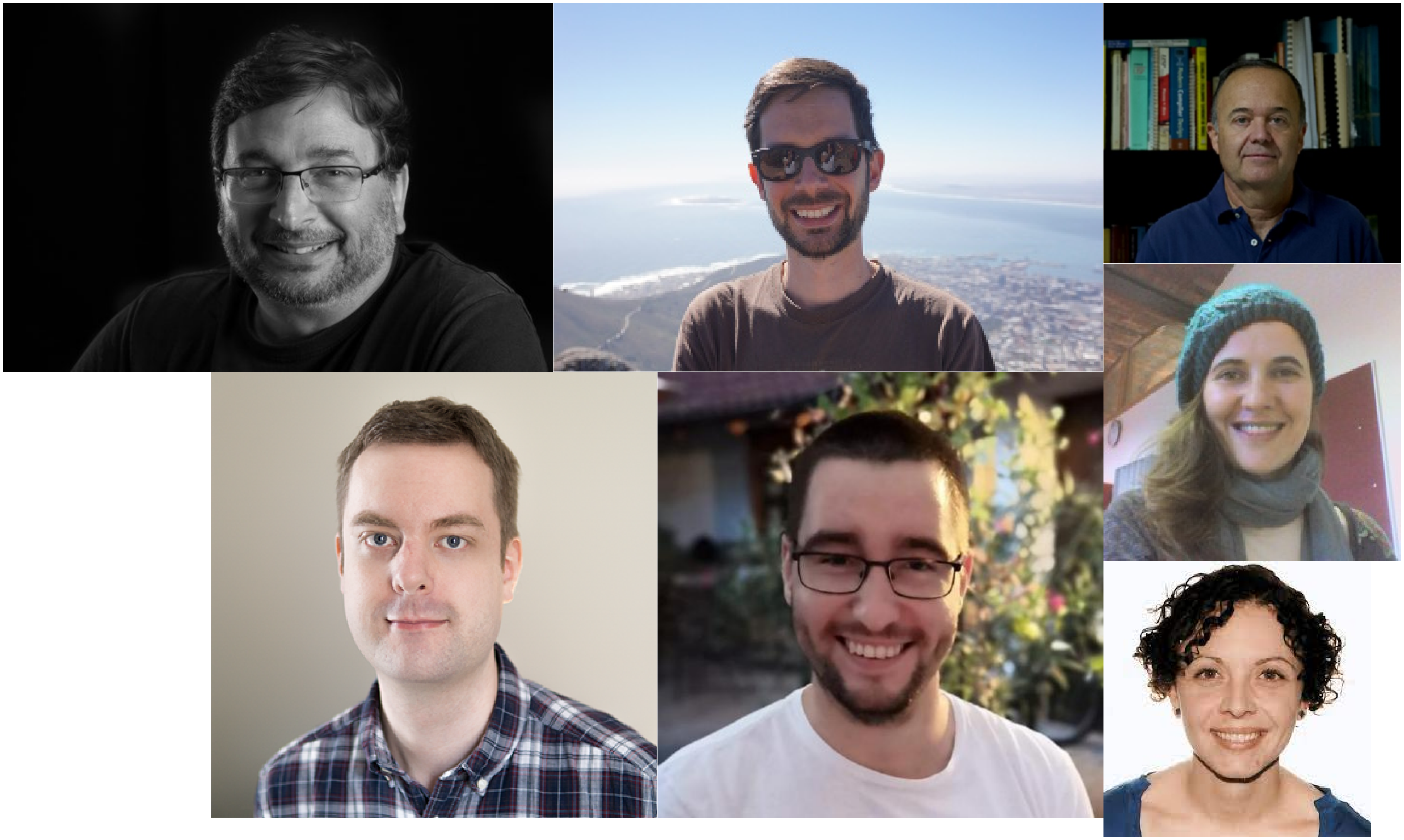
Blai Bonet
Universitat Pompeu Fabra, Barcelona

With inputs from Hector Geffner

Universitat
Pompeu Fabra
Barcelona

European Research Council
Established by the European Commission

Collaborators



Introduction

- In recent years, generalized planning has become important in planning and DRL [Hu and De Giacomo, 2011; Srivastava *et al.*, 2011; Toyer *et al.*, 2018; Garg *et al.*, 2021; Chevalier-Boisvert *et al.*, 2019; etc]

- In the "logical setting", a successful approach expresses general policies with rules over state features where
  - ▷ features provide the necessary abstraction over a class of planning instances
  - ▷ rules tell which transitions to take at non-goal states

- Rules can also be used to express **solution strategies** based on **subgoals** (called **plan sketches**) that are guaranteed to be executable in polynomial time [B. and Geffner, 2019a; Drexeler *et al.*, 2021, 2022]

- General policies and sketches can be learned from traces

- Correctness of learned policies has also been investigated

Outline

- Part I: Classical planning and generalized planning
  - ▷ Model and language
  - ▷ First-order STRIPS
  - ▷ Generalized planning
- Part II: General policies
  - ▷ Language and semantics
  - ▷ Features: Description Logics and FOL
  - ▷ Learning general policies
- Part III: Formal guarantees for generalization
  - ▷ Showing that a general policy solves a class of problems
  - ▷ What is a guarantee?
  - ▷ Guarantees as certificates over reachable states
  - ▷ Synthesis of certificates
- Wrap up

Part I:

Classical Planning and Generalized Planning

State Model for Classical AI Planning

A (classical) **state model** is tuple $\mathcal{S} = \langle S, s_0, S_G, Act, A, f, c \rangle$:

- finite and discrete **state space** S
- a known **initial state** $s_0 \in S$
- a set $S_G \subseteq S$ of **goal states**
- **actions** $A(s) \subseteq Act$ **applicable** in each $s \in S$
- a **deterministic state-transition function** $s' = f(a, s)$ for $a \in A(s)$
- positive **action costs** $c(a, s)$, assumed 1 by default

A **solution** to the model or **plan** is a sequence of applicable actions $a_0, \ldots, a_n$ that maps s_0 into S_G

i.e. there must be state sequence $s_0, \ldots, s_{n+1}$ such that $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$, and $s_{n+1} \in S_G$

Language for Classical Planning: (Grounded) STRIPS

- A (grounded) **problem** in STRIPS is tuple $P = \langle F, O, I, G \rangle$:
  - ▷ F is set of (ground) **atoms**
  - ▷ O is set of (ground) **actions**
  - ▷ $I \subseteq F$ stands for **initial situation**
  - ▷ $G \subseteq F$ stands for **goal situation**

- Actions $o \in O$ **represented** by
  - ▷ **Add** list $Add(o) \subseteq F$
  - ▷ **Delete** list $Del(o) \subseteq F$
  - ▷ **Precondition** list $Pre(o) \subseteq F$

A **problem** P in STRIPS defines **state model** $S(P)$ in compact form . . .

From Language to Models

STRIPS problem $P = \langle F, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- states $s \in S$ are collections of atoms from F
- initial state s_0 is I
- goal states s_G are such that $G \subseteq s_G$
- actions a in $A(s)$ are ops in O s.t. $Prec(a) \subseteq s$
- next state is $s' = [s \setminus Del(a)] \cup Add(a)$
- action costs $c(a, s)$ are all 1

Common approach for solving P is using **path-finding/heuristic search** algorithms over **graph** defined by $\mathcal{S}(P)$ where nodes are states s, and edges (s, s') are state transitions caused by an action a; i.e., $s' = f(a, s)$ and $a \in A(s)$

The **source** node is the initial state s_0, and the **targets** are the goal states s_G

Language for Generalized Planning: First-Order STRIPS

Problems specified as **instances** $P = \langle D, I \rangle$ of **general** planning domain:

- **Domain** D specified in terms of **action schemas** and **predicates**
- **Instance** is $P = \langle D, I \rangle$ where I details **objects**, **init**, **goal**

Distinction between **general** domain D and **specific** instance $P = \langle D, I \rangle$ important for **reusing** action models, and also for **learning** them

Generalized planning deals with collection of problems that **share domain** D

Example: 2-Gripper Problem $P = \langle D, I \rangle$ in PDDL

```
(define (domain gripper)
   (:requirements :typing)
   (:types room ball gripper)
   (:constants left right - gripper)
   (:predicates (at-robot ?r - room) (at ?b - ball ?r - room) (free ?g - gripper)
       (carry ?o - ball ?g - gripper))
   (:action move
       :parameters   (?from ?to - room)
       :precondition (at-robot ?from)
       :effect       (and (at-robot ?to) (not (at-robot ?from))))
   (:action pick
       :parameters   (?obj - ball ?room - room ?gripper - gripper)
       :precondition (and (at ?obj ?room) (at-robot ?room) (free ?gripper))
       :effect       (and (carry ?obj ?gripper) (not (at ?obj ?room)) (not (free ?gripper))))
   (:action drop
       :parameters   (?obj - ball ?room - room ?gripper - gripper)
       :precondition (and (carry ?obj ?gripper) (at-robot ?room))
       :effect       (and (at ?obj ?room) (free ?gripper) (not (carry ?obj ?gripper)))))

(define (problem gripper2)
    (:domain gripper)
    (:objects roomA roomB - room Ball1 Ball2 - ball)
    (:init (at-robot roomA) (free left) (free right) (at Ball1 roomA) (at Ball2 roomA))
    (:goal (and (at Ball1 roomB) (at Ball2 roomB))))
```

Generalized Planning

Generalized task is collection $\mathcal{Q}$ of ground instances $P_i = \langle D, I_i \rangle$ that share a common first-order STRIPS domain D together with a **goal description**

For example, all $\mathcal{Q}_{gripper}$ is the task of all gripper instances with **any number of balls and any number of rooms,** with the goal of having all balls in "room B"

This is an infinite class of instances

Instances assumed to be **"well-formed"**; e.g., for all reachable states in all P_i in $\mathcal{Q}$, each ball is in **exactly one** position, and the agent is in **exactly one room**

Part II:

General Policies

General Policies

- **General policy** represents strategy for solving **multiple** instances **reactively**; i.e., without having to search or plan
  - ▷ E.g., policy for achieving $on(x, y)$ for **any** $\#$ of blocks, **any** configuration
- What are good **languages** for expressing such policies?
- Number of works have addressed the problem [Khardon 1999; Martin and G., 2004; Fern *et al.*, 2006; Srivastava *et al.*, 2011; Hu and De Giacomo, 2011]
- **Obstacle:** set of (ground) actions change from instance to instance with objects

A Language for General Policies [B. and Geffner, 2018]

- **General policies** are given by **rules** $C \mapsto E$ over set Φ of **features**

- **Features** f are state functions that have well-defined value $f(s)$ on every reachable state of any instance of the domain
  - ▷ **Boolean** features p: $p(s)$ is true or false
  - ▷ **Numerical** features n: $n(s)$ is non-negative integer

Computation of feature values assumed to be "cheap": features assumed to have **linear** number of values at most, computable in **linear** time (in #atoms in $|P|$)

Example: General Policy for $clear(\mathbf{X})$

- **Features** $\Phi = \{H, n\}$: 'holding a block' and 'number of blocks above x'
- **Policy** π for class $\mathcal{Q}$ of Block problems with goal $clear(x)$ given by two rules:

$$\{\neg H, n > 0\} \mapsto \{H, n\!\downarrow\} \quad ; \quad \{H, n > 0\} \mapsto \{\neg H\}$$

Meaning:

– if $\neg H$ & $n > 0$, move to successor state where H holds and n **decreases**

– if H & $n > 0$, move to successor state where $\neg H$ holds, n **doesn't change**

Language and Semantics of General Policies: Definitions

- **Policy rules** $C \mapsto E$ over set Φ of Boolean and numerical **features** p, n:
  - ▷ *Boolean conditions* in C: p, $\neg p$, $n = 0$, $n > 0$
  - ▷ *qualitative effects* in E: p, $\neg p$, $p?$, $n{\downarrow}$, $n{\uparrow}$, $n?$
- **State transition** (s, s') **satisfies** rule $C \mapsto E$ if
  - ▷ $f(s)$ makes body C true
  - ▷ change from $f(s)$ to $f(s')$ satisfies E
- A **policy** π for class $\mathcal{Q}$ of problems P is given by set of policy rules $C \mapsto E$
  - ▷ *Transition* (s, s') in P compatible with π if (s, s') satisfies a policy rule
  - ▷ *Trajectory* $s_0, s_1, \ldots$ compatible if s_0 of P and transitions compatible with π
- π **solves** P if all max trajectories compatible with π reach goal of P
- π **solves** collection of problems $\mathcal{Q}$ if it solves each $P \in \mathcal{Q}$

Example: Delivery

- Pick packages spread in $n \times m$ grid, one by one, to target location
- **Features** $\Phi = \{H, p, t, n\}$: hold, dist. to nearest pkg & target, # undelivered
- Policy π that solves class $\mathcal{Q}_D$: **any** # of pkgs and distribution, **any** grid size

$\{\neg H, p > 0\} \mapsto \{p\!\downarrow, t?\}$	go to nearest package
$\{\neg H, p = 0\} \mapsto \{H, p?\}$	pick it up
$\{H, t > 0\} \mapsto \{t\!\downarrow, p?\}$	go to target cell
$\{H, t = 0\} \mapsto \{\neg H, n\!\downarrow, p?\}$	drop package

Features: Desc. Logics [B. et al., 2019a; Francès et al., 2021]

- **Description logic grammar** allows generation of **concepts** and **roles** from **domain predicates**
- Pool $\mathcal{F}$ obtained from concepts of complexity **bounded by parameter**
- Complexity of concept/role given by **size of its syntax tree**
- Denotation of concept C in state s is **subset** $C(s)$ **of objects**
- Each concept C defines num and Bool features $n_C(s) = |C(s)|$; $p_C(s){=}\top$ iff $|C(s)| > 0$
- Grammar:
  - ▷ Primitive: C_p given by unary predicates p and unary "goal predicates" p_G
  - ▷ Universal: C_u contains all objects
  - ▷ Nominals: $C_a = \{a\}$ for constants/parameter a
  - ▷ Negation: $\neg C$ contains $C_u \setminus C$
  - ▷ Intersection: $C \sqcap C'$
  - ▷ Quantified: $\exists R.C{=}\{x : \exists y[R(x,y) \wedge C(y)]\}$ and $\forall R.C{=}\{x : \forall y[R(x,y) \wedge C(y)]\}$
  - ▷ Roles (for binary predicate p): R_p, R_p^{-1}, R_p^+, and $[R_p^{-1}]^+$
- **Additional distance features:** $dist(C_1, R, C_2)$ for concepts C_1 and C_2 and role R that evaluates to d in state s iff minimum R-distance between object in C_1 to object in C_2 is d

First-Order Features [B. et al., 2019b]

- For STRIPS domain D, **signature** $\sigma(D)$ comprises of domain predicates in D plus predicates p^* and p^+ for binary predicates p in D

- Predicates p^* and p^+ added because not definable in FOL

- **FO concept:** $C = \{\bar{o} : \Psi(\bar{o})\}$ defined by FO formula Ψ over $\sigma(D)$

- Denotation $C(s)$ at state s is $C(s) = \{\bar{o} : s \vDash \Psi(\bar{o})\}$; i.e., denotation of C may contain object tuples

- FO feature f given by FO concept C with value $f(s) = |C(s)|$

- All DL features **except distance features** are FO features, but there are FO features that aren't DL features

Learning General Policies (and Sketches)

- General policies learned from small sample of traces $\mathcal{T}$ and DL feature pool $\mathcal{F}$
- Learning task formulated as **combinatorial optimization problem** [B. *et al.*, 2019a; Francès *et al.*, 2021]
- Learned policy then **verified** empirically over test instances of bigger size (and latter verified by "hand" that policies are indeed general)
- Policy sketches also learned using combinatorial optimization [Drexeler *et al.*, 2022]
- **Deep learning** approach using GNNs doesn't need pool $\mathcal{F}$ [Ståhlberg *et al.* 2022a,b]

Part III:

Formal Guarantees for Generalization

Proving that General Policy Solves Class of Instances $\mathcal{Q}$

How to **prove** that this policy π achieves $clear(x)$ in all Block problems?

$$\{\neg H, n > 0\} \mapsto \{H, n\downarrow\} \quad ; \quad \{H, n > 0\} \mapsto \{\neg H\}$$

- **Soundness:** policy π applies in every **non-goal** state s
  - ▷ for any such s, there is transition (s, s') compatible with π
- **Acyclicity:** no sequence of transitions (s_i, s_{i+1}) compatible with π **cycles**

Theorem: If π is **sound and acyclic** in $\mathcal{Q}$, π **solves** $\mathcal{Q}$

Acyclicity, Termination, and QNPs

- **Termination:** structural criterion that ensures policy is **acyclic** over **any** domain
- A policy π is **terminating** if for all **infinite** trajectories $s_0, \ldots, s_i, \ldots$ compatible with π, there is a **numerical feature** n such that:
  - ▷ n is **decremented** in some recurrent transition (s, s'); i.e., $n(s') < n(s)$
  - ▷ n is **not incremented** in any recurrent transition (s, s'); i.e., $n(s') \not> n(s)$
- Every such trajectory deemed **impossible** or **unfair** (n can't decrement below 0), thus if π terminates, π-trajectories **terminate**
- **Termination** notion is from **QNPs**; verifiable in time $O(2^{|\Phi|})$ by SIEVE algorithm [Srivastava *et al.*, 2011], where Φ is set of features involved in the policy
- Also characterized logically using fairness assumptions [Rodriguez *et al.* 2021]

Acyclicity for Policies over First-Order Features

- If all features in policy π are FO features, termination condition can be weakened

- π-trajectory $s_0, \ldots, s_i, \ldots$ on **STRIPS instance** P terminates if for some numerical feature f:
  - ▷ f is decreased (resp. increased) an **infinite number** of times
  - ▷ f is increased (resp. decreased) a **finite number** of times

- Reason is that number of object tuples in any STRIPS instance is **finite**

- QNP termination is stronger since
  - ▷ QNP variables are not necessarily bounded from above
  - ▷ QNP variables are not necessarily integer-valued

Soundness

- The other property needed for showing that π solves $\mathcal{Q}$: for non-goal reachable states s, there is transition (s, s') that is **compatible** with π

- For example, how do we know the following policy is sound for Blocks?

$$\{\neg H, n > 0\} \mapsto \{H, n\downarrow\} \qquad ; \qquad \{H, n > 0\} \mapsto \{\neg H\}$$

  E.g., suppose the hand is empty and there are blocks above x in state s:

  ▷ Is there a transition (s, s') compatible with the effect $\{H, n\downarrow\}$?

  ▷ **Yes:** any tower in **"well-formed"** state for Blocks, **ends up in a clear block**

- Soundness isn't structural property of π; **it depends on reachable states!**

What's a (Formal) Guarantee?

- It is **certificate** $\mathcal{C}_\pi$ that shows that π solves $\mathcal{Q}$:
  - ▷ all trajectories for P in $\mathcal{Q}$ compatible with π are **acyclic**
  - ▷ for any non-goal reachable state s in $\mathcal{Q}$, there is (s, s') **compatible** with π

- Acyclicity established from π alone (structurally)

- Soundness must be established using knowledge about instances in $\mathcal{Q}$

IDEA: Rather than formalize well-formedness of states and then reason, better is to **characterize subclass** of instances on which π is **guaranteed to be sound**

Yields principled and clear path for automatically obtaining guarantees

Guarantees as Invariants over Reachable States

- Often, general plan π guaranteed to succeed when certain properties (invariants) hold on set of reachable states
  - ▷ E.g. for $clear$(X), it's enough that for all reachable states, the tower containing X ends up in a clear block, and no two blocks are on common block

- If features in policy π are **first-order,** one can obtain invariants automatically by requiring reasonable properties:
  - ▷ Decrement $n\downarrow$ across (s, s') **shrinks denotation** of $n(s)$ (i.e., $n(s') \subsetneq n(s)$)
  - ▷ Increment $n\uparrow$ across (s, s') **enlarges denotation** of $n(s)$ (i.e., $n(s) \subsetneq n(s)$)

Certificates: Language and Semantics [B. et al., 2019b]

For policy π given by rules $\{r : C_r \mapsto E_r\}$ and STRIPS domain D:

- We aim at **certificate** $\mathcal{C}_\pi = \{\Phi_r : r \in \pi\}$ where $\Phi_r = \exists\bar{z}\big(\bigvee_{a \in D} \Psi^a_r(\bar{z})\big)$:
  - ▷ a is action schema in domain D
  - ▷ $\bar{z}$ is arguments of a, existentially quantified on objects
  - ▷ if $s \vDash C_r \wedge \Psi^a_r(\bar{o})$, then (s, s') is compatible with E_r for $s' = res(s, a(\bar{o}))$
  - ▷ That is, $C_r \wedge \Psi^a_r$ **sufficient to establish soundness of** π **at** s
- Certificate $\mathcal{C}_\pi = \{\, \Phi_r = \exists\bar{z}\big(\bigvee_{a \in D} \Psi^a_r(\bar{z})\big) : r \,\}$ is **valid in domain** D iff for **any state** s **(reachable or not):**

$$s \vDash C_r \wedge \Psi^a_r(\bar{o}) \implies E_r \text{ is } \textbf{compatible} \text{ with } (s, res(s, a(\bar{o})))$$

Certificates: Scope and Result [B. et al., 2019b]

- $\mathcal{Q}[\mathcal{C}_\pi] = \{ P : \text{for all } r \text{ in } \pi, C_r \Rightarrow \Phi_r \text{ holds in } \textbf{all reachable states} \text{ of } P \}$

Theorem: If π is **acyclic** and $\mathcal{C}_\pi$ is **valid**, π **solves** $\mathcal{Q}[\mathcal{C}_\pi]$

That is, $\mathcal{C}_\pi$ guarantees that π is sound on the class (scope) $\mathcal{Q}[\mathcal{C}_\pi]$!

Hence, given π and instance P, to show π solves P:

- Show that π is acyclic (structural check, automatic)
- Obtain valid certificate $\mathcal{C}_\pi$ (automatic synthesis, see next slides)
- Check $C_r \Rightarrow \Phi_r$ hold on **reachable states** in P

Synthesis of Valid Certificates

- **Valid certificate** $\mathcal{C}_\pi$ obtained **automatically** using deduction:
  - ▷ Start with **base-for-deduction** that gives **sufficient/necessary** for ground atom $p(\bar{u})$ to hold **after** ground action $a(\bar{o})$ is applied in state s
  - ▷ Lift: Use induction on FO-formulas defining features f in rule r to obtain sufficient/necessary conditions for **value change** of f across an a-transition that is compatible with E_r
  - ▷ Combine lifted formulas with preconditions of concrete actions
- **Example:** $f(s) = |C(s)|$ **decrements** across (s, a, s') for $C = \{\bar{o} : \Psi(\bar{o})\}$ **if**

$$C(s') \subsetneq C(s) \quad \textbf{if}$$

$$S_C^{dec}(\bar{z}) = \forall \bar{x}\big(N_C^a(\bar{z}, \bar{x}) \Rightarrow \Psi(\bar{x})\big) \wedge \exists \bar{x}\big(\Psi(\bar{x}) \wedge \neg N_C^a(\bar{z}, \bar{x})\big)$$

where $N_C^a(\bar{z}, \bar{x})$ is **necessary condition** for $\bar{x}$ to be in $C(res(s, a(\bar{z})))$

Example: $clear(\mathbf{X})$ on Blocks with 3 Schemas (No Hand)

- **Schemas:** a_1=Newtower(z_1, z_2), a_2=Move(z_3, z_4, z_5), and a_3=Stack(z_6, z_7)
- Generalized policy π with single rule $\{n > 0\} \mapsto \{n\downarrow\}$ where $n = |\exists x(on^+(x, \mathsf{X}))|$
- Certificate $\mathcal{C}_\pi = \{\Phi = \exists \bar{z}\big(\bigvee_{a \in D} \Psi^a(\bar{z})\big)\}$ is singleton as there is single rule in π:

$$
\begin{aligned}
\Psi^{a1} &= Pre(a_1) \wedge on^*(z_2, \mathsf{A}) \wedge \forall y\big(on(z_1, y) \wedge on^*(y, \mathsf{A}) \Rightarrow y = z_2\big) \\
\Psi^{a2} &= Pre(a_2) \wedge on^+(z_3, \mathsf{A}) \wedge \neg on^*(z_5, \mathsf{A}) \wedge \forall y\big(on(z_3, y) \wedge on^*(y, \mathsf{A}) \Rightarrow y = z_4\big) \\
\Psi^{a3} &= \bot \\
\Phi &= \exists \bar{z}\big(\Psi^{a1}(z_1, z_2) \vee \Psi^{a2}(z_3, z_4, z_5)\big) \\
\mathcal{Q}[\mathcal{C}_\pi] &= \{\, P \;:\; \text{for } s \text{ reachable in } P,\ s \vDash \exists x(on^+(x, \mathsf{A})) \Rightarrow \Phi \,\}
\end{aligned}
$$

- Interpretation:
  - ▷ Ψ^{a1} says $clear(z_1)$, $on(z_1, z_2)$, $on^+(z_1, \mathsf{A})$, and if $on(z_1, y) \wedge on^*(y, \mathsf{A})$, then $y = z_2$
  - ▷ Ψ^{a2} says similar for (z_3, z_4) with the addition of $\neg on^*(z_5, \mathsf{A})$
  - ▷ $\Psi^{a3} = \bot$ since no ground instance of a_3 decreases n
  - ▷ $\mathcal{C}_\pi$ valid in **well-formed** Blocks instances; i.e. **π achieves $clear(\mathbf{A})$ in any such instance**

Example: Gripper with 3 Schemas

- **Schemas:** a_1=Move$(?r1,?r2)$, a_2=Pick$(?b,?g,?r)$, and a_3=Drop$(?b,?g,?r)$
- Acyclic solution for Gripper **learned with small instances with 2 rooms** [B. *et al.*, 2019b]:

$r_1 = \{\neg X, b > 0, g > 0\} \mapsto \{b\downarrow, g\downarrow, c\uparrow\}$	pick ball
$r_2 = \{X, c > 0\} \mapsto \{c\downarrow, (\uparrow g)\}$	drop ball
$r_3 = \{\neg X, b = 0, c > 0, g > 0\} \mapsto \{X\}$	go to room A (ver 1)
$r_4 = \{\neg X, c > 0, g = 0\} \mapsto \{X\}$	go to room A (ver 2)
$r_5 = \{X, c = 0, g > 0\} \mapsto \{\neg X\}$	leave room A

- Defined over the features:
  - ▷ $X = \{\, r \,:\, at(r) \wedge r = \text{A} \,\}$ tells if robot is in room A
  - ▷ $b = \{\, x \,:\, \exists r(in(x,r) \wedge r \neq \text{A}) \,\}$ counts balls in room B
  - ▷ $c = \{\, x \,:\, \exists g(carry(x,g)) \,\}$ counts balls being held
  - ▷ $g = \{\, x \,:\, free(x) \,\}$ counts free grippers
- **Example:** $\Psi^{a_2}_{r_1} = at(?r) \wedge in(?b,?r) \wedge free(?g) \wedge \forall x[\neg carry(?b,x)] \wedge ?r \neq \text{A}$
- $\mathcal{C}_\pi$ entailed by **standard mutexes** in Gripper: **π solves any instance with 2 rooms**

Challenge: Reduction of Certificates to Initial State

- **Know:** how to get valid certificate $\mathcal{C}_\pi = \{\Phi_r : r \in \pi\}$ from π
- If π is acyclic, π solves all instances in $\mathcal{Q}[\mathcal{C}_\pi]$ **($\mathcal{C}_\pi$ gives scope of π)**
- For given P, deciding if $P \in \mathcal{Q}[\mathcal{C}_\pi]$ involves checking $\bigwedge_{r \in \pi}(C_r \Rightarrow \Phi_r)$ on the **reachable states** in P
- It'd be much nicer to do some check **only on the initial state** of P
- Is there (non-trivial) Λ_π so that π is sound on $\mathcal{Q}[\Lambda_\pi] = \{\, P{=}(D, I) \,:\, I \vDash \Lambda_\pi \,\}$?

- Another recent approach for automatically checking soundness of abstractions has been put forward [Cui *et al.*, 2022]

Wrap Up

- Generalized planning is the problem of obtaining policies for solving classes of instances $\mathcal{Q}$
- Language of Boolean and numerical features allows expressive and succinct abstractions
- General policy is set of rules defined with features that filter out transitions
- Policy π solves $\mathcal{Q}$ if **acyclic** and **sound** on each instance P in $\mathcal{Q}$
- Acyclicity established by structural properties of π
- Soundness requires reasoning with reachable states in "well-formed instances"
- Yet, given π, one can characterize subclass $\mathcal{Q}'$ on which π **guaranteed** to succeed
- Challenges: reduce invariants to initial state, distance features, . . .

References

[Bonet et al., 2019a] Bonet, B., Francès, G., and Geffner, H. (2019a). Learning features and abstract actions for computing generalized plans. In *Proc. AAAI*, pages 2703–2710.

[Bonet et al., 2019b] Bonet, B., Fuentetaja, R., E-Martín, Y., and Borrajo, D. (2019b). Guarantees for sound abstractions for generalized planning. In *Proc. IJCAI*, pages 1566–1573.

[Bonet and Geffner, 2018] Bonet, B. and Geffner, H. (2018). Features, projections, and representation change for generalized planning. In *Proc. IJCAI*, pages 4667–4673.

[Bonet and Geffner, 2020] Bonet, B. and Geffner, H. (2020). Qualitative numeric planning: Reductions and complexity. *Journal of AI Research*, 69:923–961.

[Cui et al., 2022] Cui, Z., Kuang, W., and Liu, Y. (2022). Automatic verification of sound abstractions for generalized planning. *arXiv preprint arXiv:2205.11898*.

[Drexler et al., 2021] Drexler, D., Seipp, J., and Geffner, H. (2021). Expressing and exploiting the common subgoal structure of classical planning domains using sketches. In *Proc. KR*, pages 258–268.

[Drexler et al., 2022] Drexler, D., Seipp, J., and Geffner, H. (2022). Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proc. ICAPS*.

[Fern et al., 2006] Fern, A., Yoon, S., and Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118.

[Francès et al., 2021] Francès, G., Bonet, B., and Geffner, H. (2021). Learning general planning policies from small examples without supervision. In *Proc. AAAI*, pages 11801–11808.

[Geffner and Bonet, 2013] Geffner, H. and Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

[Hu and De Giacomo, 2011] Hu, Y. and De Giacomo, G. (2011). Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. IJCAI*, pages 918–923.

[Khardon, 1999] Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148.

[Martín and Geffner, 2004] Martín, M. and Geffner, H. (2004). Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20(1):9–19.

[Rodriguez et al., 2021] Rodriguez, I. D., Bonet, B., Sardina, S., , and Geffner, H. (2021). Flexible fond planning with explicit fairness assumptions. In *Proc. ICAPS*, pages 290–298.

[Srivastava et al., 2011] Srivastava, S., Zilberstein, S., Immerman, N., and Geffner, H. (2011). Qualitative numeric planning. In *AAAI*.

[Ståhlberg et al., 2022a] Ståhlberg, S., Bonet, B., and Geffner, H. (2022a). Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proc. ICAPS*.

[Ståhlberg et al., 2022b] Ståhlberg, S., Bonet, B., and Geffner, H. (2022b). Learning generalized policies without supervision using gnns. In *Proc. KR*.

Appendix: Synthesis of Valid Certificates

- **Example:** Gripper with schemas Move$(r1, r2)$, Pick(b, g, r) and Drop(b, g, r), and let concept $C = \{x : free(x)\}$ track set of free grippers

  ▷ Feature $f = |C|$ **decreases** across (s, s') due to Pick(b,g,r) **if** $C(s) \supsetneq C(s')$ **if** $s \vDash free(g)$ since $\neg free(g)$ is negative effect of action; i.e.,

$$
\begin{aligned}
& \forall x\big(N_C^a(\bar{z}, x) \Rightarrow \Psi(x)\big) \wedge \exists x\big(\Psi(x) \wedge \neg N_C^a(\bar{z}, x)\big) \\
\equiv\; & \forall x\big(N_C^a(\bar{z}, x) \Rightarrow free(x)\big) \wedge \exists x\big(free(x) \wedge \neg N_C^a(\bar{z}, x)\big) \\
\equiv\; & \top \wedge \exists x\big(free(x) \wedge \neg N_C^a(\bar{z}, x)\big) \\
\equiv\; & \exists x\big(free(x) \wedge \neg([\![free(x) \in Post]\!] \vee (free(x) \wedge [\![\neg free(x) \notin Post]\!]))\big) \\
\equiv\; & \exists x\big(free(x) \wedge \neg(\bot \vee (free(x) \wedge (x \neq g)))\big) \\
\equiv\; & free(g)
\end{aligned}
$$

  ▷ Thus, $f{\downarrow}$ across (s, s') by $Pick(b, g, r)$ if $s \vDash free(g)$ which true by prec

- General synthesis method for policies defined with FO-features [B. *et al.*, 2019b]

Appendix: Base for Deduction

Reference	Formula	
$\mathcal{B}_X(a,p)(\bar{z},\bar{x})$	$[\![p(\bar{x}) \in \text{Post}]\!] \vee (p(\bar{x}) \wedge [\![\neg p(\bar{x}) \notin \text{Post}]\!])$	
$\mathcal{B}_N(a,p^*)(\bar{z},x,y)$	$p^*(x,y) \vee \exists uv\big([\![p(u,v) \in \text{Post}]\!] \wedge p^*(x,u) \wedge p^*(v,y)\big)$	(action adds at most 1 p-atom)
	$p^*(x,y) \vee \exists uv\big([\![p(u,v) \in \text{Post}]\!] \wedge (p^*(x,u) \vee p^*(v,y))\big)$	(action adds 2 or more p-atoms)
$\mathcal{B}_S(a,p^*)(\bar{z},x,y)$	$(x=y) \vee \big(p^*(x,y) \wedge \forall uv([\![\neg p(u,v) \in \text{Post}]\!] \Rightarrow u=v)\big)$	

Table 1: General base $\mathcal{B}$ for synthesis of any domain $\mathcal{D}$. Post$(a(\bar{z}))$ is abbreviated by Post. $X \in \{N, S\}$. There are two versions of the necessary condition for p^*; one for actions that add at most one atom $p(u,v)$, and the other for actions that add two or more atoms of this form. The first version uses a conjunction, $p^*(x,u) \wedge p^*(v,y)$, while the second version replaces it with a disjunction.