Accelerating Long-Horizon Planning with Affordance-Directed Dynamic Grounding of Abstract Strategies

Khen Elimelech, Zachary Kingston, Wil Thomason, Moshe Y. Vardi and Lydia E. Kavraki

***Abstract*— Long-horizon task planning is important for robot autonomy, especially as a subroutine for frameworks such as Integrated Task and Motion Planning. However, task planning is computationally challenging and struggles to scale to realistic problem settings. We propose to accelerate task planning over an agent's lifetime by integrating *abstract strategies*: a generalizable planning experience encoding introduced in earlier work. In this work, we contribute a practical approach to planning with strategies by introducing a novel formalism of planning in a *strategy-augmented domain*. We also introduce and formulate the notion of a strategy's *affordance*, which indicates its predicted benefit to the solution, and use it to guide the planning and strategy grounding processes. Together, our observations yield an affordance-directed, lazy-search planning algorithm, which can seamlessly compose strategies and actions to solve long-horizon planning problems. We evaluate our planner in an object rearrangement domain, where we demonstrate performance benefits relative to a state-of-the-art task planner.**

I. Introduction

Long-horizon task planning—reasoning about temporally extended sequences of actions to achieve a symbolically specified goal—is core to robot autonomy. Many approaches to embodied planning, such as Integrated Task and Motion Planning (TAMP), use task planning as a repeatedly called subroutine [1], which must be fast to execute. However, task planning is computationally hard [2] and empirically difficult to scale to large, realistic planning domains, e.g., with many potentially irrelevant objects and distractions.

We propose to accelerate long-horizon planning by incorporating *abstract strategies*. Such strategies, introduced in previous work [3–5] (where they were referred to as "abstract skills"), encode generalizable planning experience for future reuse. They comprise a state trace ("road map") in a strategy-specific abstract state space, coupled with an "abstraction key" defining a mapping between this abstract state space and the state space of the planning problem. Each abstraction key performs a type of transformation on the abstract strategy's road map to dynamically adjust it to new domains. Following a strategy's road map decomposes a planning problem into simpler sub-problems to guide the planning process.

In this paper, we extend prior techniques for planning with strategies [4] into a novel and more practical approach, by incorporating the estimation of the strategies' *affordances*. These affordances implicitly define a local "basin of attraction" for each strategy, guiding the planner toward strategies that best match the problem, and are expected to accelerate

Work on this paper was supported in part by NSF-IIS-1830549, NSF ITR 2127309, and Rice University Funds. The authors are with the Department of Computer Science, Rice University, Houston, TX 77005, USA. {elimelech,vardi,kavraki}@rice.edu

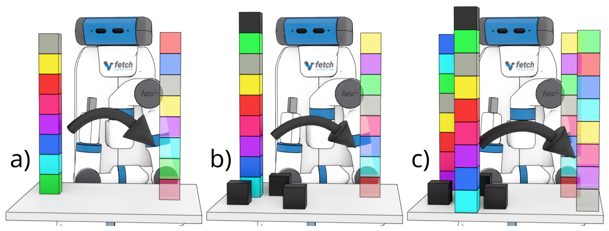

Fig. 1: Generalization and composition of strategies. **(a)** An example problem in which the agent seeks to permute a tower of blocks. **(b)** Later, another tower of blocks must be permuted in the same way, but with irrelevant distracting objects; the prescribed approach for strategy transfer enables the prior experience from (a) to be applied to solve this problem faster. **(c)** An example problem in which permutation must be applied to two different towers; this problem can be solved efficiently by composing multiple strategies.

the solution. Affordance-based planning alleviates the need to find a single strategy that exactly bridges a problem's start and goal (a limitation of prior work [4]), allowing us to effectively *compose* and long-horizon plan from multiple strategies and actions. These insights, together with the self-supervised strategy learning algorithm proposed in earlier work [5], support life-long, continually-improving planning agents, capable of automatically building and using a library of strategies to accelerate planning over a series of problems.

Specifically, **we contribute**: 1) a formulation of planning with strategies as "planning in the augmented domain." This formulation is supported by 2) a novel conceptualization of a strategy's "affordance," which is the basis for 3) a method for dynamically yielding the relevant groundings (reconstructions) of abstract strategies and actions, without enumeration. This approach underpins 4) a family of algorithms for affordance-directed planning with strategy libraries, including an "optimistic" planner, with "lazy" strategy realization. Finally, we provide 5) a practical proof-of-concept system (Fig. 1) implementing these algorithms, which, together with a strategy learning algorithm from previous work [5], accelerates planning in a lifelong plan-learn loop.

A. Background and Related work

Planning with strategies builds upon the basic formalisms of states, actions, and planning problems from task planning (also referred to as "AI/automated planning") [6, 7]. Task planning (using the Planning Domain Definition Language (PDDL) [8]) models planning problems as "domains" describing a deterministic transition system between states using actions; such states describe the agent and the scene in which it acts. Task planning has produced many successful domain-independent planners, most of which rely either on heuristic search of a plan graph [9, 10] or encoding the problem as Boolean satisfiability (SAT) [11, 12]. We compare against planners from this literature in Sec. VI, demonstrating that even a simple search-based planning algorithm incorporating strategies can outperform a highly-optimized state-of-the-art task planner, thanks to the strategy-induced problem decomposition. Further, our formulation of planning with strategies as planning in an strategy-augmented domain may allow strategies to integrate directly with existing planners for even greater performance improvement.

High-level actions or macro-actions [13, 14] have also been used to accelerate planning. These ideas and strategies both impose a hierarchy on the planning space, but strategies improve flexibility by only prescribing a sequence of states and not the actions to be used to transition between them. Similarly, task planning based on Hierarchical Task Networks and Hierarchical Goal Networks (HTNs and HGNs) predefines a hierarchy of actions (or sub-goals) and iteratively refines it into sequences of primitive actions [15–17]. In contrast, strategies do not predefine concrete sub-goals or action sequences and do not require a strict hierarchy of action types. Abstract strategies are dynamically grounded, meaning that their sub-goals depend on the context of their use and that they can be used interchangeably with primitive actions. In hierarchical reinforcement learning, high-level actions are often encoded as policies, referred to as "skills" or "options" [18–21]. Such components simplify search in Markov decision process problems. Other work has investigated task planner-guided manipulation strategy learning [22] in the context of TAMP; these strategies correspond to continuous controllers for specific instances of primitive actions. Also in TAMP, Vega-Brown and Roy [23] investigate bounds on the estimated cost of high-level actions for accelerating planning without sacrificing optimality. These perspectives on strategies largely address different problems than we study in this work.

The term "affordance" is overloaded in robotics (particularly in manipulation) and commonly refers to a model of an action's immediate value [24–26]. Work on TAMP and long-horizon embodied planning has used learned value functions as affordances to guide action selection [27, 28] or instantiation [29]. Some work reasons over expected affordances of actions at future states [30], which temporally extends the classical notion of affordance in manipulation. Perhaps most similar to this paper, Awaad et al. [31] use affordance estimation to improve HTN planning performance. Our concept of "affordance" generalizes many of these concepts, as it can be considered a collection of primitive affordance heuristics. For example, in this work, we combine predicted estimates of a strategy's immediate value, its cost of use (i.e., the cost of solving the sub-problems it imposes), and the remaining "cost-to-go" to the goal after using the strategy.

II. Preliminaries

Definition 1: A *task planning domain* $D \doteq (\mathcal{S}, \mathcal{A}, \mathcal{T})$ comprises a state space $\mathcal{S}$ (continuous or discrete), an action space $\mathcal{A}$, and a set $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ of discrete transitions. An *execution* $\boldsymbol{E}$ in is a sequence of alternating states and actions, i.e., $\boldsymbol{E} \doteq (\boldsymbol{S}_0, a_1, \boldsymbol{S}_1, a_2, \ldots, \boldsymbol{S}_n)$. $\boldsymbol{E}$ is *feasible* in D if $\forall i \in \{1, \ldots, n\}$, $(\boldsymbol{S}_{i-1}, a_i, \boldsymbol{S}_i) \in \mathcal{T}$. In that case, we say that $\boldsymbol{E}$ is induced by the action sequence $(a_1, \ldots, a_n)$. The trace of states from $\boldsymbol{E}$ is marked $\mathfrak{S}(\boldsymbol{E})$.

A task is a collection of constraints on the states and/or actions of an execution. For conciseness and practicality, this paper considers tasks with global reach-avoid constraints.

Definition 2: A *reach-avoid task* $T \doteq (\mathcal{G}oals, \mathcal{S}_{\text{avoid}})$ comprises a set $\mathcal{G}oals$ of goal regions $\subseteq \mathcal{S}$, and a region $\mathcal{S}_{\text{avoid}} \subseteq \mathcal{S}$ to avoid. For an execution $\boldsymbol{E}$, the task constraints require us to visit all the goal regions (in any order) and never reach the avoid regions:

$$\forall \mathcal{S}_{\text{goal}} \in \mathcal{G}oals,\ \exists \boldsymbol{S} \in \mathfrak{S}(\boldsymbol{E}),\ \text{s.t. } \boldsymbol{S} \in \mathcal{S}_{\text{goal}}, \tag{1}$$

$$\mathfrak{S}(\boldsymbol{E}) \cap \mathcal{S}_{\text{avoid}} = \emptyset. \tag{2}$$

To solve a task, we seek a plan (i.e., a sequence of actions) to be applied from the current start state, whose execution is feasible in the domain and satisfies the task constraints.

A. Abstract Strategies and Abstraction Keys

Definition 3: An *abstraction key* comprises a tuple, $AK \doteq (\texttt{project}_p, \texttt{reconst}_p, \mathcal{P})$, of a state projection function $\texttt{project}_p\colon \mathcal{S} \to \Xi$, its inverse—a state reconstruction function $\texttt{reconst}_p\colon \Xi \to \mathcal{S}$, and a parameter space $\mathcal{P}$, such that $p \in \mathcal{P}$. The sets of valid parameters for projection of $\boldsymbol{S}$ and reconstruction of ξ are $\mathcal{P}_{\boldsymbol{S}}$ and $\mathcal{P}^{\xi}$, respectively.

Every abstraction key allows to perform a certain *type* of transformation on states. Intuitively, projection removes a property from a state—leading to an abstract state in the *AK*-induced abstract state space Ξ—and reconstruction re-sets the property. The parameter p is used to specify these properties. In our experiments to follow, we use two previously-defined abstractions keys: "attention," and "symbol stripping."[1]

With that, we can discuss the concept of *abstract strategies*.

Definition 4: An *abstract strategy* $\boldsymbol{K} \doteq (ARM, AK)$ comprises an Abstract Road Map ARM, and an Abstraction Key *AK*. ARM is a trace in the *AK*-induced abstract state space.

Broadly, an abstract strategy can be learned and cached upon solution of a task by projecting (a part of) the state trace of the plan execution $\mathfrak{S}(\boldsymbol{E})$ into an abstract domain—using an abstraction key *AK*, and a projection parameter p chosen for that state trace. See [5] for details. Later, this strategy's ARM can be reconstructed (grounded) into a new domain by choosing a new parameter p' for reconstruction. The Reconstructed Road Map (RRM) will be used to guide the planning into "potentially promising" directions.

Definition 5: An abstract strategy $\boldsymbol{K}$ is *feasible* in domain D if exists a reconstruction of its ARM into D, i.e.,

$$\exists p \in AK.\mathcal{P}^{ARM} \text{ s.t. } RRM_p \doteq \texttt{reconst}_p(ARM) \subseteq D.\mathcal{S}, \tag{3}$$

for which exists a sequence of actions that traverses through the RRM_p. In that case we can say that RRM_p is feasible.

We refer to the process of matching the abstract strategy to a problem, reconstructing the abstract strategy's ARM into the problem's domain, and finding a sequence of actions to follow it in that domain as *realization of the abstract strategy*.

[1]For convenience, a formal definition and illustration of these keys is provided in `http://khen.io/icra24appendix.pdf`

III. Planning with Strategies: Novel Formulation

Our goal is to exploit a library of abstract strategies $\mathcal{L}ibrary$ to accelerate the solution of new planning problems in our agent's planning domain D. Prior work indicated that exactly matching an abstract strategy to a planning problem can indeed achieve this goal. Yet, finding a single strategy that is applicable directly from the initial state while also solves the task completely (as considered there), proved to be challenging. For more flexibility, we want to extend the formulation of "planning with abstract strategies" to support composition of of multiple strategies and primitive actions.

A. Abstract Strategies as Parametric Symbolic Actions

Consider an abstract strategy $\boldsymbol{K} \in \mathcal{L}ibrary$ that we wish to realize into D. This strategy can have multiple feasible reconstructions in D, using different reconstruction parameters; i.e., this abstract strategy may yield a *set* of realized strategies. Further, to reason about the *composability* of such strategies, we only need to consider their start and end states, i.e., the first and last states in each such strategy's (reconstructed) road map. In other words, for a realization of $\boldsymbol{K}$ using the reconstruction parameter p, it is sufficient to reason about the symbolic transition between $RRM_p[1]$ and $RRM_p[end]$, while encapsulating the internal states and actions; this transition can be labeled with the *strategy-action* K_p.

Overall, all the feasible realizations of an abstract strategy $\boldsymbol{K}$ in our domain can be represented using a *parametric* symbolic strategy-action K. The set of parameters with which reconstruction is feasible is also the parameter space of K, and is marked $\mathsf{K}.\mathcal{P}$. As commonly agreed, selection and assignment of a parameter value p for a parametric action (such as the the one defined here) is referred to as *grounding*.

B. Planning in the Strategy-Augmented Domain

Definition 6: A domain $D' = (\mathcal{S}, \mathcal{A}', \mathcal{T}')$ is an *augmentation* of domain $D = (\mathcal{S}, \mathcal{A}, \mathcal{T})$ if $\mathcal{T} \subseteq \mathcal{T}'$ and $\mathcal{A} \subseteq \mathcal{A}'$. In that case, we mark $D \subseteq D'$. The augmentation is *valid* if every feasible execution in D is also a feasible execution in D'. Meaning, the augmentation does not affect the state connectivity. In the context of D', we refer to D as the *base domain*, and to actions in $\mathcal{A}$ as *atomic actions*.

The added actions and transitions in the augmented domain symbolize multi-action transitions in the base domain. Following prior art, we use the term *action refinement* [32] to name the process of realizing a symbolic action in the augmented domain into a sequence of atomic actions in the base domain that will transition between the same start and end states. Accordingly, refining a plan corresponds to refining its symbolic actions into a base plan that transitions through all the states the original plan would.

This leads us to suggest a novel planning scheme: instead of planning directly in a base domain, try to augment it and plan in the augmented domain; then, refine the resulting "high-level plan" into a "base plan," and return it. If we perform the augmentation wisely, the transitions added to the domain can act as "shortcuts" in the search space.

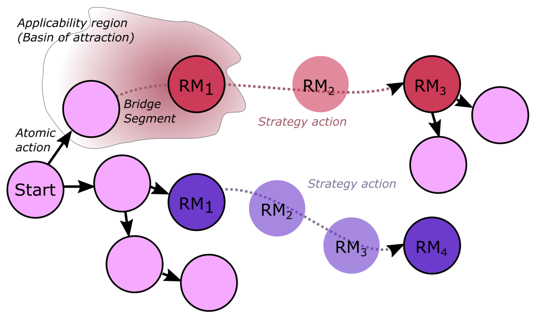

Fig. 2: Search-based planning in the strategy-augmented domain. Circles represent states, solid arrows represent atomic actions, and dotted arrows represent strategy actions—symbolic transitions through the strategy's Road Map (RM), to be refined into a sequence of atomic actions. The end state of a strategy action is known prior to refinement, allowing us to fast-track through the search space.

Combining the previous two definitions, we can consider augmenting our planning domain D with strategy-actions derived from $\mathcal{L}ibrary$. Conveniently, strategy-actions not only serve as shortcuts to accelerate the search in the augmented domain, but also provide a road map of intermediate states, to accelerate their own refinement. Such augmentation would allow us to plan with strategies and actions indistinguishably.

To allow us to discover useful strategies quickly, with less exploration, we may consider each strategy-action K_p applicable from multiple states, beyond its inherent start state $\mathsf{K}_p.RRM[1]$. Since a strategy-action is road-map-based, applying it from a different state $\boldsymbol{S}$ simply means extending the encapsulated road map with one additional state, connected via a "bridge segment." Accounting for this extension will occur seamlessly during the strategy-action refinement. In that case, each strategy-action symbolizes multiple transitions, indicating its applicability from multiple states. Since RRM is fixed in space regardless of this $\boldsymbol{S}$, every strategy-action K_p in the augmented domain acts as a "funnel," where $\mathcal{A}pplic\mathcal{R}eg(\mathsf{K}_p)$ marks its "basin of attraction" (see Fig. 2). Finally, we formulate the strategy-augmented planning problem:

`given:`	a task T, a domain D, a start state $\boldsymbol{S}_{\text{start}}$, and an abstract strategy $\mathcal{L}ibrary$
`find:`	A high-level plan $\boldsymbol{P}'$ in D' A base plan $\boldsymbol{P}$ in D
`s.t.:`	D' is augmentation of D using $\mathcal{L}ibrary$, $\boldsymbol{P}$ is a refinement of $\boldsymbol{P}'$, $\boldsymbol{P}'$ induces a feasible execution from $\boldsymbol{S}_{\text{start}}$ in D', $\boldsymbol{P}$ induces a feasible execution from $\boldsymbol{S}_{\text{start}}$ in D, Applying $\boldsymbol{P}$ from $\boldsymbol{S}_{\text{start}}$ completes the task.

Our goal is to solve this problem and return a feasible, task-satisfying plan $\boldsymbol{P}$ as fast as possible. Search-based planning in the strategy-augmented domain is depicted in Fig. 2.

IV. Understanding Strategy Affordances

Affordance is a circumstantial incentive to choose a certain course of action, which expresses the predicted goodness of match between the course of action and the agent's objective, in the context of the process history, the world the agent operates in, inherent qualities of this course of action, and possibly other subjective preferences. In our case, the agent's goal is to achieve a task-satisfying plan with minimal effort. Hence, the affordance measures we consider are based on the predicted effort required to include a (strategy-)action in our plan, in order to minimize the need for additional planning. Unlike reward or cost, which are calculated in retrospect, affordance is determined before action evaluation, predictively. The importance of affordance is more prominent when considering actions may have a non-trivial and varying evaluation effort, as is the case with strategy-actions. Affordance can also be measured for atomic actions by considering them "redundant strategies" that need no refinement.

A. *Affordance Scores*

As we suggest, affordance can be practically modeled as a numerical vector, where each element represents the strength of a specific incentive. Also, as implied, these incentives are circumstantial and conditioned on the context of application, including the current state, goals, and history. As a convention, we define the affordance scores to be non-negative numbers, where zero represents the "best" affordance value. As mentioned, our goal is to minimize the planning effort. The affordance models we shall define rely on an agent's internal models of distance in the state space, which should express the predicted effort to plan a transition between states; we mark these models as $\texttt{eff}_p$ and $\texttt{eff}_r$, for *p*lanning in the augmented domain and planning in the original domain (i.e., *r*efinement), respectively. Many planning domains have intuitive distance measures that can be utilized for this cause. For example, in PDDL-based domains, we may consider the "edit distance" (number of disagreeing predicates); in geometric domains we may consider a euclidean metric. More specialized models can be learned for specific domains. Next, we provide several exemplary affordance scores for strategy-actions, which aim to encapsulate the different effects an action selection would have on planning efficiency.

1) Affordance granted by the current state: This score measures the proximity of our current state to a state from which execution of the strategy can start:

$$\texttt{startAff}(\mathsf{K}_p \mid \boldsymbol{S}) \doteq \texttt{eff}_r\left(\boldsymbol{S},\, \mathsf{K}_p.RRM[1]\right). \tag{4}$$

This accounts for the "length" of the bridge segment from our current state to the first state in the strategy's road map.

2) Affordance granted inherently by the strategy: This score measures the effort for planning a path traversing through the strategy's road map. Meaning, this score measures the predicted effort required for refining this strategy:

$$\texttt{strategyAff}(\mathsf{K}_p) \doteq \left\| \begin{bmatrix} \texttt{eff}_r\left(RRM[1], RRM[2]\right) \\ \vdots \\ \texttt{eff}_r\left(RRM[end-1], RRM[end]\right) \end{bmatrix} \right\|_l . \tag{5}$$

We may choose the $l = 1$ or the $l = \infty$ norms.

3) Affordance granted by the task: This score quantifies how much application of the strategy advances us towards task completion, considering the history and progress thus far. For reach-avoid tasks with a single standing goal region $T.\mathcal{S}_{\text{goal}}$, as in a classical task, this score would simply measure the effort to plan a path to this goal (as long as the road map does not compromise the "avoid" constraints):

$$\texttt{taskAff}(\mathsf{K}_p \mid T, \textit{prefix}^{\boldsymbol{S}}) \doteq \begin{cases} \texttt{eff}_p\left(RRM[end], T.\mathcal{S}_{\text{goal}}\right) & RRM \cap T.\mathcal{S}_{\text{avoid}} = \emptyset \\ \infty & \text{else} \end{cases}, \tag{6}$$

where $\textit{prefix}^{\boldsymbol{S}}$ is the planning history that led to the current state $\boldsymbol{S}$. If case of multiple goals, we may measure the norm of efforts to reach from the strategy's end state to each of the standing goal regions, those not satisfied by $\textit{prefix}^{\boldsymbol{S}}$.

The affordance scores presented (and others) of a strategy-action K_p can be assembled into an *affordance vector*, $\texttt{aff}\left(\mathsf{K}_p \mid \boldsymbol{S}, T, \textit{prefix}^{\boldsymbol{S}}\right)$. Note that this vector is defined for a specific grounding. As previously stated, we can use the affordance vector to define the applicability region $\mathcal{A}\textit{pplic}\mathcal{R}\textit{eg}$ of K_p, i.e., the set of states from which K_p is applicable, or its *basin of attraction*. We do so by choosing an affordance threshold vector $\overline{\texttt{aff}}$:

$$\begin{aligned} &\mathcal{A}\textit{pplic}\mathcal{R}\textit{eg}(\mathsf{K}_p \mid T, \textit{prefix}^{\boldsymbol{S}}) \doteq \\ &\quad \Big\{\boldsymbol{S} \in \mathcal{S} \mid \texttt{aff}\left(\mathsf{K}_p \mid \boldsymbol{S}, T, \textit{prefix}^{\boldsymbol{S}}\right) \leq \overline{\texttt{aff}} \wedge \\ &\qquad\qquad \mathsf{K}_p.RRM[1] \text{ is reachable from } \boldsymbol{S}\Big\}. \end{aligned} \tag{7}$$

This idea (visualized in Fig. 4) conveys a significant, conceptual, and innovative change of view, and means that applicability of actions is not predefined in the domain, but dynamically changes throughout the planning process.

V. Solving the Planning Problem

Formulating the problem of planning with abstract strategies as planning in an augmented domain allows to solve it with existing task planners. Indeed, if our base domain D is PDDL-specified, we can easily encode strategy-actions as PDDL-compliant, parametric actions with "preconditions" and "effect," where the preconditions encode the applicability region, and the effect encapsulates the road map. This makes strategy grounding and refinement implicit sub-routines of precondition evaluation. Nevertheless, this solution approach ignores the unique properties of the strategy-augmented planning domain. We hence suggest several practical concepts to improve solution efficiency. These suggestions can be summarized into a lazy, affordance-directed search algorithm[2], or integrated into existing planning algorithms individually.

[2]As provided in `http://khen.io/icra24appendix.pdf`

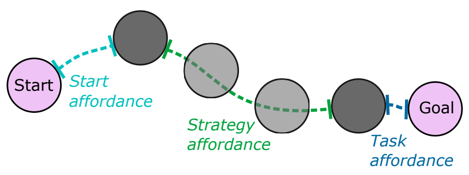

Fig. 3: Illustration of the different affordance scores of a strategy-action, as measured using its road map, which is fixed in space, regardless of the start and goal states.

A. Encoding and Dynamic Grounding of Parametric Actions

Preconditions are often not a practical encoding for the parameter-dependent applicability region.

1) Parameter filtering and dynamic grounding: For PDDL actions, preconditions are used to specify from which states it is applicable. During planning, each action's preconditions are evaluated from the current state, to find which actions are applicable from it. When considering *parametric* actions, standard planners (e.g., FD [9]) begin the planning process by performing an a-priori full grounding of each parametric action (i.e., explicit assignment of all possible parameter values, by enumerating over the parameter space) to yield a set of non-parametric grounded actions. However, when considering complex domains, neither the pre-grounding nor the explicit precondition evaluation are practical.

Hence, we now suggest an alternative to parametric preconditions: defining a state-dependent parameter space, according to which we can perform on-demand grounding—*to only yield the grounded actions applicable from that state*. Practically, for each parametric action, this parameter space can be specified as the solution space of a state-dependent constraint satisfaction problem, which we call *the parameter filter*. Logically, instead of specifying a state-filtering rule for each parametric action, we specify a parameter-filtering rule for each state. Since such parameter filtering and grounding is based on the current state, it should be performed *dynamically*. This formulation allows us to efficiently rule out invalid state-parameter combinations, even without explicit query.

2) Affordance-directed dynamic grounding: Affordance defines the applicability of each strategy-action; hence, for such actions, the grounding is *affordance-directed*. The parameter filter of a strategy-action can be defined as:

given: an abstract strategy $\boldsymbol{K} = (ARM, AK) \in \mathcal{L}ibrary$,
an affordance threshold vector $\overline{\texttt{aff}}$,
a start state $\boldsymbol{S}_{\text{start}}$, a task T, a domain D,
and a plan $prefix^{\boldsymbol{S}_{\text{start}}}$,

find: a strategy-action K_p,
affordance score vector $\boldsymbol{V}$,

s.t.: $p \in \boldsymbol{K}.AK.\mathcal{P}^{ARM}$ (i.e., valid reconstruction),
$\mathsf{K}_p.RRM = \texttt{reconst}_p(\boldsymbol{K}.ARM)$,
$\mathsf{K}_p.RRM \subseteq D.\mathcal{S}$ (i.e., RRM in state space),
$\mathsf{K}_p.RRM$ is feasible in D,
$\mathsf{K}_p.RRM[1]$ is reachable from $\boldsymbol{S}_{\text{start}}$,
$\boldsymbol{V} = \texttt{aff}\left(\mathsf{K}_p \mid \boldsymbol{S}_{\text{start}}, T, prefix^{\boldsymbol{S}_{\text{start}}}\right)$,
$\boldsymbol{V}[i] \leq \overline{\texttt{aff}}[i], \forall i$,

For each state $\boldsymbol{S}$ and abstract strategy $\boldsymbol{K}$, the solution space of the filter problem contains all the corresponding strategy-actions whose $\mathcal{A}pplic\mathcal{R}eg$ contains $\boldsymbol{S}$. To understand overall which strategy-actions are applicable from $\boldsymbol{S}$, we should solve this for every abstract strategy. We refer to this process as *strategy matching*. The solution to each matching problem can most simply be found by feeding the problem to an automated constraint satisfier (e.g., Z3 [33]). Conveniently, solving this problem allows to simultaneously (i) understand which groundings of the parametric strategy-action are applicable from the state; (ii) perform the grounding by calculating the strategy-action's RRM and effect; and (iii) calculate the grounded strategy-action's affordance, which can be used to guide the search.

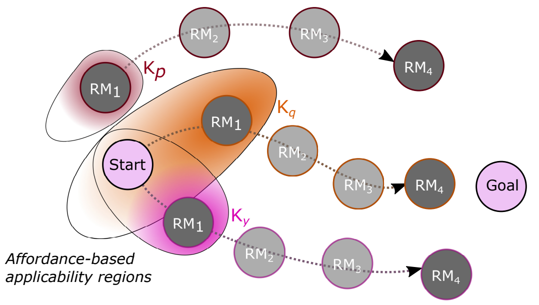

Fig. 4: Different strategy-actions (groundings) of an abstract strategy. Setting a threshold over a strategy-action's affordance score vector defines its applicability region. Multiple groundings of the same abstract strategy can be applicable from the same state. To limit the branching factor during forward search, we can measure the (affordance-based) "strength" of attraction to each grounding, find the optimal one (here, K_q), and eliminate the rest.

B. Optimistic Planning with Lazy Strategy Refinement

Naively using the aforementioned encoding(s) would require us to refine each suggested grounding of a parametric strategy-action (to validate its feasibility) before declaring it is applicable and continuing the strategy sequencing. Meaning, in this scheme, which we refer to as "cautious," planning in the augmented domain (for strategy sequencing) and planning in the base domain (for strategy refinement) are interleaved. Unfortunately, this might cause us to unnecessarily refine strategy-actions that would not be a part of the final plan.

Instead, we may recall that strategies allow us to know their "effect" even without explicitly specifying a realizing action sequence. This property allows us to plan the steps following a strategy-action, even without refining it. Thus, instead of interleaved cautious planning, a natural way to solve the problem is to separate it into planning in two planning levels: long horizon "high-level" planning in the augmented domain to create a plan skeleton, followed by multiple "base level" planning sessions, for filling up the gaps (by refining the strategies selected in that high level plan). We refer to this bi-level scheme as "optimistic planning," as during the high-level planning, we implicitly assume all selected strategies are feasible, and perform "lazy" refinement only after finding a solution. If a refinement failure occurs, instead of triggering a completely new planning process in the augmented domain, we can go back to the original planning graph, truncate the edge of the failed transitions, and resume the planning process. Regardless of the scheme, each of the refinement problems may be solved using any user-selected planning algorithm.

C. Using Affordances to Guide the High-Level Plan Search

To grow the planning graph (see Fig. 2), we need to determine guidelines for node selection and expansion.

1) Node selection: Many prominent planners (e.g., FF [10]) rely on a "distance to goal" heuristic, to select the most promising node for expansion. Yet, this would not be appropriate when planning with strategies. First, our goal is to minimize the planning effort, and not find an optimal plan; hence, a "distance to goal" heuristic should be more generally translated to "effort to completion," e.g., as measured by the task affordance (Eq. (6)). Second, as explained, if we consider the graph of an "optimistic" planner, edges in the prefix of a node might represent transitions (strategy-action segments) not yet refined. Hence, when choosing a node for expansion, we not only want to account for the effort to complete the task from it, but also the effort to refine its prefix. Conveniently, if we consider the selected node to represent the end-state $\boldsymbol{S}_{\text{end}}$ of an action K_p applied from $\boldsymbol{S}_{\text{start}}$, the effort to refine its prefix can be calculated incrementally, from the effort to refine the prefix of $\boldsymbol{S}_{\text{start}}$ and the segments of K_p. With that, the node score can easily be expressed as

$$\texttt{nodeScore}(\boldsymbol{S}_{\text{end}}) = \left\| \begin{bmatrix} \texttt{eff}_r\left(\textit{prefix}^{\boldsymbol{S}_{\text{end}}}\right) \\ \texttt{taskAff}(\mathsf{K}_p) \end{bmatrix} \right\|_l = \left\| \begin{bmatrix} \texttt{eff}_r\left(\textit{prefix}^{\boldsymbol{S}_{\text{start}}}\right) \\ \texttt{aff}\left(\mathsf{K}_p\right) \end{bmatrix} \right\|_l. \quad (8)$$

2) Node expansion and optimized grounding: As explained, we use affordances to define the parameter filter, determine the applicability, and guide the grounding of strategy-actions from each state; by such, we can say that node expansion is also guided by affordance. Now, consider that multiple groundings of the same abstract strategy $\boldsymbol{K}$ are applicable from the same state $\boldsymbol{S}$. Based on the former conclusions, when node selection is guided by affordance, we would always prefer the node at the end of the strategy-action with minimal $\|\texttt{aff}\|_l$, over the end-node of any other grounding of $\boldsymbol{K}$ from $\boldsymbol{S}$. Supported by this claim, we can choose to perform restricted matching and, instead of considering all possible groundings, only extend a single edge, corresponding to the optimal grounding of $\boldsymbol{K}$. This means adapting the parameter filter to an "optimized version":

$$\texttt{argmin} \quad \left\| \texttt{aff}\left(\mathsf{K}_p \mid \boldsymbol{S}_{\text{start}}, T, \textit{prefix}^{\boldsymbol{S}_{\text{start}}}\right) \right\|_l, \quad (9)$$

under the same conditions provided in the original filter. This is equivalent to performing greedy edge elimination after each node expansion. While this choice might cause us to miss potentially useful strategy-actions, it also helps controlling the search branching factor. It has no effect on completeness.

VI. Experimental Results

We implemented a system combining the strategy learning algorithm from [5] with our suggested bi-level strategy-based planning algorithm. We then evaluated the prominent task planner Fast Downward (FD) [34], against our planner (which relied on FD for strategy refinement), on a set of problems in the `blockworld` domain from the 2nd International Planning Competition [35]. These involve permutation of blocks in towers of sizes up to 8 blocks—a starting configuration of blocks in a tower must be reconfigured into a different configuration of the same size. We solved 4 sets of test

TABLE I: Results on tower rearrangement problems, with increasing numbers of towers to be rearranged. Planning with strategies results in higher success rate and lower median planning time than Fast Downward [34] as the number of towers increases.

	FD		Planning with Strategies (FD for refinement)				
n	Solved (%)	Time (s)	Solved (%)	Time (s)	Extract (s)	Strategies	Actions
1	**100%**	**0.19**	**100%**	1.33	0.008	1.00	0.88
2	50%	43.74	**92%**	**2.88**	0.008	1.46	1.39
3	44%	120.0	**54%**	**31.41**	0.006	1.26	2.04
4	20%	**120.0**	**38%**	**120.0**	0.007	1.79	1.53

problems, increasing the number of towers to reconfigure in each problem from 1 tower to 4 towers. Additionally, the test problems introduced up to four irrelevant objects that can appear on the table or on top of starting towers. We began each set of experiments by building a library of abstract strategies from solutions of 10 random single-tower training problems. Example problems are depicted in Fig. 1.

Table I reports the median values for planning times over 50 randomly generated test problems. Planner timeout is 120 seconds. The times for planning with strategies contain the time used for strategy matching and refinement. Indeed, thanks to the problem decomposition it induces, planning with strategies allowed us to utilize FD much more effectively, as our planning resulted in a higher success rate over all problems. Despite performing slower than FD on the single tower case, strategies outperforms FD in terms of median planning time as the number of towers increases, indicating the increasing difficulty of the problems as well as the power of strategies to accelerate search. The "Strategies" and "Actions" columns indicate the median numbers of strategy-actions and of atomic-actions used in the successful plans, respectively. As problems grow more difficult, more strategies and actions are used, demonstrating the ability of strategy-based planner to generalize and compose to solve more complex problems. Note that the initial learning phase was near instantaneous ("Extract" column), taking only milliseconds to generate strategies from training problems.

VII. Conclusion

In this paper we developed a novel *theoretical* planning formalism, planning in a strategy-augmented domain, allowing us to achieve *practical* acceleration of long-horizon task planning. This formalism relies on our recently-introduced framework [3–5] for learning "abstract strategies." Previous work conceptually proved that strategies can generalize and be used to solve new tasks. The approach presented here extends it, to supports arbitrary composition of strategies and actions. To allow effective planning with strategies, we also contributed a novel formulation for the concepts of "affordances" and (affordance-directed) "dynamic grounding," and incorporated them into a novel algorithm for planning with strategies. Finally, we presented an experimental validation of the approach, showing that it allows an agent to take advantage of its planning experience and generalize it in real-time to solve new, harder tasks, which were *not solvable using a state-of-the-art task planner*. Overall, this paper is a stepping stone for extending the "abstract strategies" framework to support life-long, continually-improving planning agents.

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. "Integrated Task and Motion Planning". In: *Annu. Rev. Control Robot. Auton. Syst.* 4.1 (2021), pp. 265–293.

[2] T. Bylander. "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 69.1-2 (Sept. 1994), pp. 165–204.

[3] K. Elimelech, L. E. Kavraki, and M. Y. Vardi. "Automatic Cross-domain Task Plan Transfer by Caching Abstract Skills". In: *Algorithmic Foundations of Robotics XV*. College Park, MD, USA, June 2022, pp. 470–487.

[4] K. Elimelech, L. E. Kavraki, and M. Y. Vardi. "Efficient Task Planning Using Abstract Skills and Dynamic Road Map Matching". In: *International Symposium on Robotics Research*. Geneva, Switzerland, Sept. 2022.

[5] K. Elimelech, L. E. Kavraki, and M. Y. Vardi. "Extracting Generalizable Skills from a Single Plan Execution Using Abstraction-Critical State Detection". In: *IEEE International Conference on Robotics and Automation*. London, United Kingdom, May 2023.

[6] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning and Acting*. Cambridge: Cambridge University Press, 2014.

[7] E. Karpas and D. Magazzeni. "Automated Planning for Robotics". In: *Annu. Rev. Control Robot. Auton. Syst.* 3.1 (May 2020), pp. 417–439.

[8] D. McDermott. "PDDL—The Planning Domain Definition Language". In: *AIPS Planning Competition*. AIPS Planning Competition. 1998.

[9] M. Helmert. "The Fast Downward Planning System". In: *jair* 26 (July 2006), pp. 191–246. arXiv: `1109.6051`.

[10] J. Hoffmann and B. Nebel. "The FF Planning System: Fast Plan Generation through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 263–312. arXiv: `1106.0675`.

[11] J. Rintanen. "Madagascar: Scalable Planning with SAT". In: *Proceedings of the 8th International Planning Competition (IPC-2014)*. Vol. 21. 2014, pp. 1–5.

[12] H. A. Kautz and B. Selman. "Planning as Satisfiability". In: *ECAI*. Vol. 92. 1992, pp. 359–363.

[13] B. Marthi, S. J. Russell, and J. Wolfe. "Angelic Semantics for High-Level Actions". In: *ICAPS*. 2007, pp. 232–239.

[14] A. Coles, M. Fox, and A. Smith. "Online Identification of Useful Macro-Actions for Planning." In: *ICAPS*. 2007, pp. 97–104.

[15] I. Georgievski and M. Aiello. "HTN planning: Overview, comparison, and beyond". In: *Artificial Intelligence* 222 (2015), pp. 124–156.

[16] J. Jeon, H.-r. Jung, T. Luong, F. Yumbla, and H. Moon. "Combined Task and Motion Planning System for the Service Robot Using Hierarchical Action Decomposition". In: *Intell. Serv. Robot.* 15.4 (Sept. 2022), pp. 487–501.

[17] V. Shivashankar, U. Kuter, D. Nau, and R. Alford. "A Hierarchical Goal-Based Formalism and Algorithm for Single-Agent Planning". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 2012, pp. 981–988.

[18] R. S. Sutton, D. Precup, and S. Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112.1 (Aug. 1999), pp. 181–211.

[19] A. Bagaria and G. Konidaris. "Option Discovery Using Deep Skill Chaining". In: *International Conference on Learning Representations*. Mar. 2020.

[20] Y. Jinnai, D. Abel, D. E. Hershkowitz, M. L. Littman, and G. Konidaris. "Finding Options That Minimize Planning Time". In: *International Conference on Machine Learning*. 2019, p. 19.

[21] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. "Robot Learning from Demonstration by Constructing Skill Trees". In: *The International Journal of Robotics Research* 31.3 (Mar. 2012), pp. 360–375.

[22] S. Cheng and D. Xu. *Guided Skill Learning and Abstraction for Long-Horizon Manipulation*. Oct. 2022. arXiv: `2210.12631 [cs]`.

[23] W. Vega-Brown and N. Roy. "Admissible Abstractions for Near-optimal Task and Motion Planning". In: *International Joint Conference on Artificial Intelligence*. 2018, pp. 4852–4859.

[24] T. E. Horton, A. Chakraborty, and R. S. Amant. "Affordances for Robots: A Brief Survey". In: *AVANT* 3.2 (Dec. 2012).

[25] P. Ardón, È. Pairet, K. S. Lohan, S. Ramamoorthy, and R. P. A. Petrick. *Affordances in Robotic Tasks – A Survey*. Apr. 2020.

[26] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett. "Long-Horizon Manipulation of Unknown Objects via Task and Motion Planning with Estimated Affordances". In: *ICRA*. arXiv, 2022.

[27] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, and M. Yan. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv:2204.01691 [cs]* (Apr. 2022). arXiv: `2204.01691 [cs]`.

[28] C. Agia, T. Migimatsu, J. Wu, and J. Bohg. *TAPS: Task-Agnostic Policy Sequencing*. Oct. 2022. arXiv: `2210.12250 [cs]`.

[29] T. McMahon, O. C. Jenkins, and N. Amato. "Affordance Wayfields for Task and Motion Planning". In: *2018 IEEE/RSJ International Conference on Intelligent*

Robots and Systems (IROS). Madrid: IEEE, Oct. 2018, pp. 2955–2962.

[30] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei. "Deep Affordance Foresight: Planning Through What Can Be Done in the Future". In: *arXiv:2011.08424 [cs]* (Nov. 2020). arXiv: `2011.08424 [cs]`.

[31] I. Awaad, G. K. Kraetzschmar, and J. Hertzberg. "Affordance-Based Reasoning in Robot Task Planning". In: *Planning and Robotics (PlanRob) Workshop ICAPS-2013*. 2013.

[32] R. Gorrieri and A. Rensink. "Chapter 16 - Action Refinement". In: *Handbook of Process Algebra*. Ed. by J. Bergstra, A. Ponse, and S. Smolka. Amsterdam: Elsevier Science, 2001, pp. 1047–1147.

[33] L. de Moura and N. Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and J. Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[34] M. Helmert. "The fast downward planning system". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246.

[35] D. Long and M. Fox. "The 3rd international planning competition: Results and analysis". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 1–59.

Appendix I
Detailed Planning Algorithm

Algorithm 1: Planning in the strategy-augmented domain: detailed algorithm.

```
1  Algorithm planWithAbstractStrategies(start state $S_{start}$, goal region $\mathcal{S}_{goal}$, domain $D$, Library of abstract strategies, task Planner)
                                           // plan in the augmented domain
2      $queue \leftarrow []$                                  // node queue
       // struct: 1. state, 2. parent node, 3. incoming edge, 4. prefix effort, 5. remaining effort
3      $newNode \leftarrow [S_{start}, \textbf{Null}, \textbf{Null}, 0, \infty]$
4      $queue$.insert($newNode, \infty$)                // insert by score
5      while $queue$.isNotEmpty() do                   // go over state nodes
6          $curNode \leftarrow queue$.pop()
                                                 // extend atomic-actions
7          forall action $a \in D.\mathcal{A}$ do
8              if $a$.precond($curNode.S$) =True then
9                  $S_{end} \leftarrow a(curNode.S)$
10                 $preEffort \leftarrow curNode.preEffort$
11                 $remEffort \leftarrow \texttt{eff}_p(S_{end}, \mathcal{S}_{goal})$
12                 $newNode \leftarrow [S_{end}, curNode, a, preEffort, remEffort]$
13                 if $S_{end} \in \mathcal{S}_{goal}$ then          // reached goal
14                     goto 29
15                 $queue$.insert($newNode, preEffort + remEffort$)
16         end
                                  // dynamically ground abstract strategies
17         forall abstract strategy $K \in \mathcal{L}ibrary$ do
18             $strategyAct$, startAfford, strategyAfford, taskAfford
                   $\leftarrow$ matchOptStrategy($K, D, curNode.S_{start}$)
19             if $strategyAct$ is not Null then
20                 $S_{end} \leftarrow strategyAct.RRM[end]$
21                 $preEffort \leftarrow curNode.preEffort +$ startAfford + strategyAfford
22                 $remEffort \leftarrow$ taskAfford
23                 $newNode \leftarrow [S_{end}, curNode, strategyAct, preEffort, remEffort]$
24                 if $S_{end} \in \mathcal{S}_{goal}$ then          // reached goal
25                     goto 29
26                 $queue$.insert($newNode, preEffort + remEffort$)
27         end
28     end
       // potential High-Level Plan found, refine
29     $HLP \leftarrow$ trace $newNode$ to root, and return sequence of actions
30     $baseplan \leftarrow []$
31     $FAILURE \leftarrow$ False
32     forall action $a \in HLP$ do
33         if $a$ is strategy-action then
34             $subplan \leftarrow$ refine($a, S_{start}$ of $a$, Planner)
35             if $subplan$ is not Null then
36                 $baseplan \leftarrow [baseplan, subplan]$
37             else                                  // strategy unfeasible
38                 $FAILURE \leftarrow$ True; break
39         else if $a$ is atomic-action then
40             $baseplan \leftarrow [baseplan, a]$
41     end
42     if not $FAILURE$ then          // refinement done, return
43         return $baseplan$
44     else                                   // refinement failed
45         truncate sub-graph following failed edge
46         goto 5         // continue high-level planning
```

Algorithm 2: Strategy refinement: detailed procedure.

```
1  Procedure refine(strategy-action $\mathsf{K}_p$, start state $S_{start}$, task Planner)
2      $RM \leftarrow \mathsf{K}_p.RRM$
3      if not ($RRM[1] = S_{start}$) then
                              // augment RM with bridge segment
4          $RM \leftarrow [S_{start}, RM]$
5      $subplans \leftarrow []$
6      for $i$ from 1 to len($RM$) $- 1$ do
                                          // refine RM segment
7          $subplans[i] \leftarrow Planner$.plan($RRM[i], RRM[i+1]$)
8          if $subplans[i]$ = Null then
9              return Null         // segment unfeasible, abort
10     end
11     return $[subplans[1], \ldots, subplans[l-1]]$
```

Appendix II
Abstraction Keys

Next, we formulate two abstraction keys to be used in our experiments. A version of these keys was previously presented in [3], and is now restated to fit propositional states, as used in PDDL (and in our experiments).

A. Attention

This abstraction key allows us to focus our attention only on a subset of the propositions (statements) in a state.

The parameter space of a state S is hence defined as

$$\mathcal{P}_S \doteq \{(\text{subset of the propositions in } S)\}. \tag{10}$$

The projection and reconstruction functions are defined as:

$$\texttt{project}_p(S) \doteq \text{remove the set } p \text{ of props. from } S, \tag{11}$$

$$\texttt{reconst}_p(\xi) \doteq \text{add the set } p \text{ of props. to } S. \tag{12}$$

This abstraction key can be used to compactly cache a strategy's road map as a trace of "smaller" states, containing only the state variables which are affected by the strategy. This key is hence useful for learning "local strategies," from sequences of actions that only affect a portion of the state. For example, re-positioning of an object is a often a local strategy, as it does not affect the positions of the other objects.

B. Symbol stripping

This abstraction key allows us to focus on function over form, by removing repeating symbols from states in a state trace. This key can be used, e.g., to learn an object stacking strategy with no explicit reference to the specific object type.

The parameter space of S is hence defined as

$$\mathcal{P}_S \doteq \{\text{symbols that appear in } S\}. \tag{13}$$

The projection and reconstruction functions are defined as:

$$\texttt{project}_p(S) \doteq \text{replace each instance of } p \text{ in } S \text{ with } *, \tag{14}$$

$$\texttt{reconst}_p(\xi) \doteq \text{replace each instance of } * \text{ in } \xi \text{ with } \xi. \tag{15}$$

C. Combination

We may also consider a “combination” abstraction key, by composing the respective projection and reconstruction functions from the two keys presented. This should allow us to perform both transformations on a state at once.

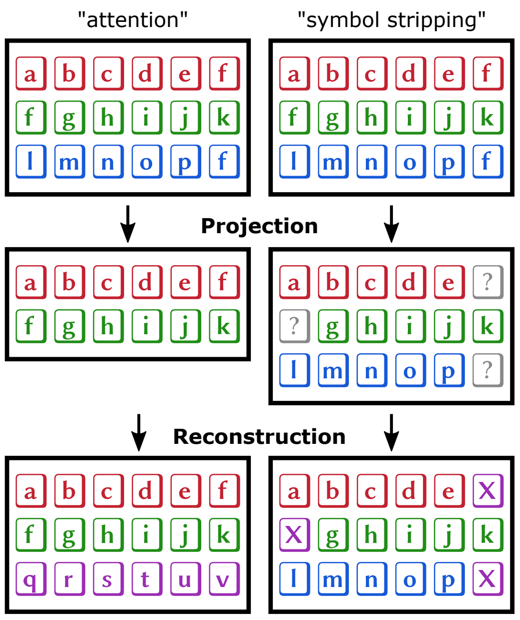

Fig. 5: State transformation using abstraction keys. The top row shows the original state, the middle row shows the abstract state, and the bottom row shows the reconstructed and transformed state.