Goal Space Abstraction in Hierarchical Reinforcement Learning via Set-Based Reachability Analysis

Mehdi Zadem[1], Sergio Mover[1], Sao Mai Nguyen[2,3]

***Abstract*— Open-ended learning benefits immensely from the use of symbolic methods for goal representation as they offer ways to structure knowledge for efficient and transferable learning. However, the existing Hierarchical Reinforcement Learning (HRL) approaches relying on symbolic reasoning are often limited as they require a manual goal representation. The challenge in autonomously discovering a symbolic goal representation is that it must preserve critical information, such as the environment dynamics. In this paper, we propose a developmental mechanism for goal discovery via an emergent representation that abstracts (i.e., groups together) sets of environment states that have similar roles in the task. We introduce a Feudal HRL algorithm that concurrently learns both the goal representation and a hierarchical policy. The algorithm uses symbolic reachability analysis for neural networks to approximate the transition relation among sets of states and to refine the goal representation. We evaluate our approach on complex navigation tasks, showing the learned representation is interpretable, transferrable and results in data efficient learning.**

I. Introduction

Symbol emergence is key for developmental learning to tackle the curse of dimensionality and scale up to open-ended high-dimensional sensorimotor space, by allowing symbolic reasoning, compositionality, hierarchical organisation of the knowledge, etc. While symbol emergence has been recently investigated for the sensor data, action symbolization can lead to a repertoire of various movement patterns by bottom-up processes, which can be used by top-down processes such as compostion to form an action sequence, planning and reasoning for more efficient learning, as reviewed in [1]. Sensorimotor symbol emergence thus is key to scaling up low-level actions into complex actions for open-ended learning, using compositionality and hierarchy.

Action hierarchies are the core idea of Hierarchical Reinforcement Learning (HRL) [2] that decomposes a task into easier subtasks. In particular, in Feudal HRL [3] a high-level agent selects subgoals that a low-level agent learns to achieve. The performance of Feudal HRL depends on the "hierarchical division of the available state space" [3], the representation of the goals that the high level agent uses to decompose a task. Yet, only few algorithms learn it automatically [4], while others either use directly the state space [5] or manually provide a representation [6], [7]. The issue of symbol emergence within hierarchical reinforcement learning has little been investigated.

[1]LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris, zadem@lix.polytechnique.fr, sergio.mover@lix.polytechnique.fr,

[2]Flowers Team, U2IS, ENSTA Paris, Institut Polytechnique de Paris & Inria, [3]IMT Atlantique, Lab-STICC, nguyensmai@gmail.com,

So far there is "no computational model that can reproduce the dynamics of symbol emergence in the real world itself" [1]. Learning online an abstraction of the sensorimotor space grounded by the experienced data is all the more challenging if the abstraction needs to be amenable for transfer learning in a new environment and to be interpretable.

In this paper, we tackle the problem of learning automatically, while learning the policy, a discrete goal representation from continuous observations that expresses the task structure for *data-efficiency*, is *transferrable*, and is *interpretable*.

We introduce a feudal HRL algorithm, GARA (Goal Abstraction via Reachability Analysis), that develops a novel symbolic goal space representation while simultaneously learning a hierarchical policy from exploration data. The representation emerges through a developmental process, gradually gaining precision from a bottom-up manner, by leveraging data acquired from exploration. This discretisation of the environment is used to orient top-down process of the goal-directed exploration, that in turn helps improving policies and this representation. We thus propose a symbolic representation of the goal space emerging dynamically from bottom-up and top-down processes.

This paper focuses on deterministic and reward-sparse environments with continuous state spaces, highlights the emergence of a structured representation, and shows its impact in terms of data efficiency, transferability and interpretability. We propose:

1) A symbolic representation of the goal space (Section II), obtained grouping together sets of similar states according to the state reachability relation, that leads to the emergence of action abstractions that can be chained into action sequences: two states are similar if a low-level policy reaches the same set of similar states. This representation is interpretable as a directed graph, where nodes describe goals and edges a reachability relation among them. To build such representation, the algorithm symbolically analyzes a neural network approximating the state reachability relation, similarly to reachability analysis used in formal verification [8].
2) A two level HRL algorithm, GARA that represent the bottom-up and top-down processes that discover and learn online the goal representation via set-based reachability analysis (Section III).
3) An experimental evaluation (Section IV) demonstrating that the abstract goal representation can be learned in a data-efficient manner. The experiments further validate that the representation of the goals enables transfer learning and is interpretable.

II. Framework

A. Feudal Hierarchical Reinforcement Learning

Reinforcement Learning (RL) algorithms learn to take actions in different states of an environment in order to complete a task. An agent is a Markov Decision Process $M = (\mathcal{S}, A, P, r)$, where $\mathcal{S}$ is a set of states, A is a set of actions, P is a probabilistic transition function, and r is a reward function. At each execution time t, the agent is in a state $s_t \in \mathcal{S}$ and chooses an action a to execute with a policy π (i.e., $a \sim \pi(s_t)$). After executing the action, the agent receives a reward r from the environment. RL algorithms learn a policy π that maximizes the expected future reward. In Hierarchical Reinforcement Learning (HRL), multiple agents operate at different levels of the problem. Higher level agents decompose the initial learning task into subtasks, while lower level agents learn a policy to achieve such sub-tasks, either by further decomposing them into smaller subtasks or by learning concrete actions for solving them.

We adopt the Feudal HRL [3] architecture, where the hierarchy is composed of two levels: a high-level agent $\mathcal{A}^{\text{High}}$ samples a goal g_t from a goal space $\mathcal{G}$. The goal space $\mathcal{G}$ can be either the environment states itself (i.e., $\mathcal{G} = \mathcal{S}$) or an *abstract goal representation*, where a function maps states in $\mathcal{S}$ to goals in $\mathcal{G}$. The high-level agent selects a goal g_t using a high-level policy π_{High} (i.e., $g_t \sim \pi_{\text{High}}(s_t)$). The policy π_{High} is learned to maximize the expected environment "external" reward r^{ext}. Then, the selected goal g_t is communicated to a low-level agent that trains to learn a policy $\pi_{\text{Low}}(s_t, g_t)$ that reaches g_t from the current state s_t. In our framework, we assume the low-level policy is a Universal Value-Function Approximator (UVFA) [9], where the goal itself is as an additional parameter to the policy. In such way, the low-level policy learns different behaviours depending on the goal g_t.

B. Goal Abstraction

We propose a goal abstraction that discretizes the continuous environment state space $\mathcal{S}$ into a finite number of disjoint subsets. Formally, the abstract goal space is a *partition* $\mathcal{G} = \{G_0, \ldots, G_n\}$ of the state space $\mathcal{S}$, that is $\left(\bigcup_{G \in \mathcal{G}} G\right) = \mathcal{S}$ and for all $G, G' \in \mathcal{G}$, $G \cap G' = \emptyset$ if $G \neq G'$. A key intuition is that the partition $\mathcal{G}$ depends on the behavior of the low-level agent and, in particular, on the *reachability relation* $R_k(G') \subseteq \mathcal{S} \times \mathcal{S}$. R_k is such that $(s, s') \in R_k(G')$ if the state s reaches the state $s' \in G'$ after applying the optimal low-level policy $\pi^*_{\text{Low}}(s, G')$ for k-steps. We extend the notation of reachability relation R_k to sets of states G to represent the set of states that G reaches using $R_k(G')$, i.e., $R_k(G, G') = \{s' \in \mathcal{S} \mid \exists s \in G.(s, s') \in R_k(G')\}$. The abstract goal space, the partition $\mathcal{G}$, satisfies the property:

$$\forall\ G, G' \in \mathcal{G}. R_k(G, G') \subseteq G'. \tag{1}$$

This property expresses that all the states in a subset $G \in \mathcal{G}$ have a similar behavior in the environment, reaching states that are also similar (i.e., a bisimulation relation among the environment's states, see, e.g. [10]). A subset G is said *stable* with respect to G' if $R_k(G, G') \subseteq G'$. The main advantage of such goal abstraction is to capture the behavior of the low-level agent, while still decomposing the state space. In fact, the high-level agent will ask the low-level agent to reach target subgoals $s' \in G'$ from an initial subgoal $s \in G$, which should be feasible with respect to the low-level policies.

III. Methodology

A. Algorithm Outline

Goal Abstraction via Reachability Analysis (GARA) (Algorithm 1) is a Feudal HRL algorithm that learns, simultaneously, the agent policies and the goal abstraction $\mathcal{G}$. GARA learns to solve a task specified with a set of initial and target states in an environment E. The algorithm starts with the initial goal set $\mathcal{G} = \{\mathcal{S}\}$ and then iterates for n_{eps} episodes.

In each episode, GARA starts in an initial state s_t in the set $G_s \in \mathcal{G}$ and then selects a target set G_d with the high-level policy π_{High}, which the algorithm learns with the environment's reward signal (r_t^{ext}). The low-level agent trains a UVFA low-level policy π_{Low} for reaching the target set G_d from the state s_t, using as reward a function ($\text{reward}_{\text{Low}}$) of the environment reward (r_t^{ext}) and the reward for reaching the target set G_d ($\mathbb{1}_{G_d}(s_{t+1})$). GARA uses Q-learning for learning the high-level policy (the high-level agent's states are finite), and Deep Q-learning [11] for the low-level policy.

The key insight of GARA is to gradually learn a *stable* goal representation $\mathcal{G}$ (see equation 1), refining the start set G_s of each abstraction edge $(G_s, G_d) \in \mathcal{E}$ that the agent visited after each episode. The algorithm uses the data acquired during training to approximate the *unknown* reachability relation R_k. GARA uses the result of each episode, stored in the memory $\mathcal{D}$ (Line 16), to train the *k-forward model* $\mathcal{F}_k : \mathcal{S} \times \mathcal{G} \to \mathcal{S}$ (line 24), a neural network that approximates the reachability relation R_k. Given a state s_t and a target set G_d, $\mathcal{F}_k(s_t, G_d)$ predicts the state s_{t+k} that the agent reaches after applying the low-level policy $\pi_{\text{Low}}(s_t, G_d)$ for a sequence of k steps. In our implementation, $\mathcal{F}_k$ is a fully connected neural networks and it uses a MSE loss function (see [12] for details). GARA then *refines* the goal abstraction $\mathcal{G}$ with the *Refine* function (line 25).

Observe that, during an episode, GARA may either not reach the target goal set or the initial partition may have a single element (i.e., $\mathcal{G} = \{\mathcal{S}\}$, so any state in $\mathcal{S}$ reaches another state in $\mathcal{S}$). In such cases, the criteria for refining an abstract goal set G is to separate the states in G reached during the episodes from the other ones (using the memory $\mathcal{D}$). GARA uses a standard density-based clustering approach (DBSCAN [13]) to identify a cluster of frequently visited states. Note that, in later episode the algorithm will eventually try to explore the new subset of G containing the unreachable states.

B. Refining Goal Abstraction via Reachability Analysis

a) Refining a Partition: the *Refine* algorithm (Algorithm 2) refines the partition of the abstraction according to equation 1. *Refine* takes as input the current abstraction $\mathcal{G}$, a set $\mathcal{E} \subseteq \mathcal{G} \times \mathcal{G}$ of transitions, and the k-forward model $\mathcal{F}_k$, and computes a new abstraction $\mathcal{G}'$. $\mathcal{G}'$ is a refinement

Algorithm 1 Goal Abstraction via Reachability Analysis

Input: Learning environment E.
Output: Computes the high-level and low-level policies π_{High} and π_{Low}

1: $\mathcal{D} \leftarrow \emptyset$, $\mathcal{G} \leftarrow \mathcal{S}$, episodes $\leftarrow 0$
2: **for** episode $\leq n_{\text{eps}}$ **do**
3: episode $\leftarrow$ episode $+ 1$, $t \leftarrow 0$, $\mathcal{E} \leftarrow \emptyset$
4: $s_{\text{init}} \leftarrow$ initial state from E, $s_t \leftarrow s_{\text{init}}$
5: $G_s \leftarrow G \in \mathcal{G}$ such that $s_t \in G$
6: $G_d \sim \pi_{\text{High}}(G_s)$
7: **while** true **do**
8: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(G_s, G_d)\}$
9: $a_t \sim \pi_{\text{Low}}(s_t, G_d)$
10: $(s_{t+1}, r_t^{\text{ext}}, \text{done}) \leftarrow$ execute the action a_t from s_t in E
11: $r_t^{\text{Low}} = \text{reward}_{\text{Low}}(\mathbb{1}_{G_d}(s_{t+1}), r_t^{\text{ext}})$
12: Update π_{Low} with reward r_t^{Low}
13: **if** not done **then**
14: $s_t \leftarrow s_{t+1}$, $t \leftarrow t + 1$
15: **if** $t \ mod \ k = 0$ or $s_t \in G_d$ **then**
16: Add $(s_{\text{init}}, G_s, s_t, G_d, r_t^{\text{ext}}, \text{done})$ to $\mathcal{D}$
17: Update π_{High} with reward r_t^{ext}
18: $s_{\text{init}} \leftarrow s_t$
19: $G_s \leftarrow G \in \mathcal{G}$ such that $s_t \in G$
20: $G_d \sim \pi_{\text{High}}(G_s)$
21: **else**
22: break the **while** loop and terminate the current episode
23: **if** $\mathcal{G} \neq \mathcal{S}$ **and** G_d is reached in $\mathcal{D}$ **then**
24: Update $\mathcal{F}_k$ with the data from $\mathcal{D}$
25: $\mathcal{G} \leftarrow$ *Refine*$(\mathcal{G}, \mathcal{E}, \mathcal{F}_k)$
26: **else**
27: $\mathcal{G} \leftarrow Cluster(\mathcal{G})$

Algorithm 2 **Refine**$(\mathcal{G},\mathcal{E},\mathcal{F}_k)$

Input: goal abstraction $\mathcal{G}$, set of edges $\mathcal{E}$, and k-forward model $\mathcal{F}_k$
Output: goal abstraction $\mathcal{G}'$ that refines $\mathcal{G}$

1: $\mathcal{G}' \leftarrow \mathcal{G}$
2: **for** $(G_s, G_d) \in \mathcal{E}$ **do**
3: **if** $error(F_k(s \in G_s, G_d))$ stable **then**
4: $(\mathcal{G}_R, \mathcal{G}_U) \leftarrow$ *SplitSet*$(G_s, G_d, \mathcal{F}_k)$
5: $\mathcal{G}' \leftarrow (\mathcal{G}' \setminus \{G_s\}) \cup \mathcal{G}_R \cup \mathcal{G}_U$
6: **return** $\mathcal{G}'$

of the abstraction $\mathcal{G}$ (i.e., $\bigcup_{G \in \mathcal{G}} G = \bigcup_{G \in \mathcal{G}'} G$ and for all $G' \in \mathcal{G}'$ there exists a set $G \in \mathcal{G}$ such that $G' \subseteq G$).

A transition (G_s, G_d) is in the set $\mathcal{E}$ if, in the last episode, the agent reached a state $s_k \in G_d$ from a state $s_0 \in G_s$ executing the low-level policy $\pi_{\text{Low}}(s_0, G_d)$ for k steps. We perform reachability analysis after F_k's error has stabilised for the considered part of the environment. Meaning the low-level policies remain relatively unchanged. Note that, in the the past episode, Algorithm 1 only acquired new data for the transitions $(G_s, G_d) \in \mathcal{E}$, and Algorithm 2 splits each "start" set G_s with the *SplitSet* function (line 4). *SplitSet* computes a set $\mathcal{G}_R$ of reachable and a set $\mathcal{G}_U$ of unreachable sets such that $\mathcal{G}_R = \{G \mid G \subseteq G_s, R_k(G) \subseteq G_d\}$, $\mathcal{G}_U = \{G \mid G \subseteq G_s, R_k(G') \subseteq \overline{G_d}\}$, where $\overline{G_d}$ is the complement of G_d, and $\bigcup_{G \in \mathcal{G}_R \cup \mathcal{G}_U} G = G_s$. Then, the algorithm obtains the refinement $\mathcal{G}'$ replacing in $\mathcal{G}$ the set G_s with the sets from $\mathcal{G}_R$ and $\mathcal{G}_U$. Since representing non-convex sets is computationally expensive, we restrict G to be convex (our implementation uses intervals [14]).

While standard partition refinement algorithms (e.g., [15]) compute the coarsest partition with respect to a relation (e.g., R_k), *Refine* only splits the source set of each edge in the set $\mathcal{E}$. Thus, the refined abstraction $\mathcal{G}'$ may not satisfy the goal

Algorithm 3 **SplitSet**$(G_s,G_d,\mathcal{F}_k)$

Input: start set G_s, target set G_d, and k-forward model $\mathcal{F}_k$
Output: sets of states $\mathcal{G}_R$ and $\mathcal{G}_U$

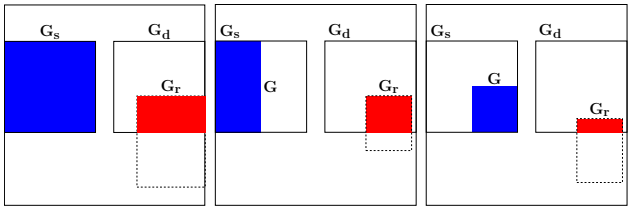
1: $\mathcal{G}_R \leftarrow \emptyset$, $\mathcal{G}_U \leftarrow \emptyset$, $L \leftarrow \{G_s\}$
2: **while** $L \neq \emptyset$ **do**
3: remove an element G from L
4: $G_R \leftarrow$ *FnnReach*$(\mathcal{F}_k, G)$
5: $r \leftarrow \frac{\text{Volume}(G_R \cap G_d)}{\text{Volume}(G_R)}$
6: **if** $r > t_{\text{re}}$ **then** $\mathcal{G}_R \leftarrow \mathcal{G}_R \cup \{G\}$
7: **else if** $r < t_{\text{unre}}$ **then** $\mathcal{G}_U \leftarrow \mathcal{G}_U \cup \{G\}$
8: **else then** $(G', G'') \leftarrow$ *Split*(G),$L \leftarrow L \cup \{G', G''\}$
9: **return** $(\mathcal{G}_R, \mathcal{G}_U)$

abstraction criteria in all the sets (e.g., suppose the algorithm splits G_s in two subsets, G'_s and G''_s, and that in $\mathcal{G}$ there is a set G_{pre} that can reach G_s; while G_{pre} may not be stable, *Refine* does not split it). In practice, eagerly splitting a set is counterproductive since we approximate the relation R_k with $\mathcal{F}_k$. If needed, GARA will refine the set in a later episode.

b) Splitting a Single Set via Reachability Analysis: *SplitSet* (Algorithm 3) computes the subsets of G_s that can reach the set G_d (i.e., a *preimage* of G_d with respect to the reachability relation R_k). As discussed earlier, we approximate R_k with the k-forward model $\mathcal{F}_k$. While there exist several algorithms that compute (an approximation of) the set of reachable states from a set of states (i.e., the image of the function $\mathcal{F}_k$, $\{s' \mid \exists s, G_d.\ s' = \mathcal{F}_k(s, G_d)\}$), there are no algorithms that directly compute a preimage. So, *SplitSet* reduces the preimage computation to several image computations (*FnnReach* in the algorithm, which we implement with the tool Ai2 [16]).

SplitSet computes the preimage of G_d iteratively. First, the algorithm (line 4) computes the set of reachable states $G_r =$ *FnnReach*$(\mathcal{F}_k, G_s)$. G_r can partially overlap the destination set G_d (Figure 1a). Since the computation of *FnnReach* introduces over-approximations, the algorithm uses an approximation error r for determining if G_r is completely contained or outside G_d. r is the ratio of the volume of $G_r \cap G_d$, and the volume of G_r (*Volume* is the volume of a set). The value of the error r is between 0 and 1: it is 0 when $G_r \subseteq \overline{G_d}$, and is 1 when $G_r \subseteq G_d$. There are three cases: (i) the algorithm splits G when G *mostly intersects* both G_d and $\overline{G_d}$ (if $t_{\text{unre}} \leq r \leq t_{\text{re}}$, Figure 1a). In our implementation, the function *Split* halves the interval G along one of the existing dimensions, obtaining two sets G', G'' that the algorithm will split recursively. (ii) The algorithm adds G to the set $\mathcal{G}_R$ when G is *mostly contained* in G_d ($r > t_{\text{re}}$, Figure 1b). (iii) The algorithm adds G to the set $\mathcal{G}_U$ when G is *mostly outside* G_d (if $r < t_{\text{unre}}$, Figure 1c).

Thus, *SplitSet* computes the subsets of G_s that mostly reach G_d ($\mathcal{G}_R$) and that mostly not reach G_d ($\mathcal{G}_U$). The thresholds t_{re} and t_{unre} ($0 \leq t_{\text{unre}} < t_{\text{re}} \leq 1$) influence the precision of the analysis and the number of final sets: increasing t_{re} will result in more precise, but more numerous, reachable sets.



(a) $t_{\text{unre}} \leq r \leq t_{\text{re}}$ (b) $t_{\text{re}} < r$ (c) $r < t_{\text{unre}}$

Fig. 1: *SplitSet* splits set G in 1a, concludes that G reaches G_d in 1b, or that G does not reach G_d in 1c.

IV. Evaluation

A. Experimental setup

We evaluate GARA with the following research questions: **RQ1)** Can GARA learn an interpretable goal space representation that decomposes the task structure? **RQ2)** Is the representation effective to achieve better data-efficiency? **RQ3)** Is the representation transferable to new environments?

a) Maze Environments: we evaluate the research questions solving navigation tasks in a maze. In our experiments, we use a U-shaped maze (Figure 2) and a 4-rooms maze (Figure 4). The environment's states $[x, y, v_x, v_y]$ are continuous: (x, y) are the agent's position and v_x, v_y are the agent's velocities on the x-axis and y-axis respectively. The agent has 4 actions to control its acceleration: UP/DOWN/LEFT/RIGHT. The agent receives a positive reward only when it reaches the exit position in the maze (i.e., the reward is **sparse**).

Such learning tasks are ideal to evaluate if our algorithm discovers an interpretable representation that decomposes the task (i.e., **RQ1**). In fact, we can intuitively decompose it into subtasks that correspond to moving from one room to the other, and we can compare such representation with the one GARA learns automatically. Moreover, in our settings the reward is sparse and not continuous (i.e., the euclidean distance from the exit) as in other settings (e.g., [7]). Such sparse reward setting is challenging for existing Feudal HRL approaches that, without a proper goal representation, struggle on both levels of the hierarchy. Observe that, despite having a modest continuous state space (i.e., 4 dimensions), the above learning tasks are *non-trivial*: the reward signal is sparse and reaching the exit requires multiple manoeuvres around obstacles. We leave the evaluation of GARA on higher-dimensional environment as future work.

b) Baselines: we compare GARA with the following combinations of HRL algorithms and representations:

- *Feudal HRL with* **Handcrafted** *representation:* Similar to [6], we manually provide a fixed goal representation (Fig. 2a, 4a) that intuitively decomposes the tasks and that **remains unchanged** during training.
- *Feudal HRL with* **Concrete** *representation:* a deep-learning version of a Feudal HRL algorithm [3] where the high-level agent directly selects **raw states** as goals.
- *Feudal HRL with* **LSTM***:* variation of **Concrete** where the high-level agent uses features of a LSTM as input to capture the time dependencies between goals in the task. The LSTM takes as input environment's states and the previous action.
- **HIRO***:* this is our implementation of the state-of-art Feudal HRL algorithm HIRO [5], which uses raw states as goals but uses a correction mechanism to address non-stationarity.

The high-level agents in GARA and the Handcrafted approach use Q-learning (both the goals and action space are discrete and finite). We use DDPG [17] to learn high-level policies in Concrete, LSTM, and HIRO, since they directly sample goals from the state space. To ensure a fair comparison, all the approaches use DQN [11] with the same network architecture for learning the low-level policy.

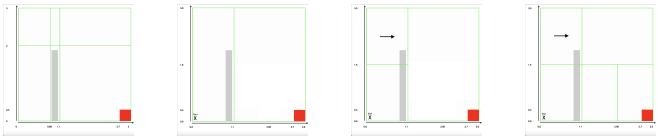
c) Experiments: We run GARA and the above algorithms on both the U-shaped and 4-rooms maze. All the results are obtained over 20 runs of each algorithm (we report the average success rate with its standard deviation). Further details regarding hyperparameters and additional results can be found in the paper's extended version [12] and the code is available at http://www.lix.polytechnique.fr/Labo/Mehdi.ZADEM/GARA/code.tar.gz

B. Results

RQ1 - *representation learning:* Focusing first on the learned representation by GARA, Fig. 2b, Fig. 2c, and Fig. 2d show the evolution of the goal space throughout the learning at 0, 10^3, and 3×10^4 steps (for a randomly selected run of the algorithm). Starting from an initial partition $\mathcal{G} = \{\mathcal{S}\}$, the clustering mechanism in GARA identifies the left half of the maze as the set of states most explored (Fig. 2b). GARA then splits the left half of the maze across the y axis and the v_x axis (Fig. 2c) with the reachability analysis mechanism, discovering the region at the top-left corner of the maze with positive velocity v_x (indicated with $\rightarrow$). Intuitively, such region provides a good starting point to learn policies that efficiently manage to reach the right half of the maze. In the final partition (Fig. 2d), GARA refines the right half region, which allows the algorithm to focus on reaching the region containing the exit point. Our intuition is that such final partition results in easier to reach goals, prompting the agent to select successful behaviours (i.e., the high-level agent samples goals that are reachable from the current state and that can, in turn, lead to a goal containing the exit). In the final representation, only the top-left region is split across a velocity variable. This is justified by the complexity of the maneuvers where a smaller set of states (where $v_x > 0$) reach the right side. Overall, Fig. 2 shows that GARA learns an interpretable representation from data collected during the HRL exploration.

RQ2 - *data efficiency:* Fig. 3 shows that GARA learns a successful hierarchical policy with a performance approaching the handcrafted representation, whereas all the other approaches, including HIRO, cannot learn to solve the task within the same time frame. We attribute this performance to the better sample efficiency due to the learned abstraction, as the agents successfully decompose the task into simple-to-achieve goals. We note that the handcrafted approach starts

improving after GARA since initially it must explore more regions.



(a) Handcrafted representation

(b) $\mathcal{G}$ by GARA via clustering after 10^3 steps

(c) $\mathcal{G}$ learned by GARA after 10^4 steps

(d) Final $\mathcal{G}$ learned by GARA

Fig. 2: Representation of the goal space $\mathcal{G}$ in the U-shaped maze for one run of algorithm. The exit is marked in red. Green boxes show intervals for x, y and the horizontal and vertical arrows indicate the sign of the velocities v_x and v_y, respectively. No arrows indicate there are no split across v_x or v_y.

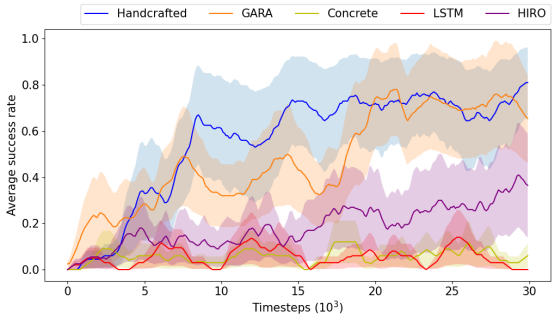
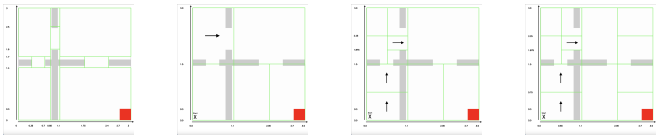


Fig. 3: Average success rate on the U-shaped Maze (over 20 runs).

RQ3 - *transfer learning:* We examine the transferability of the goal space representation when we change the environment's configuration from the U-shaped to the more challenging 4-rooms one. For GARA, we transfer the representation learned from the first task (Fig. 2d), while for Feudal HRL with Concrete representation, LSTM, and HIRO we transfer the weights of the high-level agent's neural networks. We also provide a new representation to the handcrafted approach (Fig. 4a).

Fig. 4c shows that GARA takes advantage of the transferred representation (Fig. 4b), further refining the left side of the maze to better respect its geometry and the smaller openings. Later in the training (Fig. 4d), the right side of the maze is refined to adapt to the new walls, consequently devising different policies in each of these sub-regions. This allows the agent to clearly formulate a goal identifying the openings to go past the obstacles and reach the exit.

We report the performance of the approaches learning from scratch and with transfer learning in Fig. 5. To keep the plot readable, we omit the concrete representation and LSTM approaches that always fail to learn a successful policy. From Fig. 5, we first observe that the performance of HIRO, which uses raw states as goals, do not improve with the transfer. Instead, we observe that GARA with transfer learning performs better than GARA from-scratch, reaching a reward close to the manual representation. Thus, GARA successfully leverages the learned representation and adapts it to solve a more complex task.



(a) Handcrafted representation

(b) Transferred $\mathcal{G}$ for GARA-T

(c) $\mathcal{G}$ learned by GARA-T after 10^4 steps

(d) Final $\mathcal{G}$ learned by GARA-T

Fig. 4: Goal space representations for the 4-Rooms Maze

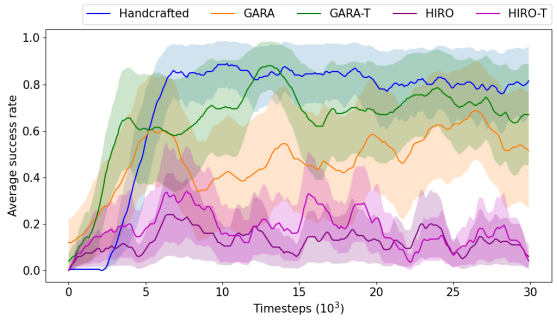


Fig. 5: Average success rate on the 4-Rooms Maze (over 20 runs). -T refers to the transferred versions of the algorithms.

V. Related Works

Learning and acting over different space and time resolutions simultaneously is a key challenge in tasks involving long-range planning. Sutton et al. [18] proposed the options framework for a temporal abstraction. Recent approaches focus on state representation. Vezhnevets et al. [4] proposes a feudal HRL [3] that learns a low-dimensional latent state space for subgoal selection, which still fails to capture critical information (e.g. temporal dependencies) to solving the task as argued in [19]. [7], [20], [21], [22], [23] propose subgoal sampling methods that are aware of the environment dynamics. Closer to our idea of analysing reachability, Zhang et al. [7] propose to learn a distance-based adjacency relation between subgoals in a predefined space and use it to sample reachable subgoals. Ghosh et al. [20] propose to learn state embeddings using the action distribution of a goal-conditioned policy but this requires already successful policies. Savinov et al. [21] propose a supervised learning approach for estimating reachabilty relations between states. Li et al. [23] propose to learn a stable state embedding along with an exploration policy that aims for novel reachable subgoals. While these representations remain in the continuous domain, GARA's abstraction constitutes a symbolic representation of a multi-dimensional continuous state space. It thus bridges the continuous world with a symbolic description, opening the door to methods such as planning, symbolic reasoning or interpretability. Indeed, Kulkarni et al. [6] also resort to a discrete, manual, object-based subgoal representations where goals are modeled as sets of states. In comparison, our approach could find an emerging abstraction by learning online. Alternatively, Nachum et al. [5] proposed an approach that relies on sampling subgoals directly from the state space. However, this unconstrained selection can result in learning inefficiency and suboptimal policies.

Symbolic methods in RL were studied in works like [24]. Lyu et al. [25] incorporate symbolic reasoning to function

as planning for RL agents. However, this requires manual symbolic domains to abstract the states, as well as complete descriptions for the learning objective. Other approaches alleviate this by learning the planning instruction on a set of given abstract states [26], [27], [28]. Eventually, the learned high-level behaviour of the agent produces faster learning and can be expressed as a logical formula that allows interpretability and transfer learning. Our work achieves these results without relying on given abstractions.

A few approaches have tacked emergence of representations for hierarchical tasks in open-ended learning. IM-PB [29] learns a recursive task decomposition for hierarchical tasks of various complexities with intrinsic motivation. CURIOUS [30] selects goals from a set of pre-defined modules. [31] uses language instructions as an abstract goal space, and can imagine new goals by language compositionality. In [32], nested modules of different complexity levels emerge from predefined low-level features and a list of predefined affordances. GARA allows for more flexible goal definitions without specific instructions or behaviour descriptions.

VI. Conclusion

In this paper, we tackled the problem of learning from self-exploration data, a symbolic goal representation of a continuous state space that efficiently captures the task structure, is transferable and interpretable. We first proposed a representation abstracting into the same goal all the states that reach the same goal with the same low-level policy. This representation as a set of states is interpretable and captures the relation between high- and low-level policies. We then introduced GARA, a HRL algorithm that discovers such representation online, while also learning the agents' policies. GARA's representation emerges dynamically from the data acquired during exploration. Technically, GARA refines the representation from the reachability relations approximated with a neural network and studied with set-based reachability analysis. Experimental results show that GARA gradually learns a data-efficient goal representation that is *necessary* to solve tasks on complex continuous environments with sparse rewards. Moreover, the experiments show that GARA's representation is interpretable and transferable.

In the future, we plan to learn a representation in higher-dimensional environments, where learning low-level policies is harder and an interval set representation is not sufficient.

References

[1] T. Taniguchi, E. Ugur, M. Hoffmann, L. Jamone, T. Nagai, B. Rosman, T. Matsuka, N. Iwahashi, E. Oztop, J. Piater, and F. Wörgötter, "Symbol emergence in cognitive developmental systems: A survey," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 4, pp. 494–516, 2019.

[2] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 1, 2003.

[3] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *NeurIPS*, vol. 5, 1992.

[4] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," *CoRR*, vol. abs/1703.01161, 2017.

[5] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *NeurIPS 2018*, 2018.

[6] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *NeurIPS*, vol. 29, 2016.

[7] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, "Generating adjacency-constrained subgoals in hierarchical reinforcement learning," in *NeurIPS*, 2020.

[8] C. Liu, T. Arnon, C. Lazarus, C. A. Strong, C. W. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Found. Trends Optim.*, vol. 4, no. 3-4, 2021.

[9] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *ICML*, vol. 37, 2015.

[10] C. Baier and J. Katoen, *Principles of model checking*, 2008.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[12] M. Zadem, S. Mover, and S. M. Nguyen, "Goal space abstraction in hierarchical reinforcement learning via set-based reachability analysis," Tech. Rep., 2023. [Online]. Available: http://www.lix.polytechnique.fr/Labo/Mehdi.ZADEM/GARA/icdl23.pdf

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, 1996.

[14] X. Rival and K. Yi, *Introduction to Static Analysis: An Abstract Interpretation Perspective*, 2020.

[15] R. Paige and R. E. Tarjan, "Three partition refinement algorithms," *SIAM J. Comput.*, vol. 16, no. 6, 1987.

[16] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *IEEE Symposium on Security and Privacy*, 2018.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2016.

[18] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, 1999.

[19] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-optimal representation learning for hierarchical reinforcement learning," in *ICLR*, 2019.

[20] D. Ghosh, A. Gupta, and S. Levine, "Learning actionable representations with goal conditioned policies," in *ICLR 2019*, 2019.

[21] N. Savinov, A. Raichuk, D. Vincent, R. Marinier, M. Pollefeys, T. P. Lillicrap, and S. Gelly, "Episodic curiosity through reachability," in *ICLR*, 2019.

[22] B. Eysenbach, R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," in *NeurIPS 2019*, 2019.

[23] S. Li, L. Zheng, J. Wang, and C. Zhang, "Learning subgoal representations with slow dynamics," in *ICLR*, 2021.

[24] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards deep symbolic reinforcement learning," *CoRR*, vol. abs/1609.05518, 2016.

[25] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," in *AAAI*, 2019.

[26] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith, "Symbolic plans as high-level instructions for reinforcement learning," in *AAAI*, 2020.

[27] Z. Ma, Y. Zhuang, P. Weng, H. H. Zhuo, D. Li, W. Liu, and J. Hao, "Learning symbolic rules for interpretable deep reinforcement learning," *CoRR*, vol. abs/2103.08228, 2021.

[28] M. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening, "DeepSynth: Automata synthesis for automatic task segmentation in deep reinforcement learning," in *AAAI 2021*, 2021.

[29] N. Duminy, S. M. Nguyen, and D. Duhaut, "Learning a set of interrelated tasks by using sequences of motor policies for a strategic intrinsically motivated learner," in *IRC*, 2018.

[30] C. Colas, P. Oudeyer, O. Sigaud, P. Fournier, and M. Chetouani, "CURIOUS: intrinsically motivated modular multi-goal reinforcement learning," in *ICML*, 2019.

[31] C. Colas, T. Karch, N. Lair, J. Dussoux, C. Moulin-Frier, P. F. Dominey, and P. Oudeyer, "Language as a cognitive tool to imagine goals in curiosity driven exploration," in *NeurIPS 2020*, 2020.

[32] E. Ugur and J. Piater, "Emergent structuring of interdependent affordance learning tasks using intrinsic motivation and empirical feature selection," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 9, no. 4, 2017.

APPENDIX

THE K-FORWARD MODEL

The $k-$forward model $\mathcal{F}_k : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{S}$ is trained to predict the state s' reached after applying $\pi_{\text{Low}}(s, G')$ for k steps. In practice, we use a fully connected neural network to approximate this relation. This network is composed of two layers with 16 neurons each. We use an MSE loss function to train the network with a learning rate of 0.1.

The forward model $F_k(s, G)$ approximates the state reached when applying a policy $\pi_{\text{Low}}(s, G)$ for k steps. Since this policy is changing during training. The model F_k is regularly updated. This can be considered as a similar process to off-policy DRL approaches where the policy networks keep track of the trained policy's generated rewards. Importantly, the reachability of a policy is only analysed after the policy has stabilised. We rely on F_k's errors to determine this criteria. More precisely, we assume that if the corresponding forward model's error remains stable across multiple episodes, and since it is trained on data generated by π_{Low}, then the low-level policy is stable.

DISCUSSING NON-STATIONARITY IN GARA

Non-stationarity issues, common in HRL algorithms, refer to a high-agent's experience replay containing data that stems from changing behaviours. As the low-level policies are trained, their outcome changes leading to replay data not reflecting the behaviour of the current agents. Non-stationarity is avoided in GARA due to the advantages offered by its structure. First, GARA's high-level agent relies on Q-learning for training π_{High} and does not need to rely on a replay buffer. This naturally lessens the dependence on old experiences and introduces less errors in training. Second, the refinement mechanism is activated once low-level policies have stabilised meaning that non-stationarity will not affect the following high-level training.

THE MAZE ENVIRONMENT

In the maze environment, the actions affect the acceleration of the agent and consequently its velocities and positions. Each action either increases or decreases the acceleration value by an increment of 0.1 at a timestep. UP/DOWN affect the vertical acceleration $a_y(t)$ while the RIGHT and LEFT action change the horizontal acceleration $a_x(t)$. When picking an action across an a certain axis, the acceleration value of the unpicked axis moves closer to 0. Concretely, these actions affect the corresponding velocities as $v_x(t) = max(min(v_x(t-1) + a_x(t), 1), -1)$ and $v_y(t+1) = max(min(v_y(t-1) + a_y(t), 1), -1)$.

HYPERPARAMETERS

We tuned the DQN networks by testing the low-level policy convergence in small, unobstructed rooms. This allows us to simulate the required behaviour from low-level policies. The Q-network for all approaches is thus composed of 2 hidden layers (with 16 and 32 neurons respectively). The learning rate is set to 0.01. The low-level agent uses an ϵ-greedy policy with ϵ annealed exponentially by a factor of 0.9995 from $\epsilon_{initial} = 1$ to $\epsilon_{min} = 0.01$. The target networks are updated every 20 time steps. For Feudal HRL with concrete representations, both Actor and Critic networks of DDPG have 2 hidden layers with 16 and 32 neurons respectively. Feudal HRL with LSTM adds to that an LSTM layer with 32 units.

For every approach, a new goal is sampled by the high-level each $k = 5$ steps. k is chosen intuitively as being the mean number of steps an agent could take to exit a room in our environment. The maximal number of steps per episode is set to 200.

ADDITIONAL RESULTS

We conducted additional experiments on other maze configurations to assess our results under other types of environment variations. First, we switch the start and exit positions in the U-shaped Maze. This is a form of task modification where the environment is unchanged. Fig.6 shows that the transferred version of GARA starts learning faster than the from-scratch version. They achieve comparable behaviours in the later stages of learning. This is explained by the fact that GARA -Transfer can train its low-level policies faster with the provided precise representation whereas the from-scratch version has to first identify the subgoal representation before it can train efficient low-level policies. It is to note that GARA -Transfer in this task does not refine the goal space as the transferred partition already satisfies the reachability property.

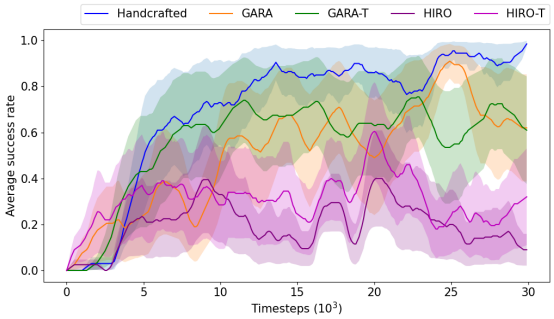


Fig. 6: Evaluation on U-shaped Maze with switched Start/Exit

Second, we expand the size of the environment and propose an N-shaped maze Fig.7. This is a configuration designed to assess transfer performance in a larger environment. Fig.8

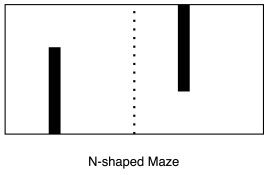


Fig. 7: N-shaped Maze

shows that the transferred version of GARA benefits greatly from the transferred representation. It manages to learn faster and better performing policies even compared to the Handcrafted approach. This is due to the handcrafted approach

that, while intuitive, does not sufficiently decompose the task and adapt it to the capabilities of the low-level agent Fig.9a. This is especially more prominent in the middle section of the maze. On the other hand, GARA is able to address the ambiguity of the large middle section as seen in Fig.9b, Fig.9c and Fig.9d.

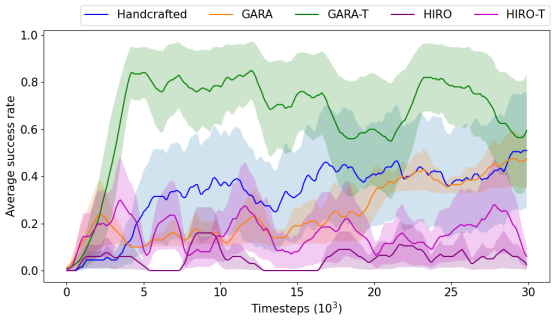
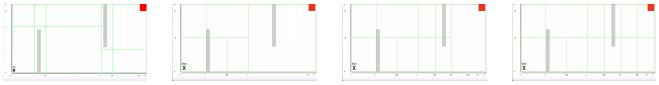


Fig. 8: Evaluation on N-shaped Maze



(a) Handcrafted representation

(b) Transferred $\mathcal{G}$ for GARA-Transfer

(c) $\mathcal{G}$ learned by GARA -Transfer after 10^3 steps

(d) Final $\mathcal{G}$ learned by GARA -Transfer

Fig. 9: Representation of the subgoal space in the N-Maze configuration