On Policy Reuse: An Expressive Language for Representing and Executing Policies that Call Other Policies

Blai Bonet, Dominik Drexler, Hector Geffner

ICAPS, 2024

Hector Geffner
RWTH Aachen University
Aachen, Germany

Linköping University
Linköping, Sweden

LINKÖPING UNIVERSITY

Alexander von HUMBOLDT STIFTUNG

RWTH AACHEN UNIVERSITY

erc
European Research Council
Established by the European Commission

TAILOR
Foundation of Trustworthy AI: Integrating Learning, Optimisation and Reasoning

Motivation

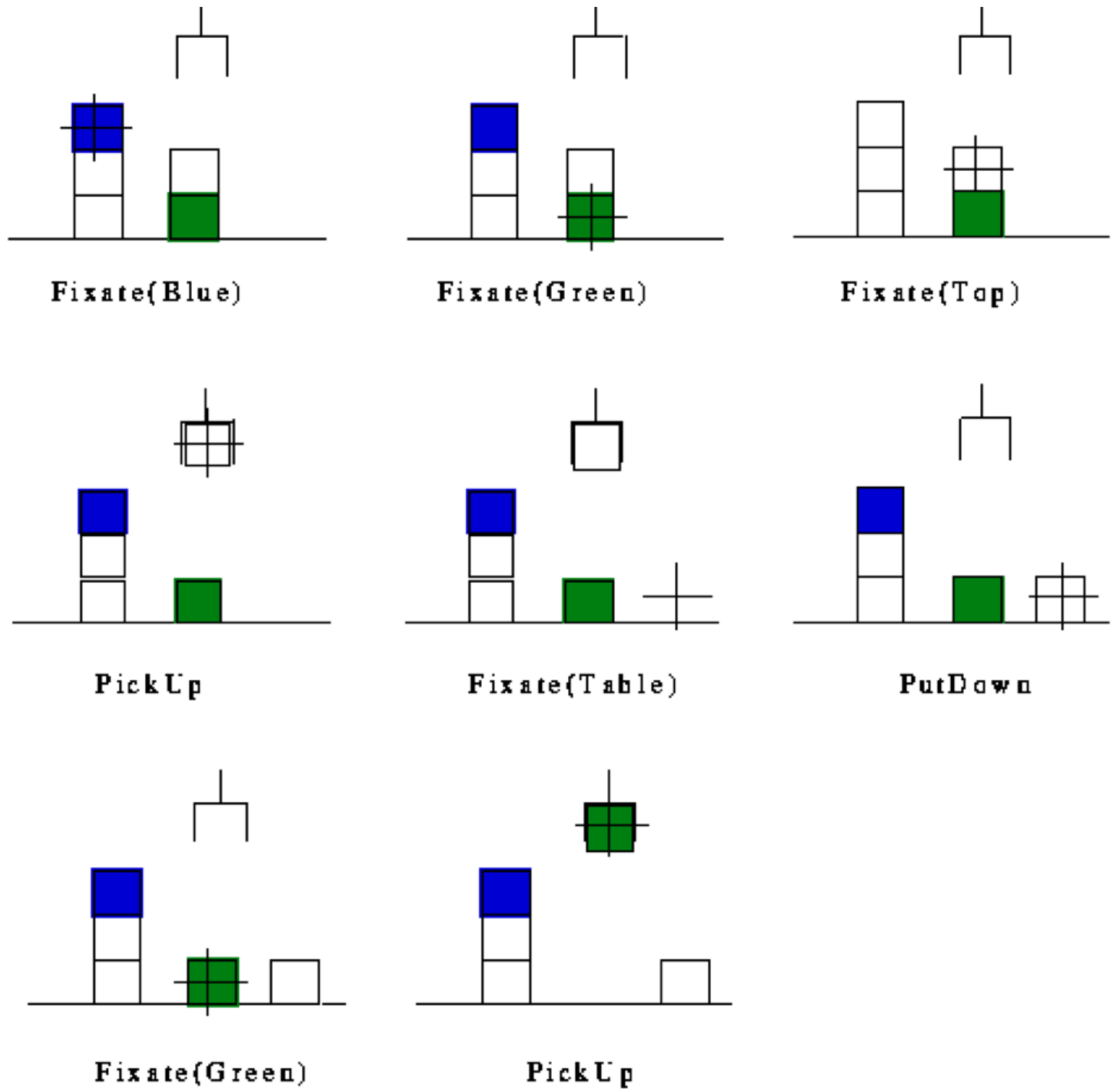
More expressive languages for **encoding** and **learning** general policies and sketches that support:

- **Reuse:** ability to call other policies by passing parameters
  - ▷ **Composition** and orchestration of subpolicies
  - ▷ **Bottom-up** construction of hierachies, as opposed to top-down
  - ▷ **Answers**: "Can policy for $on(x, y)$ be reused to construct arbitrary towers?"
- **Indexicals:** ability to refer to objects functionally, not by name
  - ▷ **Features** for capturing general policies/sketches simplified
  - ▷ **Active perception**: what to observe and when
  - ▷ **Determine action to do** without considering other actions/transitions

Related Research Threads

- **Planning programs and inductive programming** [2, 11, 12, 5].
  - ▷ Dreamcoder: Growing generalizable knowledge with program learning. K. Ellis *et al.*; 2020
  - ▷ Generalized planning as heuristic search. J. Segovia, S. Jimenez, A. Jonsson, AIJ 2021
- **General policies** [9, 10, 6], [13], [15, 8, 14].
  - ▷ Learning generalized policies using concept languages. M. Martin, H. G., KR 2000
- **Deictic representations** [4, 1, 3, 7].
  - ▷ David Chapman. Penguins can make cake. AI Magazine, 1989
  - ▷ Deictic codes for the embodiment of cognition. D. Ballard *et al.*, BBS 1997
  - ▷ The thing that we tried didn't work very well: Deictic representation in RL, S. Finney *et al.*, UAI 2022.

Example: Pick up green blocks; Ballard *et al.* 1997



This Work

Extensions to the language of **general policies** and **sketches**:

- Indexical pointers to objects
- Memory states
- Ground actions
- Modules that call other modules (reuse)

Example: General Policy for $clear(x)$

- **Policy** π for class $\mathcal{Q}_{clear}$ of problems with goal $clear(x)$ in Blocks:

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

- **Features** $\Phi = \{H, n\}$: 'holding' and 'number of blocks above x'
- **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**
- **Shortcomings:**
  - ▷ Policy doesn't select actions directly; e.g. `pickup`(A), if A top block above x
  - ▷ Feature n for 'number of blocks above x', is "complex"

Example: New indexical policy for $clear(x)$

Concepts: used as features and to sample objects

- H_1 Boolean, whether block in $\mathfrak{r}_1$ is being held
- Table_1: Boolean, whether block in $\mathfrak{r}_1$ on table
- X: concept only contains given block x
- T_0: concept that contains block on block in register $\mathfrak{r}_0$ (if any)
- T_1: concept that contains block on block in register $\mathfrak{r}_1$ (if any)
- Initial memory state is always m_0; rule application change m_i

% Internal rules (update registers and internal memory; **no state transitions involved***)*

$r_0 = m_0 \,\|\, \{\text{X} > 0\} \mapsto \{\textit{Load}(\text{X}, \mathfrak{r}_0), \text{T}_0?\} \,\|\, m_1$ (Load x into register $\mathfrak{r}_0$)

$r_1 = m_1 \,\|\, \{\text{T}_0 > 0\} \mapsto \{\textit{Load}(\text{T}_0, \mathfrak{r}_1), \text{T}_1?\} \,\|\, m_2$ (Load block on x in $\mathfrak{r}_1$, if any)

$r_2 = m_2 \,\|\, \{\text{T}_1 > 0\} \mapsto \{\textit{Load}(\text{T}_1, \mathfrak{r}_1), \text{T}_1?\} \,\|\, m_2$ (Loop. Load block on $\mathfrak{r}_1$ in $\mathfrak{r}_1$

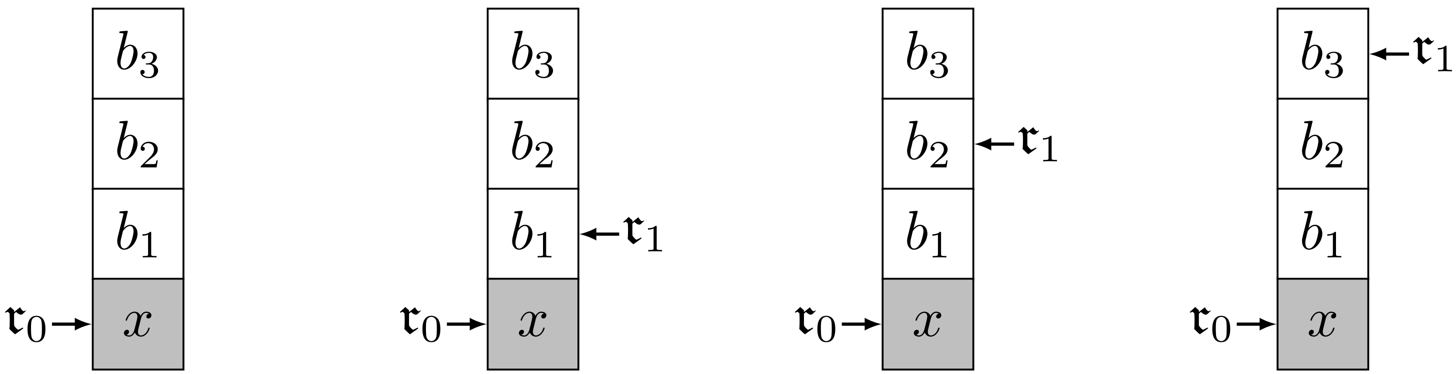
$r_3 = m_2 \,\|\, \{\text{T}_1 = 0\} \mapsto \{\} \,\|\, m_3$ until no such blocks, then

% External rules **(state transitions involved)**

$r_4 = m_3 \,\|\, \{\neg H_1\} \mapsto \{H_1\} \,\|\, m_3$ Unstack $\mathfrak{r}_1$)

$r_5 = m_3 \,\|\, \{H_1\} \mapsto \{\text{Table}_1, \neg H_1\} \,\|\, m_1$ (Put block being held on table, and loop)

Example: Execution of new indexical policy for $clear(x)$



$\mathsf{T}_0 = \{b_1\}$	$\mathsf{T}_0 = \{b_1\}$	$\mathsf{T}_0 = \{b_1\}$	$\mathsf{T}_0 = \{b_1\}$
	$\mathsf{T}_1 = \{b_2\}$	$\mathsf{T}_1 = \{b_3\}$	$\mathsf{T}_1 = \emptyset$

- Initially, load x in register $\mathfrak{r}_0$; equivalenty, **mark** x with $\mathfrak{r}_0$
- Put $\mathfrak{r}_1$ **mark** on block that is on the one marked with $\mathfrak{r}_0$
- Move $\mathfrak{r}_1$ **mark** to block that is on the one marked with $\mathfrak{r}_1$
- Until block with $\mathfrak{r}_1$ **mark** is clear and can be picked up directly

Extended Sketch/Policy Language

- **Concepts** C (unary predicates) used explicitly as Boolean features, $\mathsf{C} > 0$, numerical features C$\downarrow$, and for sampling objects

- **Registers** $\mathfrak{r}_i$ can be "loaded" with objects sampled from concepts; *Load*$(\mathsf{C}, \mathfrak{r}_i)$; registers are concepts too.

- **Memory states** m_i control flow along with Boolean conditions; e.g.,
  $m_1 \parallel \{C\} \mapsto \{E\} \parallel m_2$

- Rules with load effects or empty effects deemed as **internal rules**; others as **external rules**

- Memory states of internal rules and external rules different

- See paper for **formal syntax** and **semantics**

Modules: Reusing Policies

- Policies and sketches wrapped into *modules*
- **Modules may call other modules and do recursion passing parameters**
- Execution model uses a **stack** and caller/callee protocol, as in prog. languages
- Orchestration of collections $\{\texttt{mod}_0, \texttt{mod}_1, \texttt{mod}_2, \ldots\}$ of modules
- Additional external rules in modules:
  - ▷ **Call rules:** $m \,\|\, C \mapsto \texttt{mod}(\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_k) \,\|\, m'$ where C is condition, and m and m' are memory states, to call mod with $\mathsf{C}_1, \ldots, \mathsf{C}_k$ as arguments
  - ▷ **Do rules:** $m \,\|\, C \mapsto \texttt{act}(\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_k) \,\|\, m'$ to apply a **ground action** $\texttt{act}(o_1, o_2, \ldots, o_k)$ with objects $o_i \in \mathsf{C}_i$, for $i = 1, 2, \ldots, k$

Example: Modules for $on(x, y)$

Module $\texttt{On}(\mathsf{X}, \mathsf{Y})$:

$r_0 \;=\; m_0 \,\|\, \{\neg On\} \mapsto \texttt{Clear}(\mathsf{X}) \,\|\, m_1$ (Call `Clear` with argument X)

$r_1 \;=\; m_1 \,\|\, \{\} \mapsto \texttt{Clear}(\mathsf{Y}) \,\|\, m_2$ (Call `Clear` with argument Y)

$r_2 \;=\; m_2 \,\|\, \{\neg H_{\mathsf{X}}\} \mapsto \{H_{\mathsf{X}}\} \,\|\, m_3$ (Pick block x, either `unstack` or `pickup`)

$r_3 \;=\; m_3 \,\|\, \{H_{\mathsf{X}}\} \mapsto \texttt{stack}(\mathsf{X}, \mathsf{Y}) \,\|\, m_3$ (Apply `stack` to put x on y)

Module $\texttt{Clear}(\mathsf{X})$:

$r_0 \;=\; m_0 \,\|\, \{\mathsf{X} > 0\} \mapsto \{\mathit{Load}(\mathsf{X}, \mathfrak{r}_0), \mathsf{T}_0?\} \,\|\, m_1$ (Load x in register $\mathfrak{r}_0$)

$r_1 \;=\; m_1 \,\|\, \{\mathsf{T}_0 > 0\} \mapsto \{\mathit{Load}(\mathsf{T}_0, \mathfrak{r}_1), \mathsf{T}_1?\} \,\|\, m_2$ (Load block above x in $\mathfrak{r}_1$, if any)

$r_2 \;=\; m_2 \,\|\, \{\mathsf{T}_1 > 0\} \mapsto \{\mathit{Load}(\mathsf{T}_1, \mathfrak{r}_1), \mathsf{T}_1?\} \,\|\, m_2$ (Loop. Load block above $\mathfrak{r}_1$ in $\mathfrak{r}_1$)

$r_3 \;=\; m_2 \,\|\, \{\mathsf{T}_1 = 0\} \mapsto \{\} \,\|\, m_3$ (Go to external rules)

$r_4 \;=\; m_3 \,\|\, \{\neg H\} \mapsto \texttt{unstack}(\mathfrak{r}_1, \mathsf{B}) \,\|\, m_3$ (Apply `unstack` to pick $\mathfrak{r}_1$)

$r_5 \;=\; m_3 \,\|\, \{H\} \mapsto \texttt{putdown}(\mathfrak{r}_1) \,\|\, m_1$ (Apply `putdown` to put $\mathfrak{r}_1$ on table)

Example: Building One Tower with Module $\texttt{Tower}(\mathbf{O}, \mathbf{X})$

- Objective is to build tower $\bigwedge_{i=1}^{k} on(x_i, x_{i-1}) \wedge ontable(x_0)$
- Role argument $\mathsf{O} = \{(x_i, x_{i-1}) \mid i = 1, \ldots, k\}$
- X is concept for *lowest* block in tower that is *misplaced*
- M is concept for block to be placed on $\mathfrak{r}_0$ according to O (if any)
- W is concept for block below $\mathfrak{r}_0$ according to O (if any)

Module $\texttt{Tower}(\mathsf{O}, \mathsf{X})$:

$$r_0 \;=\; m_0 \,\|\, \{X > 0\} \mapsto \{\textit{Load}(\mathsf{X}, \mathfrak{r}_0), \mathsf{M}?, \mathsf{W}?\} \,\|\, m_1 \qquad \text{(Load X into register } \mathfrak{r}_0\text{)}$$

$$r_1 \;=\; m_1 \,\|\, \{W = 0\} \mapsto \texttt{On-Table}(\mathfrak{r}_0) \,\|\, m_2 \qquad (\texttt{On-Table} \text{ to put X on table})$$

$$r_2 \;=\; m_1 \,\|\, \{W > 0\} \mapsto \texttt{On}(\mathfrak{r}_0, \mathsf{W}) \,\|\, m_2 \qquad (\texttt{On}(\mathfrak{r}_0, \mathsf{W}) \text{ to well-place } \mathfrak{r}_0)$$

$$r_3 \;=\; m_2 \,\|\, \{M > 0\} \mapsto \texttt{Tower}(\mathsf{O}, \mathsf{M}) \,\|\, m_3 \qquad \text{(Continue building tower from M)}$$

Example: Building Many Towers

- Argument O is **role** that contains the pairs describing the towers to build
- L is concept for *lowest misplaced blocks* according to O

Module `Blocks`(O):

$$r_0 \;=\; m_0 \,\|\, \{\mathsf{L} > 0\} \mapsto \{\mathit{Load}(\mathsf{L}, \mathfrak{r}_0)\} \,\|\, m_1 \qquad \text{(Load X into register } \mathfrak{r}_0\text{)}$$

$$r_1 \;=\; m_1 \,\|\, \{\} \mapsto \mathtt{Tower}(\mathsf{O}, \mathfrak{r}_0) \,\|\, m_0 \qquad \text{(Build tower on } \mathfrak{r}_0\text{)}$$

Summary. Future

Language extensions for **encoding** and **learning** general policies and sketches:

- **Reuse** and **bottom up** composition of policies
- Don't **learn policies from scratch;** resue those learned
- **Indexicals** (registers) simplify features, determine actions to do, active perception
- **Interpreter** available, but not learning yet
- **Limitations.** Language is:
  - ▷ "too much": hard to learn and verify, too many alternative encodings
  - ▷ "too little": flexibility lacking for handling negative interactions
- One step; others to follow

References

[1] Philip E. Agre and David Chapman. What are plans for? *Robotics and Autonomous Systems*, 6:17–34, 1990.

[2] Javier Segovia Aguas, Sergio Jiménez Celorrio, and Anders Jonsson. Generalized planning with procedural domain control knowledge. In *Proc. ICAPS*, pages 285–293, 2016.

[3] Dana H. Ballard, Mary M. Hayhoe, Polly K. Pook, and Rajesh P. N. Rao. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, 20:723–742, 1996.

[4] David Chapman. Penguins can make cake. *AI magazine*, 10(4):45–45, 1989.

[5] Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Phil. Trans. R. Soc. A*, 381:20220050, 2023.

[6] Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

[7] Sarah Finney, Natalia Gardiol, Leslie Pack Kaelbling, and Tim Oates. The thing that we tried didn't work very well : Deictic representation in reinforcement learning. *CoRR*, abs/1301.0567, 2013.

[8] Sankalp Garg, Aniket Bajpai, and Mausam. Generalized neural policies for relational mdps. In *Proc. ICML*, 2020.

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

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

[11] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Computing programs for generalized planning using a classical planner. *Artificial Intelligence*, 272:52–85, 2019.

[12] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Generalized planning as heuristic search. In *ICAPS*, pages 569–577, 2021.

[13] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):393–401, 2011.

[14] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In *Proc. KR*, pages 647–657, 2023.

[15] Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68:1–68, 2020.