Systematic Generalisation of Temporal Tasks through Deep Reinforcement Learning

Borja G. León
Imperial College London
London, United Kingdom
b.gonzalez-leon19@imperial.ac.uk

Murray Shanahan
Imperial College London
London, United Kingdom
m.shanahan@imperial.ac.uk

Francesco Belardinelli
Imperial College London
London, United Kingdom
francesco.belardinelli@imperial.ac.uk

ABSTRACT

This work addresses the problem of grounded agents learning systematically from formally specified instructions and generalising their learning to execute zero-shot i.e., never-seen-before, specifications. To the best of our knowledge, we are the first to provide evidence that systematic learning can emerge with abstract operators (such as negation) when learning from a limited variety of training examples, which previous research has struggled with. In particular, we find that within a neural network, the particular architecture of convolutional layers are key when grounded agents generalise to new instructions. We also present a neuro-symbolic framework where a symbolic module transforms temporal specifications into a form that helps the training of a deep RL agent targeting generalisation, while a neural module learns systematically to solve the given tasks. Through this framework, we achieve agents that execute multi-task, zero-shot instructions.

KEYWORDS

Systematic Generalisation, Convolutional Neural Networks, Reinforcement Learning, Temporal Logic

1 INTRODUCTION

Systematic generalisation (also called combinatorial generalisation [29]) concerns the human capacity for compositional learning, that is, the algebraic capacity to understand and execute novel utterances by combining already known primitives [9, 15]. For instance, once a human agent understands the instruction "get *wood*" and the meaning of *iron*, they will understand the task "get *iron*". This is a desirable feature for a computational model, as it suggests that once the model is able to understand the components of a task, it should be able to satisfy tasks with the same components in possibly different combinations.

The ability of neural-based agents to learn systematically and generalise beyond the training environment is a recurrent point of debate in machine learning (ML). In the context of autonomous agents solving human-given instructions, deep reinforcement learning (DRL) holds considerable promise [10, 33]. Previous studies have shown that DRL agents can exhibit some degree of systematic generalisation of natural language [4, 22, 45]. In parallel, frameworks based on structured or formal languages to instruct DRL agents are also gaining momentum [11, 44, 46]. Formal languages offer several desirable properties for RL including unambiguous semantics, and compact compositional syntax, which is particularly relevant when targeting safety-aware applications [1, 25, 39]. However, works with formal languages have traditionally focused on solving training tasks [19, 20, 23] or rely on new policy networks when tackling novel instructions [27], with its associated computational burden.

In order to progress towards agents that execute zero-shot, i.e., unseen, formal instructions it is desirable to design frameworks that facilitate the emergence of systematic learning in DRL. There exists substantial evidence that DRL agents can learn systematically only when the right drivers are present [29, 34]. Unfortunately, the focus on learning from natural instructions has hindered the study of drivers for systematicity with logic operators, e.g., negation or disjunction. Only Hill et al. [21] provides evidence of DRL agents with some degree of systematicity facing negated zero-shot instructions. This work suggests that systematic learning can emerge from as few as six instructions when handling positive tasks, but that a much larger number of instructions (~ 100) is a major requirement when aiming abstract operators such as negation. This constraint would be a strong burden for DRL agents, meaning they could only generalise abstract concepts when rich and diverse environments are available.

Contributions. In this work we aim to answer the question: *can systematic learning emerge in DRL agents with abstract logic operators, e.g., disjunction, when learning from a limited variety (~ 6) of examples?* Specifically, we make the following contributions:

- We provide empirical evidence of emergent systematic generalisation in DRL agents with abstract operators – including negation, which previous works have struggled with – when learning from as few as six training instructions. Note that the previous latest research suggested that significantly larger numbers (~ 100) were strongly required [21].
- We find that the architecture of the convolutional layers (c.l.) – a background feature in previous literature – can be key when generalising to zero-shot tasks.
- We also introduce a novel neuro-symbolic framework aimed at agents that exhibit systematic generalisation. In this framework, the formal language helps neural-based agents that target zero-shot instructions, while the symbolic module facilitates generalisation to longer-than-training instructions.

The remaining of this document is structured as follows: Section 2 introduces the key concepts needed to understand our research. Section 3 presents the formal language we use to provide instructions, as well as our neuro-symbolic framework. Section 4 details the experimental settings and results obtained. Finally, Sections 5 and 6 report on related works and discussion, respectively.

2 SYSTEMATIC GENERALISATION OF FORMAL TASKS

Our situated agents operate with a fixed visual perspective and limited visual range. Thus, we work with partially observable (p.o.)

Proc. of the Adaptive and Learning Agents Workshop (ALA 2022), Cruz, Hayes, Silva, Santos (eds.), May 9-10, 2022, Online, https://ala2022.github.io/. 2022.

environments. Our target is to assess whether systematic learning is emerging in such environments while stressing that the convolutional neural network (CNN) architecture is key to facilitate generalisation when the number of training examples is limited. This section briefly introduces the concepts needed to present our work.

Reinforcement Learning. In RL p.o. environments are typically modelled as a POMDP.

DEFINITION 1 (POMDP). *A* p.o. Markovian Reward Decision Process *is a tuple* $\mathcal{M} = \langle S, A, P, R, Z, O, \gamma \rangle$ *where (i)* S *is the set of states* $\{s, s', \ldots\}$. *(ii)* A *is the set of* actions $\{a, a', \ldots\}$. *(iii)* $P\,(s'|s, a) : S \times A \times S \to [0, 1]$ *is the (probabilistic)* transition function. *(iv)* $R(s, a, s) : S \times A \times S \to \mathbb{R}$ *is the* Markovian reward function. *(v)* Z *is the set of* observations $\{z, z', \ldots\}$. *(vi)* $O(s, a, s) : S \times A \times S \to Z$ *is the* observation function. *(vii)* $\gamma \in [0, 1)$ *is the* discount factor.

At every timestep t, the agent chooses an action $a_t \in A$ updating the current state from s_t to $s_{t+1} \in S$ according to P. Then, $R_t = R(s_t, a, s_{t+1})$ provides the reward associated with the transition, and $O(s_t, a_t, s_{t+1})$ generates an observation o_{t+1} that will be provided to the agent. Intuitively, the goal of the learning agent is to choose the policy π that maximizes the expected discounted reward from any state s. Given the action-value function $Q^\pi(s, a) = \mathbb{E}\,[R_t|s_t = s, a]$, RL searches for an optimal policy whose actions are meant to maximise the action-value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$. For more detail we refer to Sutton and Barto [42].

Convolutional neural network (CNN).. A CNN is a specialized type of neural network model [30] designed for working with visual data. Particularly, a CNN is a network with one or more convolutional layers (c.l.). Relevant features of c.l. for this study include *kernels*, a matrix of weights which is slid across an input image and multiplied so that the output is enhanced in a desirable manner, and *stride*, which measures the step-size of the kernel slid. After the last convolutional layer, the output is a tensor that we refer to as *visual embedding*. We point to Goodfellow et al. [18] for a complete description of CNNs.

Evaluating the emergence of systematic generalisation. As anticipated in Sec. 1, we aim to assess whether a neuro-symbolic DRL agent is able to generalize systematically from temporal-logic (TL) instructions. Consequently, *we do not focus on the raw reward collected by the agent*, but rather on whether the agent is correctly following new instructions. To evaluate this, we work in two popular settings involving procedural maps where agents need to fulfill reachability goals, in line with previous literature [21, 24, 27]. In our experiments agents are trained with instructions such as "get gold" (reach gold) or "get something different from mud" (i.e., reach an object that is not mud). Then, we evaluate the agents with unseen instructions, e.g., "get something different from iron, then get iron". It is evident that a learning agent achieving significantly better rewards than a random walker does not imply systematic learning, e.g., the agent could accumulate larger rewards just by navigating better than a random walker and reaching all the objects promptly. For this reason, after training, we evaluate generalisation in settings called binary choice maps (BCMs). In these test maps, agents are evaluated with zero-shot reliable instructions, i.e., unseen *reliable* instructions that point to objects granting a positive reward, or zero-shot *deceptive* instructions, i.e., instructions pointing to objects granting a penalisation – a negative reward –. Since during training we always provide reliable instructions, agents that show systematic learning – not complete systematicity but at least similar to closely related works [4, 21] – should accumulate significantly higher rewards with zero-shot reliable instructions than with deceptive ones (we estimate around two times higher or greater to be a good reference in our settings, detailed in Sec. 4).

3 SOLVING ZERO-SHOT INSTRUCTIONS IN TTL

In this section we present a framework to tackle complex (multi-task) zero-shot temporal specifications that leverages a DRL agent learning systematically to solve training tasks. Our framework relies on 3 components detailed below: 1) a formal language, whose atoms are designed to facilitate the systematic learning of a neural-based module; 2) a symbolic module, whose role is to decompose specifications into sequences of simpler tasks, and provide the reward signal for the neural module on the current task to execute; 3) a neural module consisting of a DRL agent aimed to learn systematically from the tasks fed by the symbolic module.

3.1 Task temporal logic

We start by defining a novel TL language whose atomic components are used to train a DRL algorithm. Task temporal logic (TTL) is an expressive, learning-oriented TL language interpreted over finite traces, i.e, over finite episodes. Given a set AT of atomic tasks $\alpha, \beta, \ldots$, the syntax of TTL is defined as follows:

DEFINITION 2 (TTL). *Every formula* ϕ *in TTL is built from atomic tasks* $\alpha \in AT$ *by using negation "*$\sim$*" (on atoms only), sequential composition "*;*" and non-deterministic choice of atoms "*$\vee$*" and formulae "*$\cup$*":*

$$\begin{aligned} T &::= \alpha \mid \alpha\sim \mid \alpha \vee \alpha \\ \phi &::= T \mid \phi;\phi \mid \phi \cup \phi \end{aligned}$$

Intuitively, an atomic task α corresponds to a reachability goal in the TL literature, in the sense that the fulfilment condition associated with α will eventually hold. Task $\alpha\sim$ encapsulates a special form of negation; informally it means that something different from α will eventually hold. Choosing this form of negation allows us to contrast our results with those of previous studies of this operator in natural language [21]. Note that the negation symbol is intentionally positioned after the atomic task. We found that this feature helps during training with visual instructions, as it forces the neural module (Sec. 3.3) to process the same number of symbols to distinguish negative from positive tasks (we refer to Sec. E.1 in the supplemental material for further detail). Formulae $\phi;\phi'$ intuitively express that ϕ' follows ϕ in sequential order; whereas $\phi \cup \phi'$ means that either ϕ or ϕ' holds. We then also introduce an abbreviation for the concurrent (non-sequential) composition $\cap$ as follows: $\phi \cap \phi' \equiv (\phi;\phi') \cup (\phi';\phi)$. We say that our language TTL is *learning-oriented* as its logical operators and their positions in formulae are so chosen as to help the training process of the learning agent.

TTL is interpreted over finite traces λ, where $|\lambda|$ denotes the length of the trace. We denote time steps, i.e., instants, on the trace

as $\lambda[j]$, for $0 \leq j < |\lambda|$; whereas $\lambda[i, j]$ is the (sub)trace between instants i and j. In order to define the satisfaction relation for TTL, we associate with every atomic task α an atomic proposition p_α, which represents α's fulfilment condition. Note again that, in this context, α is a *reachability goal*, typically expressed in TL as an *eventuality* $\diamond p_\alpha$. Then, a *model* is a tuple $\mathcal{N} = \langle \mathcal{M}, L \rangle$, where $\mathcal{M}$ is a POMDP, and $L : S \to 2^{AP}$ is a labelling of states in S with truth evaluations of atoms p_α in some set AP of atoms.

DEFINITION 3 (SATISFACTION). *Let $\mathcal{N}$ be a model and λ a finite trace. We define the satisfaction relation $\models$ for tasks T and formulae ϕ on trace λ inductively as follows:*

$(\mathcal{N}, \lambda) \models \alpha$ *iff* *for some* $0 \leq j < |\lambda|$, $p_\alpha \in L(\lambda[j])$

$(\mathcal{N}, \lambda) \models \alpha \sim$ *iff* *for some* $0 \leq j < |\lambda|$, *for some* $q \neq p_\alpha$, $q \in L(\lambda[j])$ *and* $p_\alpha \notin L(\lambda[j])$

$(\mathcal{N}, \lambda) \models \alpha \vee \alpha'$ *iff* $(\mathcal{N}, \lambda) \models \alpha$ *or* $(\mathcal{N}, \lambda) \models \alpha'$

$(\mathcal{N}, \lambda) \models \phi; \phi'$ *iff* *for some* $0 \leq j < |\lambda|$, $(\mathcal{N}, \lambda[0, j]) \models \phi$ *and* $(\mathcal{N}, \lambda[j+1, |\lambda - 1|]) \models \phi'$

$(\mathcal{N}, \lambda) \models \phi \cup \phi'$ *iff* $(\mathcal{N}, \lambda) \models \phi$ *or* $(\mathcal{N}, \lambda) \models \phi'$

By Def. 3.2, an atomic task α indeed corresponds to a formula $\diamond p_\alpha$ of temporal logic, where p_α is the fulfilment condition associated with α. It can be immediately inferred that TTL is a fragment of the widely-used Linear-time Temporal Logic over finite traces (LTL_f) [12]. We provide a translation of TTL into LTL_f and the corresponding proof of truth preservation in Sec. F in the appendix. To better understand the language we present a toy example.

EXAMPLE 1. *The specification "get something different from grass, then collect grass and later use either the workbench or the toolshed" can be expressed in TTL as:*

$$\phi \triangleq (grass \sim); grass; (workbench \cup toolshed)$$

TTL is designed to be expressive enough to encapsulate both the tasks described in [2], a popular benchmark in the RL-TL literature [11, 31, 43], and the negated instructions from studies about negation in natural language [21], i.e., encapsulating "not wood" as "get something different from wood".

REMARK. *As a foundational work assessing the ability of learning agents to generalise abstract operators with unseen instructions, we intentionally restricted TTL syntax with atomic operators to negation and disjunction. This allows to carefully study the ability of DRL agents to generalise with these two important operators.*

3.2 The Symbolic module

Given an instruction ϕ in TTL, the first task of the symbolic module (SM), detailed in Algorithm 3.2, is to decompose ϕ into sequences of atomic tasks for the neural module (NM). Intuitively, the SM reduces the problem into a progression of POMDPs, which is a well-known procedure in the literature on RL with TL specifications [6, 43]. In particular, a task extractor $\mathcal{E}$ transforms ϕ into the set $\mathcal{K}$ of all sequences of tasks T that satisfy ϕ. As standard in the literature [2, 27], we assume that the SM has access to an internal labelling function (a.k.a. event detector[1]) $\mathcal{L}_I : Z \to 2^{AP}$, which maps the agent's observations into sets of atoms in AP. Note that $\mathcal{L}_I$ might differ from the labelling $\mathcal{L}$ originally defined in Sec. 3.1, since $\mathcal{L}$ is defined on states. For our approach to be effective, observations have to contain enough information to decide the truth of atoms, so that both the agent's internal and the model's overall labelling functions remain aligned. Given $\mathcal{L}_I$ and the sequences of tasks in $\mathcal{K}$, a progression function $\mathcal{P}$ selects the next task from $\mathcal{K}$ that the NM has to solve. Once a task is solved, $\mathcal{P}$ updates $\mathcal{K}$ and selects the next task. This process is repeated until ϕ is satisfied or the time limit is reached. We include the pseudocode for $\mathcal{P}$ and $\mathcal{K}$ subroutines in Appendix B.

[1] Note that RL literature and vanilla RL settings [42] also rely on these event detectors to provide rewards to the agents although they are not formally defined.

Algorithm 1 The symbolic module (SM)

Input: Instruction ϕ
Generate the accepted sequences of tasks $\mathcal{K} \leftarrow \mathcal{E}(\phi)$
Retrieve the current observation: z
Get the true proposition: $p \leftarrow \mathcal{L}_I(z)$
Get the first task: $T \leftarrow \mathcal{P}(\mathcal{K}, p)$
repeat
  Generate the extended observation: $z^{ext} \leftarrow z + T$
  Get the next action: $a \leftarrow \text{NM}(z^{ext})$
  Execute a, new observation z'
  New true proposition: $p' \leftarrow \mathcal{L}_I(z')$
  Provide the internal reward: $\text{NM} \leftarrow R_I(p')$
  if $p' == p_\alpha$ for $\alpha \in T$ **then**
    Update $\mathcal{K}, T' \leftarrow \mathcal{P}(\mathcal{K}, p')$
  end if
until $\mathcal{K} == \emptyset$ or time limit

When forwarding the current task T to the NM, the SM expands observation z with the subformula corresponding to T. In the current context, z is a square visual input with a fixed perspective. In our empirical setting, this is expanded either by adding extra rows to z containing T (see Figure 1 center) or providing T in a separated "language" channel (see Figure 1 right). During the episode the SM also provides the reward signal that guides the NM to solve T:

$$R(p_t) = \begin{cases} d & \text{if } p_t = \emptyset, \text{ where } d \in \mathbb{R} \text{ and } d < 0; \\ i & \text{if } p_t = p_\alpha, \text{ for } \alpha \text{ occurring in } T \text{ and} \\ & T \neq \alpha \sim, \text{ or if } p_t \neq p_\alpha, \text{ for } \alpha \text{ occurring in} \\ & T \text{ and } T \equiv \alpha \sim, \text{ where } i \in \mathbb{R} \text{ and } i \geq 0; \\ c & \text{otherwise, where } c \in \mathbb{R} \text{ and } c \ll d. \end{cases}$$

where $\mathcal{L}_I(z_t) = p_t$ for any time step t. Intuitively, the NM is rewarded when the event detector $\mathcal{L}_I$ fires a signal that satisfies the current task, and penalised when $\mathcal{L}_I$ signals an event not related to T. A small penalisation is also given at each time step to induce the agent to promptly solve T. We now illustrate the inner working of the SM.

EXAMPLE 2. *Consider one of the complex specifications used in Sec. 4: $\phi_1 = ((wood; grass) \cup (iron; axe)); workbench; (toolshed \sim)$. Given ϕ_1, the task extractor $\mathcal{E}$ outputs set $\mathcal{K} = \{\langle [wood], [grass], [workbench], [toolshed \sim] \rangle, \langle [iron], [axe], [workbench], [toolshed \sim] \rangle\}$ with two available sequences of tasks. Set $\mathcal{K}$ is forwarded to the progression function $\mathcal{P}$ and since we have two lists with different initial elements, both positive, $\mathcal{P}$ selects $[wood \vee iron]$ as task T to extend the next observations of the NM. The NM iterates with the environment using the extended observations as input. At each iteration, functions $\mathcal{L}_I$ and R respectively evaluate the progression*

of T and reward the NM until T is satisfied. In our example, if the NM got wood, the second sequence and $[wood]$ *are discarded and the next task becomes* $[grass]$*. This process is repeated until $\mathcal{K}$ is empty or the time limit is reached.*

3.3 The Neural module

The neural module consists of a DRL algorithm. We use A2C, a popular synchronous version of the algorithm introduced by Mnih et al. [36]. The NM is trained in procedurally generated maps, using a fixed perspective which is known to help agents with generalisation [34]. All the agents differ on the CNN architecture.

CNN architecture. We study architectures with convolutional layers followed by a fully connected (f.c.) layer [18] and a recurrent layer (an LSTM from Gers et al. [17]). In preliminary studies, we observed that agents with different f.c. and recurrent layer configurations can exhibit similar generalisation abilities. However, as detailed in Sec. 4, different convolutional configurations yield significantly different results. Particularly, while all the convolutional architectures we test achieve a similarly good performance with training instructions, only specific configurations correctly execute zero-shot tasks. We find that the ability to generalise is correlated to the degree of *alignment* between the CNN architecture and the environment. In our gridworld settings, we say that a CNN is weakly aligned (WA) with an environment when the kernel dimensions and stride size are factors of the tile resolution. If a CNN is WA and the two first dimensions (length and width) of its visual embedding –the output of the convolutional layers– correspond to the number of input tiles in the original observation, we say that this CNN is strongly aligned (SA). Note that the number of channels of the output, i.e., the third dimension of the embedding, has no influence in the alignment. Last, when a CNN is not WA or SA, we say it is not aligned (NA). When aiming systematic learning from ~ 6 training instructions, we find that only SA networks (when giving visual instructions) or SA and WA networks (when instructions are given in a separated channel) correctly execute zero-shot tasks.

4 EXPERIMENTS

We evaluate our framework in procedurally generated grid-worlds. The first setting (Figure 1 left) is designed along the lines of a Minecraft-inspired (Minecraft for sort) benchmark widely used in RL-TL [2, 11, 43], which we adapt to match the 2D maps from Hill et al. [21] (the last study about negation). The action set consists of 4 individual actions: *Up, Down, Left, Right*, that move the agent one tile in the given direction. In this setting each tile has a resolution of 9x9x1 values. The DRL algorithm receives the observation extended with visual instructions depicting the target objects (Figure 1 center). We also evaluate the agents in the popular MiniGrid benchmark [8], depicted in Figure 1 right. Here each tile has a resolution of 8x8x3 values, and the action set is *Move forward, Turn left, Turn right*. In this benchmark instructions are given through a separated language channel.

For each setting we test four different CNN configurations. The first two are popular architectures from previous literature that we use as baselines: *Atari-CNN* [36, 37] which is NA with Minecraft and WA with MiniGrid, and *ResNet* [14, 21] WA with Minecraft and NA with MiniGrid. We contrast these networks with different architectures designed to be SA with each environment and that we refer to as Env-CNN. Last, for each environment we also test the performance of the same Env-CNNs, but when the c.l. are frozen during training, i.e., the c.l. always use a set of randomly selected parameters. We refer to these networks as Env-CNN$^{\text{RF}}$. Figure 2 includes schemes of the neural network architectures from Section 4. On the left, we see the general setting for Minecraft, where tasks instructions are embedded within the observation, and MiniGrid, where instructions are given in a separated channel. Note that we intentionally use a significantly different number of channels in the last layers with Env-CNN$_{\text{minecraft}}$ and Env-CNN$_{\text{minigrid}}$. We do this to evidence that the number of channels in the visual embedding does not affect the ability of SA architectures to generalise training instructions. In Sec 4 we observed that both SA architectures show good generalisation performance in their respective settings.

In every experimental setting we refer to the global set of objects as $\mathcal{X}$. $\mathcal{X}$ is split into training ($\mathcal{X}_1$), validation ($\mathcal{X}_2$) and test sets ($\mathcal{X}_3$) where $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$, $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$, $\mathcal{X}_1 \cap \mathcal{X}_3 = \emptyset$, and $\mathcal{X}_2 \cap \mathcal{X}_3 = \emptyset$. We contrast the results of learning with training sets of different size that we refer to as *small* ($|\mathcal{X}_1| = 6$) and *large* ($|\mathcal{X}_1| = 20$). In Minecraft, where instructions are visual, agents are trained with objects from $\mathcal{X}_1$ only. In MiniGrid, where agents require to ground words to objects, we train the agents with positive instructions from $\mathcal{X}$, while instructions concerning the negation or disjunction operators refer only to objects in $\mathcal{X}_1$. Agents are trained with atomic tasks T. Tasks are procedurally generated with objects from the corresponding set. For instance, with the small setting we have six different objects, meaning that agents are learning the negation operator from six training instructions. At each episode the agent navigates receiving rewards according to T. The episode ends when the agent receives a positive reward (meaning that T is satisfied) or the time limit is reached. See Appendix B for the reward function. During training we periodically evaluate the performance of the agents with the corresponding validation set $\mathcal{X}_2$, that we use to select the best learning weights in each run, i.e., to perform early stopping when necessary. Further details about object set generation, reward values, hyperparameters, environment mechanics and training and validation plots are included in Appendix A.

4.1 Empirical Results

Results about systematic learning. Table 1 presents the raw reward results in $\mathcal{X}_1$ and $\mathcal{X}_3$ with the different CNNs. Note that "train" refers to results with objects from the training set but with the weights that achieve the best rewards in $\mathcal{X}_2$. Therefore, all the networks can accumulate higher rewards in training if we let them overfit (See Figure 3 in appendix for training and validation plots). Note that results in Table 1 alone do not yield a strong correlation between the alignment of the CNN and the generalisation ability. Yet, as anticipated in Sec. 2 this correlation becomes evident when evaluating whether the agents are actually following the test instructions. Table 2 shows the results in a particular set that we call binary choice maps (BCMs). Here maps have only two objects (except in "disjunction 2 choices", with four), one granting a positive reward ($i = +1$) the other a penalisation ($c = -1$). In "disjunction 2 choices" two objects penalise, and two provide a reward. Since agents are always trained with reliable instructions pointing to objects that

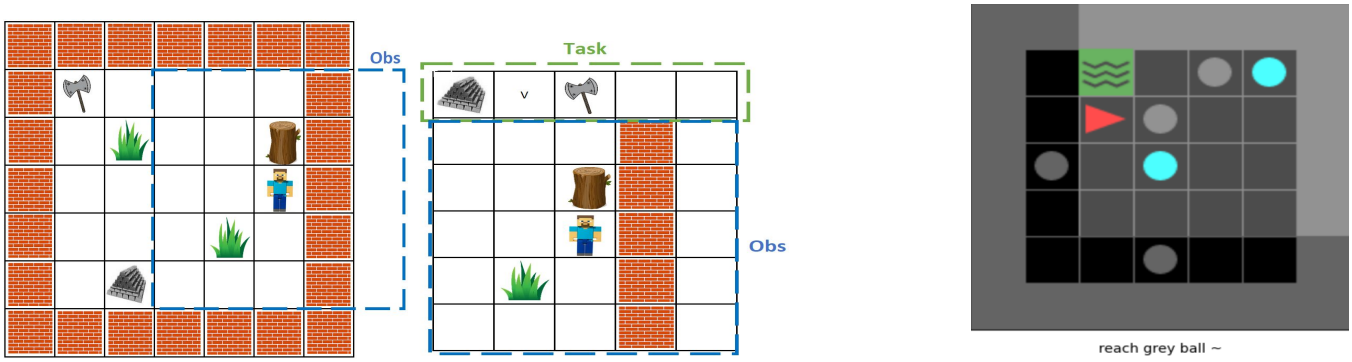

Figure 1: Left. Example of a procedurally generated test map (left) in the Minecraft-inspired environment with its corresponding extended observation (center) when the agent's task is to get either iron or an axe (top row). Tasks are specified in the top row of the extended observation in this setting by depicting the object itself. Right. A MiniGrid map where the task is to reach an object different from the grey ball. Here instructions are given on a separated channel and objects are combinations of shapes and colors.

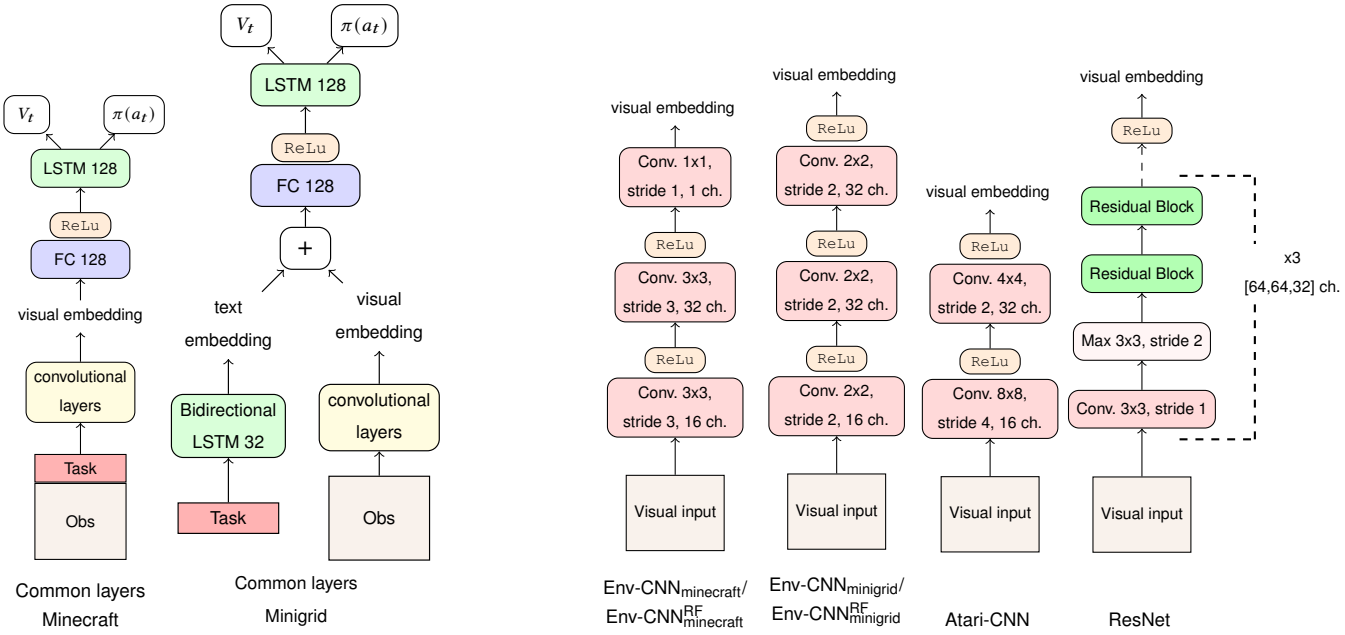

Figure 2: The neural network architectures evaluated in Sec. 4. All the variants follow the structures depicted in the left for the corresponding settings. $\pi(a_t)$ and V_t refer to the actor and critic layers respectively [36]. The different number of channels in the last layer of the Env-CNN architectures is done to provide evidence that such feature does not affect how the agent generalises.

grant a positive reward, an agent that is correctly generalising to zero-shot instructions should collect significantly higher rewards with reliable instructions than with deceptive ones (i.e., the later instructions are pointing to objects that grant a negative reward). Roughly, agents should accumulate with reliable instructions two times greater rewards than when given deceptive instructions to show similar generalisation to the experiments in previous studies about negation with 100 training instructions [21].

The first strong pattern that we extract from Table 2 is that only architectures with some form of alignment (WA and SA) give evidence of systematic learning when no more than six training instructions are available. NA architectures show minimal to none generalisation (e.g. better performance with deceptive instructions than with reliable) in all the small settings. With 20 objects, NA architectures also fail when instructions are visual (Minecraft) while struggling with the disjunction operator when instructions come in a dedicated channel (Minigrid). In the latter setting, architectures with some alignment (WA) show good generalisation performance. Yet, when instructions are visual they only learn to generalise positive and negative instructions, and exclusively in the larges setting. Notably, architectures that are strongly aligned (SA) show good generalisation

Table 1: Mean and standard deviation from the average rewards obtained from five independent runs in a set of 500 maps with atomic tasks. A value of 1.0 is the performance of a random walker. The highest value over all networks is bolded.

Minecraft	Atari-CNN (**NA**)		ResNet (**WA**)		Env-CNN$_{\text{minecraft}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minecraft}}$ (**SA**)	
$\lvert\mathcal{X}_1\rvert$	Train	Test	Train	Test	Train	Test	Train	Test
6	3.04(0.12)	**2.87**(0.21)	3.24(0.49)	2.72 (0.22)	**4.46**(0.75)	1.8(0.25)	2.90(0.81)	2.58(0.76)
20	3.11(0.14)	3.06(0.08)	**4.52**(0.44)	2.14(0.50)	4.45(0.30)	**3.81**(0.72)	3.28(0.98)	3.10(0.88)
Minigrid	Atari-CNN (**WA**)		ResNet (**NA**)		Env-CNN$_{\text{minigrid}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minigrid}}$ (**SA**)	
$\lvert\mathcal{X}_1\rvert$	Train	Test	Train	Test	Train	Test	Train	Test
6	**4.02**(1.09)	**3.96**(0.85)	3.98(0.93)	3.55 (0.69)	2.75(1.03)	2.29(1.03)	3.31(0.84)	3.39(0.31)
20	6.19(0.75)	3.33(1.06)	5.61(0.78)	**5.60**(0.99)	**7.31**(0.50)	4.09(0.92)	4.74(0.15)	2.55(3.39)

Table 2: Results from 500 binary choice maps with zero-shot instructions. In 'Reliable' higher is better while in 'Deceptive' lower is better. Agents show a stronger systematic generalisation the bigger the difference between respective reliable and deceptive instructions. Positive and negative labels refer to atomic positive or negative instruction respectively pointing to one of the two objects present. In disjunction 2$^{\text{nd}}$ choice the instructions refers to two objects, but only the second is present in the map. Disjunction 2 choices points to two objects in a map with 4 items. Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.

Minecraft	$\lvert\mathcal{X}_1\rvert$	Atari-CNN (**NA**)		ResNet (**WA**)		Env-CNN$_{\text{minecraft}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minecraft}}$ (**SA**)	
Instruction		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Positive	6	3.89 (0.12)	3.92(0.09)	3.72(0.49)	3.42(0.81)	**3.25**(1.89)	**0.26**(1.88)	**3.95**(1.8)	**0.74**(0.65)
	20	4.14(0.07)	4.16(0.09)	**3.30**(1.83)	**1.10**(1.38)	**5.43**(0.24)	**0.11**(0.03)	**3.95**(1.76)	**1.10**(1.18)
Negative	6	3.82 (0.18)	3.81(0.10)	3.52(0.52)	3.49(0.68)	**2.20**(1.30)	**0.44**(0.38)	**2.23**(1.89)	**1.04**(0.82)
	20	3.94(0.13)	3.97(0.09)	**3.09**(1.73)	**1.26**(0.93)	**4.40**(0.66)	**0.14**(0.05)	**2.25**(1.80)	**0.94**(1.15)
Disjunction	6	2.56 (0.23)	2.59(0.57)	2.29(0.12)	2.51(0.69)	**2.23**(4.79)	**0.57**(0.29)	2.54(0.67)	1.37(0.06)
2 choices	20	2.63 (0.23)	2.68(0.57)	2.16(0.12)	2.23(0.69)	**3.53**(4.79)	**0.44**(0.29)	**2.82**(0.67)	**1.07**(0.90)

Minigrid	$\lvert\mathcal{X}_1\rvert$	Atari-CNN (**WA**)		ResNet (**NA**)		Env-CNN$_{\text{minigrid}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minigrid}}$ (**SA**)	
Instruction		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Negative	6	**2.83** (0.96)	**0.44**(0.25)	3.02(1.37)	2.97(1.39)	**2.01**(1.27)	**0.41**(0.25)	**1.61**(0.21)	**0.75**(0.16)
	20	**2.77**(0.79)	**0.34**(0.10)	**5.05**(1.27)	**0.20**(0.10)	**5.03**(1.65)	**0.16**(0.05)	**2.16**(0.23)	**0.98**(0.22)
Disjunction	6	**4.21** (0.46)	**0.84** (0.50)	4.02(0.39)	3.68(0.59)	**2.67** (1.49)	**0.79**(0.54)	**4.06**(0.74)	**0.63**(0.44)
2 choices	20	**3.42**(0.97)	**1.14**(0.48)	**5.03**(0.59)	**1.75**(0.77)	**3.13**(0.47)	**0.37**(0.09)	2.72(0.97)	1.58(0.38)
Disjunction	6	**2.83** (0.93)	**0.90**(0.41)	3.52(0.79)	3.86(1.30)	**1.68**(1.25)	**0.75**(0.27)	**2.81**(0.79)	**0.88**(0.55)
2$^{\text{nd}}$ choice	20	2.11(1.42)	1.06(0.32)	2.60(0.74)	2.37(1.03)	**1.81**(0.52)	**0.50**(0.08)	1.91(0.76)	1.28(0.4)

even when the weights of the c.l. are not trained (Env-CNN$^{\text{RF}}$), further evidencing the high influence of the architecture configuration in how the agent generalises. Remarkably, the Env-CNN (the fully trained SA architecture) is the only configuration that systematically achieves a performance greater than 1 (the performance of a random agent) with reliable instructions while getting less than 1 with deceptive instructions across all the settings. That is, *SA architectures are the only configurations that have systematically learnt to generalise from the training instructions to guess not only the desired object/s, but also which undesired items should be avoided according to the test instruction*.

Last, we highlight that in preliminary studies we also explored additional overfitting-prevention methods, including dropout and autoencoders, to help NA networks. However, the results do not vary from the ones obtained by the train-validation-test set division that we carry here and that evidence the benefits of using architectures that are aligned with its environment. We only include the disjunction test where agents struggled the most, due to space constrains, additional results are included in Appendix C.

Generalisation with TTL formulae. We last evaluate the performance of the same agents trained with atomic tasks T when executing complex (multi-task) instructions given as TTL formulae ϕ that refer to zero-shot atomic tasks. Table 4 contains the five complex instructions that we use to test our agent. The first two are

Table 3: Results from a set of 200 maps with zero-shot complex instructions, i.e, longer than training instructions composed by various zero-shot tasks. Steps refers to the number of actions per instruction needed by the agents (a random walker requires 110 steps in average). The highest value over all networks is bolded.

	Atari-CNN (**NA**)		ResNet (**WA**)		Env-CNN$_{\text{minecraft}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minecraft}}$ (**SA**)	
$\lvert\mathcal{X}_1\rvert$	Reward	Steps	Reward	Steps	Reward	Steps	Reward	Steps
6	2.45 (0.25)	**26.28**(4.96)	2.68(0.65)	49.26(28.89)	**3.08**(1.49)	49.1(25.79)	2.74(1.78)	56.51(22.87)
20	2.99(0.21)	**20.96**(1.50)	3.49(0.68)	27.36(5.20)	**7.45**(3.36)	32.84(6.79)	5.49(1.21)	38.18(9.66)

Table 4: Complex instructions used in Table 3 in the main text. The 6 objects included in these instructions belong to the test set ($\mathcal{X}_3$) in the Minecraft-inspired environment.

TTL	Intuitive meaning
$((iron; workbench) \cap wood); toolshed; axe$	Get iron and then use workbench, also get wood. Then use the toolshed. Later use the axe.
$(wood \cap iron); workbench$	Get wood and iron. Then use the workbench
$(grass \sim); grass; (workbench \cup toolshed)$	Get an object different from grass. Then get grass. Later use either the workbench or the toolshed.
$((workbench \sim) \cap (toolshed \sim)); toolshed$	Use an object different from the workbench and use another object different from the toolshed. Then use the toolshed
$((wood; grass) \cup (iron; axe)); workbench; (toolshed \sim)$	Get either wood and then grass or iron and an axe. Then use the workbench. Later use something different from the toolshed.

the hardest specifications from [2, 43]. The three additional instructions were defined by aiming for combinations of negated tasks and inclusive/exclusive non-deterministic choices with non-atomic components together with sequential instances. We recall that operator $\cap$ can be defined in terms of sequential composition ; and non-deterministic choice $\cup$. Table 3 shows the average reward and number of actions needed in 200 maps with the complex instructions. In line with the results with TTL tasks from Table 1, we see that a stronger alignment yields better the final performance (i.e., SA>WA>NA accumulated reward).

5 RELATED WORK

This work bridges the fields of systematic generalisation with formal methods in RL. Here we include the most relevant literature from each field to this research:

Systematic Generalisation. Generalisation beyond training instructions has been widely studied in the RL literature. For instance, Oh et al. [38] presents a framework that relies on hierarchical RL and task decomposition of instructions in English to enhance generalisation in DRL. Later work from Mao et al. [34] introduces a neuro-symbolic concept learner that jointly learns visual concepts, words, and semantic parsing of sentences from natural supervision. Yu et al. [45] present a model that reports strong generalisation in a 2D environment that is aimed at language understanding for positive instructions. Closer to our line of work, there is growing literature (see e.g., Bahdanau et al. [4], Lake [29], Smolensky [40]) focused on finding the right drivers enabling systematic learning from natural language instructions, e.g., fixed perspectives and varied training data. Recent work [21] suggests that systematic generalisation is not a binary question, but an *emergent* property of agents interacting with a situated environment [35] and explores the drivers that enable agents generalise an abstract operator such as negation. We expand the state of this line of works by first, providing evidence that systematic learning can emerge with logic operators from as few examples as with positive instructions, second, giving evidence that the CNN architecture is a key driver –that previous research was oblivious about– towards generalisation.

Reinforcement Learning and Temporal Logic. Training autonomous agents to solve multi-task goals expressed in temporal logic is drawing growing attention from the research community. Examples include following instructions expressed as TL formulae by using logic progression and RL [2, 32], tackling continuous-action environments [46], multi-agent systems [31] and studies on the applicability of different TL languages [7]. Complex goals expressed as temporal formulae can also be easily decomposed by using other techniques, including finite state automata [11, 23] and progression [43]. More recent work has further exploited automata techniques, combining hierarchical approaches and reward machines – a particular form of automata– [16] or high-level planning with low-level RL solutions [26]. These works focus on solving non-Markovian models by extending the state space in a minimal form (to minimally impact the state space) so that the RL agent learns from an equivalent Markovian model, where an optimal behavior is also optimal in the original system [3]. A drawback of the mentioned literature is their exclusive focus on learning to solve compositions of training tasks. For this

reason Kuo et al. [27] targets generalisation of temporal logic instructions with DRL at the expense of requiring an additional neural network for each new symbol or object encountered. Our work advances the state of the art by presenting a framework that can execute zero-shot TL formulae while relying on a single neural network.

6 DISCUSSION AND CONCLUSIONS

We presented a framework that can execute unseen formal instructions while learning from a limited number of atomic tasks. With respect to generalisation, an obvious limitation of our work is the use of regular gridworlds in our experiments. However, this does not lessen our contributions since previous closely-related work have struggled to provide evidence of generalisation with abstract operators in similar settings. Our experiments evidence that *systematic generalisation can emerge with these operators as early as with positive instructions*, and that convolutional layers play a key role in this emergence. Moreover, the good generalisation results with the untrained convolutional layers in the two settings suggests that finding the right architecture may be more beneficial than training a generic convolutional configuration when aiming at generalisation. Consequently, when working in real-world environments – where the prior-knowledge about object resolution cannot be directly applied – our research suggests that methods for automatic generation of deep learning architectures are a potential way forward. Another limitation is that — in line with related work on generalisation [4, 21, 28] – we do not give nomological explanations for the generalisation ability (or lack thereof) that we evidence. Yet, this out of our scope, that is proving the emergence of generalisation and the key elements lacked in previous works in similar settings. Our findings suggest that from as few as six different examples agents with aligned architectures learn not only how to follow training commands, but also abstract information about how symbols and operators in the given language compose and how the combination of those symbols influences what the agent should do. This contrasts with Hill et al. [21] – the only previous study evidencing some generalisation with unseen negated instructions – which suggests that ∼100 instructions are required with negated instructions. Using the same form of negation, we find that the CNN architecture plays a key role when learning from limited examples. Still, we observe that the ResNet used in Hill et al. [21] shows evidence of generalising negation with 20 instructions in our experiments. This also contrasts with Hill et al. [21], where the ResNet performed barely better than random when trained with 40 negated instructions. We find that this difference comes from our use of a validation set, not common in RL literature and consequently not present in previous research. We provide empirical evidence of this point in Appendix D.

With respect to formal methods and RL (FM-RL) literature, we use a language that is less expressive than the latest contributions in the field. This constraint was needed to carry a careful experimental evaluation of the systematic learning with the different types of operators. Given the positive results, more expressive languages can be used in future iterations. Nevertheless, we presented the first framework that, leveraging on a learning-oriented syntax and the compositional structure of logical formulae, is capable of executing zero-shot complex formulae in temporal logic while relying on a single network. As anticipated in Sec. 5, this contrasts with dominant methods in the FM-RL literature that rely on a new policy network per task (e.g., De Giacomo et al. [11], Furelos-Blanco et al. [16], Icarte et al. [24], Jothimurugan et al. [26]) or a new network per word and symbol [27].

Concluding remarks. We presented a foundational study that acts as a bridge between the areas of DRL, FM and systematic generalisation. Our framework demonstrates the possibility of generalising to unseen multi-task TL formulae, where future works may apply more sophisticated symbolic modules such as reward machines. We have empirically demonstrated that systematic learning can emerge with abstract operators such as negation from as few as six instructions. Hence, we plan exploring more expressive language without requiring large numbers of training instructions for new symbols. In the context of generalisation, we also gave evidence of the important role that convolutional layers play in how agents generalise. In our case, exploiting prior-knowledge of the environment – the tile resolution of the gridworld – was key to achieve better agents. Our findings highlight that choosing the right convolutional layers can play a more important role than training such layers. This suggests that future work may explore meta-learning and evolutionary techniques [13, 41], which provide methods to automatically search well-suited architectures, as a potential solution when targeting generalisation in visually complex environments.

7 APPENDIX

We refer readers to https://drive.google.com/file/d/1pU1n7b4n4FCI9HTmoUyywniZZZGKIEwt/view?usp=sharing for the appendix.

REFERENCES

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[2] Jacob Andreas, Dan Klein, and Sergey Levine. 2017. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 166–175.

[3] Fahiem Bacchus, Craig Boutilier, and Adam Grove. 1996. Rewarding behaviors. In *Proceedings of the National Conference on Artificial Intelligence*. 1160–1167.

[4] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron C. Courville. 2019. Systematic Generalization: What Is Required and Can It Be Learned?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[5] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT press.

[6] Ronen I Brafman, Giuseppe De Giacomo, and Fabio Patrizi. 2018. LTLf/LDLf nonmarkovian rewards. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[7] Alberto Camacho, R Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2019. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. 6065–6073.

[8] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

[9] Noam Chomsky and David W Lightfoot. 2002. *Syntactic structures*. Walter de Gruyter.

[10] John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, Pieter Abbeel, and Sergey Levine. 2019. Meta-learning language-guided policy learning. In *International Conference on Learning Representations*, Vol. 3.

[11] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. 2019. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 29. 128–136.

[12] Giuseppe De Giacomo and Moshe Y Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

[13] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. 2019. Neural architecture search: A survey. *J. Mach. Learn. Res.* 20, 55 (2019), 1–21.

[14] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. 80 (2018), 1406–1415.

[15] Jerry A Fodor and Zenon W Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition* 28, 1-2 (1988), 3–71.

[16] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. 2021. Induction and exploitation of subgoal automata for reinforcement learning. *Journal of Artificial Intelligence Research* 70 (2021), 1031–1116.

[17] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. *Neural computation* 12, 10 (2000), 2451–2471.

[18] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[19] Lewis Hammond, Alessandro Abate, Julian Gutierrez, and Michael Wooldridge. 2021. Multi-Agent Reinforcement Learning with Temporal Logic Specifications. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 583–592.

[20] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. 2021. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. In *The Thirty-Fifth {AAAI} Conference on Artificial Intelligence, {AAAI}*, Vol. 2. 36.

[21] Felix Hill, Andrew Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L McClelland, and Adam Santoro. 2020. Environmental drivers of systematicity and generalization in a situated agent. In *International Conference on Learning Representations*.

[22] Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. 2021. Grounded Language Learning Fast and Slow. In *International Conference on Learning Representations*.

[23] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*. 2112–2121.

[24] Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. 2019. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 15497–15508.

[25] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. 2019. A Composable Specification Language for Reinforcement Learning Tasks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 13021–13030.

[26] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. 2021. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems* 34 (2021).

[27] Yen-Ling Kuo, Boris Katz, and Andrei Barbu. 2020. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 5604–5610.

[28] Brenden Lake and Marco Baroni. 2018. Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks. In *International Conference on Machine Learning*. 2873–2882.

[29] Brenden M Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems*. 9788–9798.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[31] Borja G. León and Francesco Belardinelli. 2020. Extended Markov Games to Learn Multiple Tasks in Multi-Agent Reinforcement Learning. In *ECAI 2020 - 24th European Conference on Artificial Intelligence (Frontiers in Artificial Intelligence and Applications, Vol. 325)*. IOS Press, 139–146. https://doi.org/10.3233/FAIA200086

[32] Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. 2017. Environment-independent task specifications via gltl. *arXiv preprint arXiv:1704.04341* (2017).

[33] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. 2019. A Survey of Reinforcement Learning Informed by Natural Language. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, 6309–6317. https://doi.org/10.24963/ijcai.2019/880

[34] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. 2019. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[35] James L McClelland, Matthew M Botvinick, David C Noelle, David C Plaut, Timothy T Rogers, Mark S Seidenberg, and Linda B Smith. 2010. Letting structure emerge: connectionist and dynamical systems approaches to cognition. *Trends in cognitive sciences* 14, 8 (2010), 348–356.

[36] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[38] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. 2017. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2661–2670.

[39] Thiago D Simão, Nils Jansen, and Matthijs TJ Spaan. 2021. AlwaysSafe: Reinforcement learning without safety constraint violations during training. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 1226–1235.

[40] Paul Smolensky. 1988. On the proper treatment of connectionism. *Behavioral and brain sciences* 11, 1 (1988), 1–23.

[41] Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. 2020. Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 3769–3776. https://doi.org/10.1109/IROS45743.2020.9341571

[42] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[43] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2018. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 452–461.

[44] Yichen Yang, Jeevana Priya Inala, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, and Martin Rinard. 2021. Program Synthesis Guided Reinforcement Learning. In *Advances in neural information processing systems*.

[45] Haonan Yu, Haichao Zhang, and Wei Xu. 2018. Interactive Grounded Language Acquisition and Generalization in a 2D World. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[46] Lim Zun Yuan, Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2019. Modular Deep Reinforcement Learning with Temporal Logic Specifications. *arXiv preprint arXiv:1909.11591* (2019).