Automatic Cross-Domain Task Plan Transfer by Caching Abstract Skills

Khen Elimelech

In collaboration with **Lydia E. Kavraki** and **Moshe Y. Vardi**

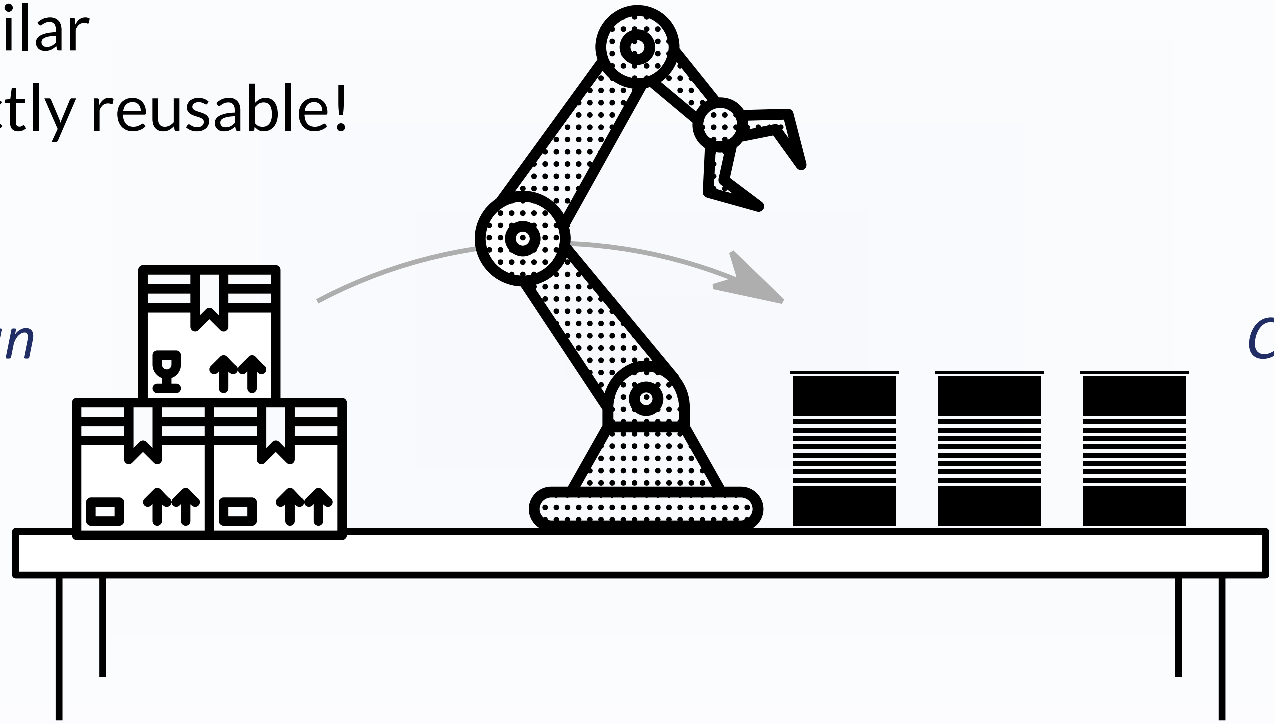
Department of Computer Science, Rice University

RICE UNIVERSITY

Motivation

- Intuitively similar
- Plans not exactly reusable!

Box stacking plan



Can stacking plan

"How can we <u>automatically adapt</u> and reuse <u>knowledge</u> from past successful plans to efficiently solve new tasks (in new domains)?"

Just a few basic definitions…

- **Planning domain:** state space, action space, discrete deterministic transitions

- **Plan:** a discrete sequence of actions $(a_1, \dots, a_n)$

- **Task:** specified objectives we care to satisfy with the plan execution
  - Can assume every task in a domain

- **Task planning problem:** given a planning domain, an initial state, and a task, find a plan whose execution satisfies the task objective

Plan transfer: formally

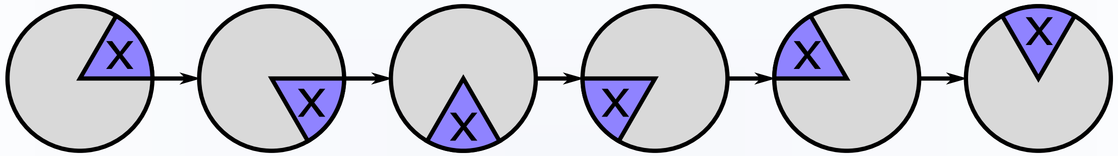
- **Given a plan $(\boldsymbol{a_1}, \dots, \boldsymbol{a_n})$ which solves a task planning problem $\boldsymbol{P}$ in domain $\boldsymbol{D}$, find a plan $(\boldsymbol{a'_1}, \dots, \boldsymbol{a'_m})$ which solves a similar planning problem $\boldsymbol{P}$' in domain $\boldsymbol{D}$'**

- Standard techniques:
  - Find/learn an explicit mappings ("transfer functions") between the action sequences:
  $$(a_1, \dots, a_n) \mapsto (a'_1, \dots, a'_m)$$
  - Find/learn reusable "macro actions" or policies that are applicable to both problems
- Numerous drawbacks, e.g.
  1. Only allows transfer between robots with the same capabilities (action set)
  2. "Fragile" – if one action is inapplicable/unfeasible, the entire plan becomes invalid
  3. Unclear how to automate: must learn transfer for every new problem, requires prior domain/task knowledge...

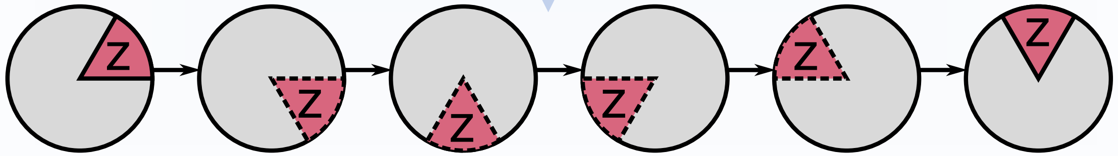
Transferring state road maps

- Let us instead examine the successful <u>execution</u> of the given task plan in our domain:

$$\text{execution} \doteq (\boldsymbol{S}_0, a_1, \boldsymbol{S}_1, \dots, a_n, \boldsymbol{S}_n)$$



- We suggest to transfer the sequence of states (“road map”) – not actions!
- Actions can be recovered in the destination domain, after the transfer

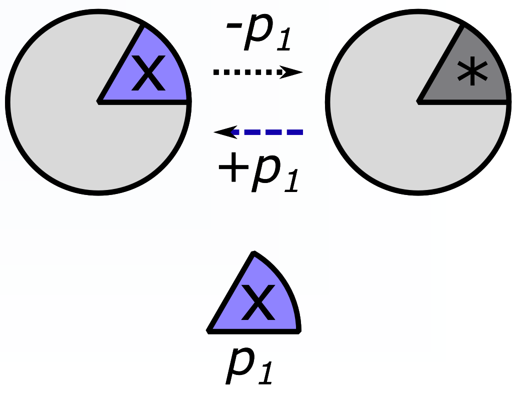


- States are intuitively transferable, even if considering different action sets
- Flexible – can adapt* actions between states, easily recover from state fail, de/compose road maps, etc.

(*) In practice, decomposing a task into <u>dynamically-defined</u> sub-tasks

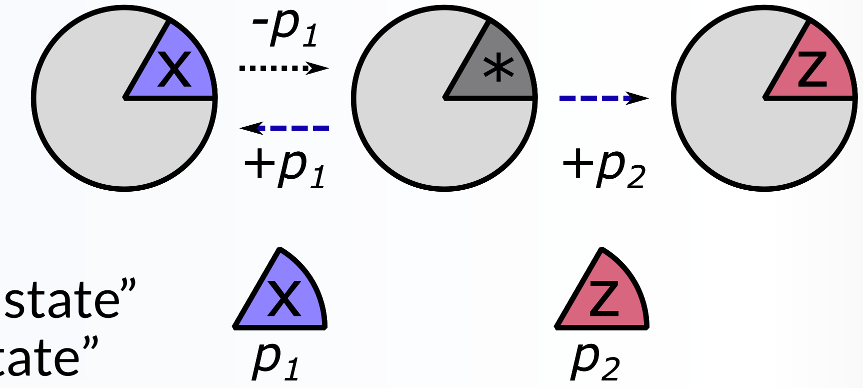
Road map transfer using abstraction keys

- **Public abstraction key**: a pair of inverse parametric functions
  - $Project_p$: $state \mapsto abstract\_state$
  - $Reconstruct_p$: $abstract\_state \mapsto state$
- **Private abstraction key**: a problem/state-specific parameter value p
- Similar to "encryption keys", requires both keys to reconstruct an abstracted state
- For example, the "symbol stripping" abstraction key:
- Abstract state = a state described with a subset of symbols



Road map transfer using abstraction keys

- Provide a systematic and automatable way to perform state transformation: **Project and reconstruct with an alternative private key**

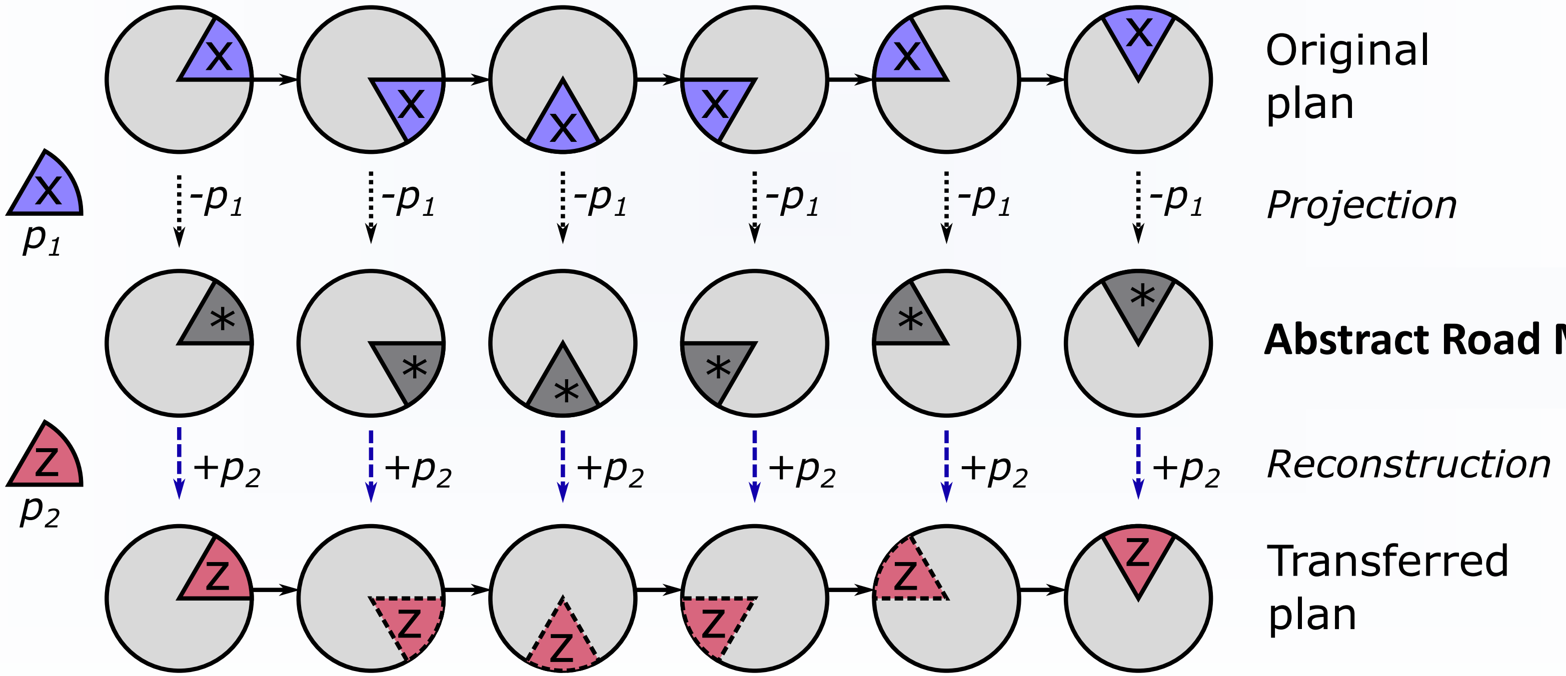


- **Intuitively**:
  - p specifies a "property value"
  - Projection function "removes the reference to p from state"
  - Reconstruction function "adds the reference to p to state"

- Different public abstraction keys allow to perform different transformations or modify different state properties

Plan transfer using “symbol stripping” key

- We can apply this technique to the entire road map…

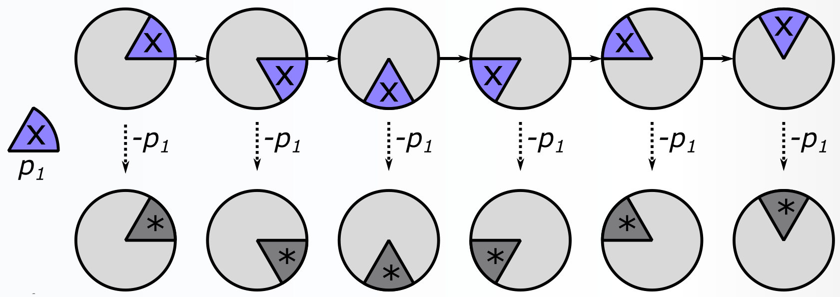


Caching abstract skills

- The two stages of the transfer can and should be done separately!

1. Upon solution of a task planning problem:

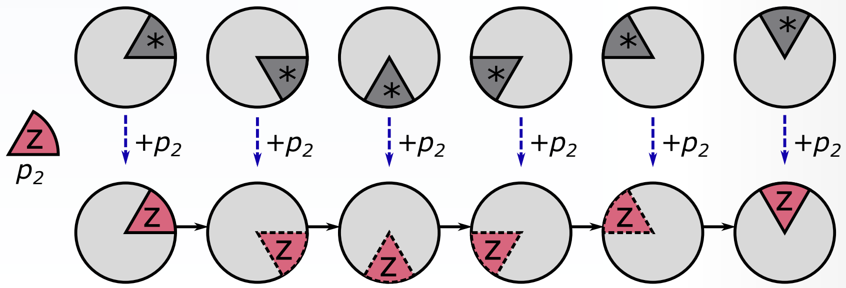
- Project the road map to an abstract one, and cache it
- **The abstract road map (and public key) represent an <u>abstract skill</u>**



---

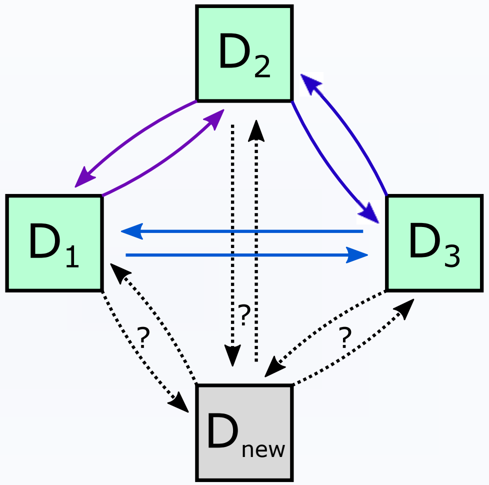
2. On demand, when facing a new task:

- Reconstruct ("ground") the abstract road map in the new domain
- Recover actions to follow the road map

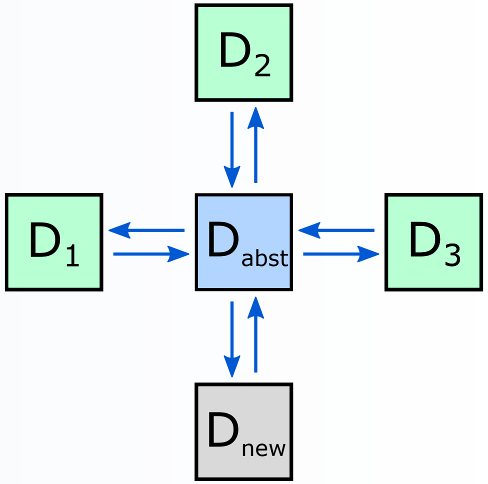


Transfer through an abstract domain

Naïve transfer



Transfer with a central abstract domain



This way….

- No domain-to-domain coupling
- Can transfer skills to unseen domains
- We can maintain a unified and compact skill library
- Increased scalability

* For each skill abstract domain is dynamically determined by the choice of abstraction keys

Recap: novel contributions

- Presented fundamental theoretical framework [1]
    - Skills represented as <u>state road maps</u>, not action plans (more flexible!)
    - Skills transferred through and cached in an <u>abstract domain</u>
    - Transfer can be done automatically using <u>abstraction keys</u>
    - Essentially, automatically learning a generalizable skill from a <u>single demonstration</u>

- Practical aspects in follow-up paper [2]
    - Finding private keys for transfer as a constraint satisfaction problem
    - Towards Task and Motion Planning (TAMP)

[1] Elimelech et al., *GenPlan Workshop* and *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2022.

[2] Elimelech et al., *International Symposium on Robotics Research (ISRR)*, 2022.

Thank you!

www.khen.io

Be kind to yourself. Be kind to others. Be kind to Nature.