Relational Deep Reinforcement Learning and Latent Goals for Following Instructions in Temporal Logic

Borja G. León [1] **Murray Shanahan** [1] **Francesco Belardinelli** [1]

Abstract

We address the problem of building agents aiming to satisfy out-of distribution (OOD) multi-task instructions expressed in temporal logic (TL) through deep reinforcement learning (DRL). There exists evidence that the deep learning architecture is a key feature when teaching a DRL agent to solve OOD tasks in TL. Yet, the studies on the performance of different networks are still limited. Here, we first study the impact of introducing relational layers and key-value attention in OOD safety-aware navigation TL tasks. Then, we propose a novel architecture configuration that induces a DRL agent to generate latent representations of its current task given its perception of the environment and the formal instruction. We find that inducing this latent goal representation significantly improves the performance of neural-based agents in OOD environments.

1. Introduction

Building autonomous agents capable of learning from human instructions and apply this knowledge in a compositional fashion is crucial goal of artificial intelligence (Lake, 2019; Hill et al., 2021). Lately, this goal is mainly pursued through deep reinforcement learning (DRL) agents following instructions expressed in natural language (Yu et al., 2018; Mao et al., 2019). Unfortunately, generalisation in DRL is linked to training the autonomous agents with large numbers of instructions, requiring manually building natural expressions with corresponding reward functions, which prevents them from scaling well (Vaezipoor et al., 2021).

This issue motivates ongoing research on agents learning from formally specified instructions (Alshiekh et al., 2018; Simão et al., 2021) as a replacement of natural language. Formal languages (Huth & Ryan, 2004) offer desirable properties such as unambiguous semantics and compositional syntax, allowing to automatically generate large amounts of training instructions and reward functions. Earlier contributions in this area rely on the compositional nature of the language, using large numbers of policy networks to execute temporal logic (TL) instructions (Andreas et al., 2017; Icarte et al., 2018; Kuo et al., 2020). Recent works (León et al., 2020; Vaezipoor et al., 2021) have presented DRL frameworks that are capable of generalizing to out-of-distribution (OOD, i.e., never seen during training) instructions while relying on a single policy network. Those studies provide evidence that the particular neural network architecture within the DRL agent is a key feature towards the final performance of the agent. Still, we believe that no work has studied the impact of networks using relational layers (Santoro et al., 2018) and the key-value attention mechanism (Chen et al., 2019) which have been successful in other domains when tackling generalisation (Zambaldi et al., 2018).

Contributions. We study the generalisation performance of DRL agents following formal instructions in OOD environments, and explore the best network configurations in a widely-used navigation benchmark (Andreas et al., 2017; Toro Icarte et al., 2018; De Giacomo et al., 2019; León et al., 2020; Vaezipoor et al., 2021). More specifically, our contributions are as follows: 1) We are first to assess the extent whereby various relational and attention-based modules are better suited than fully-connected layers to follow safety-aware OOD instructions in temporal logic. We find that, despite using a benchmark that encourages relational reasoning, a simple multi-layer perceptron (MLP) outperforms various complex relational and attention-based modules when using standard architectures. 2) We propose a novel architecture configuration that induces the neural network to generate latent representations of its current objective, and significantly improves the performance of all the networks in the hardest OOD environments (up to 74%).

2. Background

Reinforcement Learning. Our p.o. environment is modelled as a p.o. Markov decision process POMDP, a tuple $\mathcal{M} = \langle S, A, P, R, Z, O, \gamma \rangle$ where (i) S is the set of

[1]Department of Computing, Imperial College London, London, United Kingdom. Correspondence to: Borja G. León <b.gonzalez-leon19@imperial.ac.uk>.

Presented at the ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning., Copyright 2021 by the author(s).

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) $R : S \times A \times S \to \mathbb{R}$ is the reward function. (v) Z is the set of observations $z, z', \ldots$. (vi) $O : S \times A \times S \to Z$ is the observation function. (vii) $\gamma \in [0,1)$ is the discount factor. Through the given observations $z_t \in Z$, where t refers to a time step, the goal of the RL agent is to choose a policy π that for every state selects the action that maximizes the expected return $Q(s,a) = \mathbb{E}[R_t \mid s_t = s, a]$ (Sutton & Barto, 2018).

We also make use of *relational networks*, a particular form of deep learning (Goodfellow et al., 2016) that incorporates layers whose structures is explicitly designed for *relational reasoning*. Due to space constraints we refer to Santoro et al. (2017) for details about relational networks, and Bahdanau et al. (2015) for the key-value attention mechanism, which have proved to be an effective addition to relational networks (Zambaldi et al., 2018; Shanahan et al., 2020).

3. Learning to Solve SATTL Instructions

The following section details the features shared across all the agents tested in this work. We first introduce the specific formal language used to procedurally generate instructions. Then we detail the symbolic and neural modules used by our framework. Last, we explain the empirical environment.

We are interested in agents learning compositionally TL instructions in a similar fashion to León et al. (2020) with task temporal logic (TTL), a learning-oriented TL language interpreted over finite episodes, which was originally intended to deal only with reachability goals. However, here we tackle safety constraints as well. Hence, we extend TTL to safety-aware task temporal logic (SATTL).

Definition 1 (SATTL)**.** *Let AP be a set of* propositional atoms*. The sets of* literals l*,* atomic tasks α*, and* temporal formulas T *in SATTL are inductiveley defined as follows:*

$$\begin{aligned} l &::= +p \mid -p \mid l \vee l \\ \alpha &::= lUl \\ T &::= \alpha \mid T;T \mid T \cup T \end{aligned}$$

Literals are positive ($+p$) or negative ($-p$) propositional atoms, or disjunctions thereof. Atomic tasks α are obtained by applying the temporal operator *until* (U) to pairs of literals. An atom $\alpha = lUl'$ is read as *it is the case that l until l' holds*. Temporal formulas T are built from atomic tasks by using sequential composition ($;$) and non-deterministic choice ($\cup$). We use an explicit positive operator ($+$) so that both positive and negative tasks have the same length. This is beneficial for the learning process of the negation operator, as highlighted in (León et al., 2020) with TTL. As TTL, SATTL is interpreted over finite traces, i.e., finite sequences of states and actions. Details about semantics and satisfaction is given in Appendix D. Intuitively, an atomic task $\alpha = c_\alpha U g_\alpha$ is satisfied if the "safety" condition c_α remains true until g_α is fulfilled. In the context of this work we are not interested in strict safety warranties but in agents that are trying to reach a goal while trying to minimize the violation of additional conditions. Thus, we may say our agent has satisfied α while it has not fulfilled c_α at every time step before g_α. Formally, in those cases we are considering traces that are not starting at the beginning of the episode. An example of a task we use is $\alpha_1 = -\text{soil} U (+\text{axe} \vee +\text{sword})$ "avoid soil until you reach an axe or a sword".

Our framework consists of a *symbolic module* (SM) and a *neural module* (NM). Given a formal instruction T in SATTL, the SM decomposes T into a sequence of atomic tasks α to be solved by the NM, which is instantiated as a DRL algorithm. Once α is selected, the SM uses an internal labelling function $\mathcal{L}_\mathcal{I} : Z \to 2^{AP}$ that maps observations z to truth evaluations p. Intuitively, $\mathcal{L}_\mathcal{I}$ acts as an *event detector* that fires when the propositions in AP hold. $\mathcal{L}_\mathcal{I}$ allows the SM to generate a reward function R_I for the NM:

$$R_I(p_t) = \begin{cases} -0.05 & \text{if } p_t = c_\alpha; \\ 1 & \text{if } p_t = g_\alpha; \\ -1 & \text{otherwise.} \end{cases}$$

where $\mathcal{L}_I(z_t) = p_t$ for any time step t. For further detail about the SM see Appendix B. The neural module consists of a DRL algorithm that interacts with the environment given the current observation and task, aiming to maximize the cumulative reward from R_I. All the variants for the NM use the A2C algorithm, a synchronous version of A3C (Mnih et al., 2016) sharing the same hyperparemeters and encoder, which are detailed in Appendix C. Note that the encoder has been designed bearing in mind the resolution of the tiles within so that the embedding (i.e., the outputs of the encoder) of the tasks are independent from the embedding of the original observation z. This inductive bias is exploited in the architectures presented in Sec. 4.

Experimental setting. For the empirical evaluation we use Minecraft-inspired environments (Andreas et al., 2017). These are 2D grid-worlds (see Fig. 1 left), where the agent has 4 possible actions $Up, Down, Left, Right$ to move one square from its current position, according to the given direction. Specifically, we use a configuration where maps are procedurally generated by randomly selecting objects and starting positions, and where each tile within has a resolution of 9x9 values. At each time step the agent receives an egocentric observation extended with the current task to be solved (see Fig. 1 right). Since our goal is to test autonomous agents with OOD objects and environment sizes, we modified the environments to have maps of variable size. While training, the generated maps have a size in the range

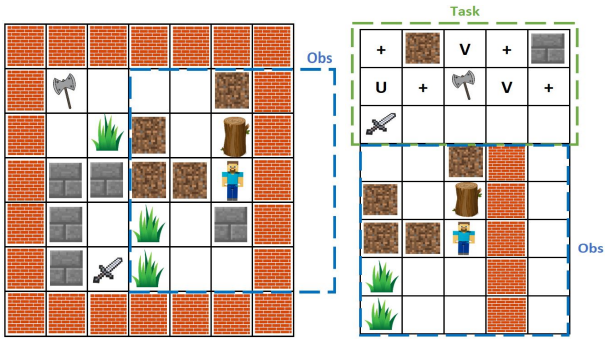

Figure 1. Example of a procedurally generated training map (left) of size 7x7 with its corresponding extended observation (right) with the task "move through either soil or stone until reaching an axe or a sword". Tasks are specified by depicting the objects themselves and the SATTL operators. Each position in the grid has a resolution of 9x9 values. The total input size is 72x45.

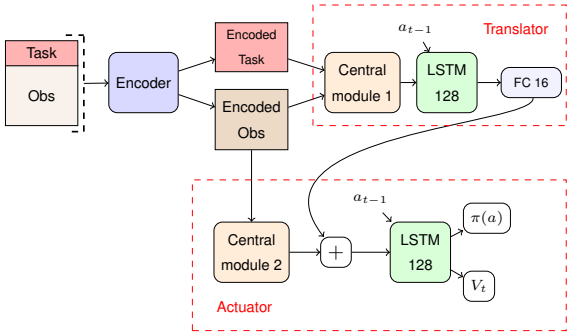

Figure 2. Architecture with inductive latent goals. The *translator* modules generate a latent goal given the SATTL task and the observation, which is concatenated with the output of central module 2 (CM2) of the *actuator*. CM2 has no information of the task.

[7-10] squared. In test, maps can be either of size 7x7, 14x14 or 22x22. The global set of objects is referred to as $\mathcal{X}$, where the total number of objects is $|\mathcal{X}| = 50$. The set is partitioned into a pretraining set $|\mathcal{X}_1| = 35$ a training set $|\mathcal{X}_2| = 20$ and a test set $|\mathcal{X}_3| = 15$, where $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_3$, $\mathcal{X}_2 \subset \mathcal{X}_1$ and $\mathcal{X}_1 \cap \mathcal{X}_3 = \emptyset$. Note that these environments are specially suited for agents capable of relational reasoning (Kemp & Tenenbaum, 2008), i.e., for an agent to perform well it needs to 1) *relate* the objects and symbols within the instruction and their relative positions and 2) relate the instruction with what it perceives around itself. For more details about the experiments we refer to Appendix C.

4. Relational Networks and Latent Goals

In this section we include the main contributions of our work. First, we contrast the performance of various relational networks and provide evidence that, despite their theoretical advantages, only specific relational modules outperform a basic MLP following temporal logic instructions when used within a standard architecture. Then we introduce inductive latent goals, an inner sparse goal communication architecture that yields stronger generalisation in OOD environments. Last we show the potential benefits of inducing latent goals in existing hierarchical recurrent architectures using a sparse attention mechanism for communication.

4.1. Relational Networks in Standard Architectures

Literature about relational networks applied to autonomous agents commonly follows a standard network architecture (see Figure 4 in Appendix) where the output of the encoder is passed through a central module (CM), which in our context is either a relational layer or an MLP, followed by a recurrent layer (where we use an LSTM). In this section we evaluate four different networks for the CM. The first one is a relation net (RN) from (Santoro et al., 2017). The second network is a multi-head attention net (MHA) (Zambaldi et al., 2018), which combines relational connections with key-value attention. The third one is a PrediNet (Shanahan et al., 2020), that also combines relational and attention operations but specifically designed to form propositional representations of the output. For more details see Appendix C.3. The fourth module is an MLP consisting of a single FC layer, which is meant to act as baseline. In order to have a meaningful comparison between the different CMs, every CM has the same input and output size.

Table 1 (up) shows the results obtained by the four networks. Overall we see that the chosen CM has a significant impact in the final performance. Notably the MHA is the only "specialized" network that clearly outperforms a basic MLP baseline when using a standard architecture.

4.2. Inducing Latent Goals

We propose improving standard architecture configurations by modifying the input of the last recurrent layer and using two input streams in that layer: 1) a task-agnostic or general sensory input, that excludes the given instruction; 2) a *sparse* latent representation of the current goal that uses the whole input (sensors and instruction). Figure 2 depicts the proposed configuration with the networks studied in Sec. 4 when using an small FC layer (acting as a bottleneck) for the latent goal (LG). Note that we assume to be able to split the instruction from the rest of the observation. Yet, we believe this is not a strong limitation since most of the existing frameworks with DRL agents tackling human-given instructions, either formally expressed (e.g., temporal logic) or in natural language, work by communicating the tasks through a separated instruction channel (Hill et al., 2020) or through a handcrafted extension (Icarte et al., 2019).

Intuitively, we propose a configuration that aims to induce the situated agent to "interpret" the human instructions in

Table 1. Results with either training (train) and unseen (test) objects in 500 maps (per size) of different dimensions. Note that sizes 14 and 22 are OOD for all the agents. Results are mean and standard deviation from 10 independent runs (i.r.). Values are normalized so that 100 refers to the highest performance achieved by the best run globally in maps of that size.

Map size	MLP		RN		MHA		PrediNet		Random
	Train	Test	Train	Test	Train	Test	Train	Test	
7x7	33.4(7.9)	21.1(3.2)	5(1.9)	3.2 (1.6)	41.1(7.7)	36.6(4.3)	21.8(3.7)	16.1(3.5)	3.7(0.1)
14x14	38.7(6.5)	32(4.5)	8.3(2)	7.9(2.4)	51.7(9.3)	45.8 (7.6)	40.2(5)	31.9(6.2)	11.5(1.4)
22x22	47.8(3.6)	37.8(3.8)	9.2(1.5)	8.3(1.8)	56.9(10.7)	49.2(5)	48.7(10.3)	38(8.5)	11.1(0.5)
Map size	MLP^{LG}		RN^{LG}		MHA^{LG}		$\text{PrediNet}^{\text{LG}}$		
	Train	Test	Train	Test	Train	Test	Train	Test	
7x7	40.2(5.7)	24.7(2.6)	4.4(1)	4.2 (0.7)	35.2(5)	31.6(5.3)	55.2(17.6)	29.5(8.2)	
14x14	41.7(5.1)	35.4(4.2)	13.1(2.6)	11.5(2.2)	48.1(4.4)	41.8 (4.9)	67(5.2)	51.7(6.5)	
22x22	52.6(8.6)	41.3(7.7)	11.7(1.5)	10.7(2.1)	58.4 (4.5)	51.5(5.5)	77.5(9.1)	65.3(6.1)	

Table 2. Five i.r. of a vanilla BRIM (BRIM), a BRIM with a residual connection (ResBRIM) and a BRIM with latent goals (BRIM^{LG}).

Map size	BRIM		ResBRIM		BRIM^{LG} (ours)	
	Train	Test	Train	Test	Train	Test
7x7	27.9(6.3)	15.6(2.8)	58.6(10.4)	15.8 (2.3)	77.2(19.5)	26.4(7.0)
14x14	37.9(8.2)	26.5(6.7)	63.5(5.5)	39.6(7.0)	77.3(21.3)	46.8 (8.1)
22x22	41.5(8.4)	29.3(4.1)	60.6(10.4)	40.7(8.2)	79.8(17.0)	58.4(14.8)

its current context and "sparsely" communicate a latent goal to the network layers deciding the action to take. Sparsity in the latent goal communication forces the network to produce more general representations of the task, and is achieved either through a relatively small FC layer (which acts as a communication bottleneck) or a *top-k attention mechanism* (see Section 4.3). We believe that inducing latent goals may become a default architecture configuration for multi-task agents aimed for OOD generalisation.

From table 1 (down) we see that inducing latent goals improves the performance of all the networks in the hardest test environments (22x22). Unlike with the standard architectures, here all the relational modules that use attention (MHA and PrediNet) outperform an MLP. Notably, the two most opposed results come from the these two networks (MHA and PrediNet). From our results in an ablation study (see Appendix A), we find that the element-wise comparison in the detected objects is a key feature within the PrediNet yielding a better performance.

4.3. Latent goals in hierarchical architectures

We demonstrate that inducing latent goals can improve the performance of hierarchical architectures. Specifically, we show that giving both instruction and visual input to the bottom layer of a hierarchical network, while giving a task-agnostic representation of the environment to the top layer, improves the performance of the architecture. Here we fix the PrediNet as the central module and substitute the LSTMs by a bidirectional independent recurrent network (BRIM) (Mittal et al., 2020). A BRIM is a modularized hierarchical network that has shown strong OOD generalisation performance in several tasks. A particularly interesting feature of BRIMs in this work is that they employ a sparse attention mechanism to communicate between different hierarchies of layers. Specifically, each layer is composed of various modules, and at each time step only the top k modules (according to the key-value attention mechanism) will be able to communicate with modules from different layers, where k is a hyperparameter. An illustration of the architectures using BRIMs is given in Appendix C.3. Table 2 contrasts the performance when using a vanilla BRIM layer against using a residual connection (He et al., 2016) or latent goals. We see that inducing the latent goals significantly improves the performance of the hierarchical network in all the maps.

5. Conclusions

Inducing neural architectures to generate sparsely-communicated latent goals can yield stronger generalisation when agents are aiming to solve novel instructions in OOD environments. This includes deep hierarchical networks such as BRIMs where latent goals proved to be more beneficial than classic residual connections. We believe that these findings may have a broad impact within the communities working with DRL, OOD generalisation and formal methods. Future work can investigate how well DRL agents generalize when the symbolic part cannot always provide reliable feedback on task progression.

References

Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. Safe reinforcement learning via shielding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 166–175. JMLR. org, 2017.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Chen, Y., Dong, C., Palanisamy, P., Mudalige, P., Muelling, K., and Dolan, J. M. Attention-based hierarchical deep reinforcement learning for lane change behaviors in autonomous driving. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pp. 3697–3703. IEEE, 2019. doi: 10.1109/IROS40897.2019.8968565.

Chomsky, N. and Lightfoot, D. W. *Syntactic structures*. Walter de Gruyter, 2002.

De Giacomo, G., Iocchi, L., Favorito, M., and Patrizi, F. Reinforcement learning for ltlf/ldlf goals. *arXiv preprint arXiv:1807.06333*, 2018.

De Giacomo, G., Iocchi, L., Favorito, M., and Patrizi, F. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pp. 128–136, 2019.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90.

Hill, F., Lampinen, A., Schneider, R., Clark, S., Botvinick, M., McClelland, J. L., and Santoro, A. Environmental drivers of systematicity and generalization in a situated agent. In *International Conference on Learning Representations*, 2020.

Hill, F., Tieleman, O., von Glehn, T., Wong, N., Merzic, H., and Clark, S. Grounded language learning fast and slow. In *International Conference on Learning Representations*, 2021.

Huth, M. and Ryan, M. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

Icarte, R. T., Klassen, T., Valenzano, R., and McIlraith, S. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pp. 2112–2121, 2018.

Icarte, R. T., Waldie, E., Klassen, T., Valenzano, R., Castro, M., and McIlraith, S. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15497–15508, 2019.

Kemp, C. and Tenenbaum, J. B. The discovery of structural form. *Proc. Natl. Acad. Sci. USA*, 105(31):10687–10692, 2008. doi: 10.1073/pnas.0802631105.

Kuo, Y., Katz, B., and Barbu, A. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. *CoRR*, abs/2006.01110, 2020.

Lake, B. M. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems*, pp. 9788–9798, 2019.

León, B. G., Shanahan, M., and Belardinelli, F. Systematic generalisation through task temporal logic and deep reinforcement learning. *arXiv preprint arXiv:2006.08767*, 2020.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. 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, 2019.

Mittal, S., Lamb, A., Goyal, A., Voleti, V., Shanahan, M., Lajoie, G., Mozer, M., and Bengio, Y. Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6972–6986. PMLR, 2020.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P. W., and Lillicrap, T. A simple neural network module for relational reasoning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 4967–4976, 2017.

Santoro, A., Faulkner, R., Raposo, D., Rae, J. W., Chrzanowski, M., Weber, T., Wierstra, D., Vinyals, O., Pascanu, R., and Lillicrap, T. P. Relational recurrent neural networks. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 7310–7321, 2018.

Shanahan, M., Nikiforou, K., Creswell, A., Kaplanis, C., Barrett, D. G. T., and Garnelo, M. An explicitly relational neural network architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8593–8603. PMLR, 2020.

Simão, T. D., Jansen, N., and Spaan, M. T. Alwayssafe: Reinforcement learning without safety constraint violations during training. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1226–1235, 2021.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 452–461. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Vaezipoor, P., Li, A., Icarte, R. T., and McIlraith, S. Ltl2action: Generalizing ltl instructions for multi-task rl. *International Conference of Machine Learning (to appear)*, 2021.

Yu, H., Zhang, H., and Xu, W. 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, 2018.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., et al. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2018.

A. Ablation studies

A.1. A Study on PrediNet^{LG}

This section is devoted to do a deeper analysis on the performance of attention-based relational networks when generating latent goals. Specifically, we aim to identify what feature (or features) within the PrediNet makes it a better choice for architectures inducing latent goals as suggested by the results in Sec. 4.2. We tackle this analysis by extending the contrast of the MHA (Zambaldi et al., 2018) and PrediNet (Shanahan et al., 2020) networks since both use relational and attention layers. Theoretically, we observe that the most significant design difference comes from a channeling of the input information present only within the PrediNet. That network encourages a kind of the semantic separation of the representations that it learns, making an element-wise subtraction between the detected elements not present in the MHA. Additionally, we see that the PrediNet modifies its output to explicitly represent the positions of the pair of features selected by each of its heads. Also this operation is not present in the MHA.

Table 3 shows the impact of eliminating the cited features. In PrediNet$^{LG}_{noSub}$ we replaced the element-wise subtraction by an FC layer, while in PrediNet$^{LG}_{noPos}$ the position mapping is replaced by a constant value. We also tested two hybrid networks called PNMHA^{LG} (using a PrediNet in CM1 to generate the latent goal and an MHA in CM2 for the task-agnostic representation) and MHAPN^{LG} (with an MHA in CM1 and a PrediNet in CM2). The hybrid networks serve to verify whether the PrediNet is better suited only for a specific central module or not. Contrasting the test results from PrediNet$^{LG}_{noSub}$ and PrediNet$^{LG}_{noPos}$ (Table 3) with the ones from PrediNet^{LG} (Table 1) we see that eliminating the output positions (PrediNet$^{LG}_{noPos}$) had a smaller impact in the performance (a drop in the range [0.2-8.9]) w.r.t. replacing the element-wise subtraction by an FC layer (PrediNet$^{LG}_{noSub}$, a drop in the range [6.7-18.9]). Moreover, by removing the element-wise subtraction, the PrediNet$^{LG}_{noSub}$ has a similar performance to one of an MHA^{LG} in the hardest OOD maps. All of this suggests that the element-wise comparison is a key feature that enhances the PrediNet to be used within architectures inducing latent goals. Moving to the results of the PNMHA^{LG} and MHAPN^{LG} from Table 3, we see that MHAPN^{LG} tends to show better performance than PNMHA^{LG}. Yet, the best results still come from using a PrediNet in both central modules, i.e., from using the PrediNet to generate both data-streams (latent goals and task-agnostic).

A.2. Compositional Learning

Here we want to assess whether our agents are learning compositionally from the SATTL instructions and whether they are following the given safety constraints when applied to OOD objects or they are simply learning to navigate towards the goal. Moreover, it is of special relevance to analyse their performance with negated constraints. Compositional learning (Lake, 2019) (also known as systematic learning (Hill et al., 2020)) refers to the ability of understanding and producing novel utterances by combining already known primitives (Chomsky & Lightfoot, 2002). In our context, an agent that is learning compositionally should be able to solve the instruction $-c\,U + p'$ if the primitives c and p are known and it already learnt to solve $-c'U + p'$.

Table 4 shows the results from a control experiment, where we track the performance of the agents according to tasks of the form $-c\,U + p$ when giving reliable, partially occluded or deceptive instructions. Specifically, every row shows the test performance of the agents trained in Sec. 4 and 4.2 when providing rewards according to the instruction $-c\,U + p$, which intuitively means "avoid c until reaching p". The first row shows the performance of the agents when the SM provides *reliable instructions* to the NM, i.e., the instruction provided is the one used to provide rewards. The second row shows the performance under the same setting, but where the extended observation provided to the NM is a reachability goal (true $U + p$). Intuitively, for the second row the agents are given the right goal p but not the safety constraint, i.e., partially occluding information of the real task. The third row shows the performance when the SM gives deceptive information abut the safety constrain ($+cU + p$), i.e., the SM is "telling" the NM to go through the objects that actually have to be avoided. Agents learning systematically should show worse performance with partially occluded instructions than with the reliable ones, but still better than a random walker. Additionally, the worst performance should be when provided deceptive instructions.

From Table 4 we see that all the variants using the MHA and MLP modules follow the rule 1^{st} row $> 2^{nd} > 3^{rd}$ (compositional rule, or c.r. for short). Notably, the best performance comes from a variant that does not use relational networks nor attention (the MLP^{LG}). Still, this does not imply that MLPs are better suited for negation since agents were trained in a much wider variety of tasks and the MHA, MHA^{LG} and PrediNet^{LG} outperformed the MLP^{LG} in the general evaluation test. In the case of the RN, the c.r. is satisfied but the general low performance and the small difference between the results of the first and the second rows (14 and 13.3 respectively) indicates a weak generalisation. This is worse with the RN^{LG} and the PrediNet having both equal or better performance with partially occluded instructions than when receiving the real task as input. This suggests that these networks are not correctly generalising negated instructions. This is not the case for the PrediNet^{LG}, whose results correctly follow the c.r.. In addition, the worse performance of the PrediNet^{LG} than the MHA with partially occluded and deceptive instructions suggests that

Table 3. Ablation study of PrediNet^{LG} (5 i.r. per variant). PrediNet$^{LG}_{noSub}$ and PrediNet$^{LG}_{noPos}$ are variants of PrediNet^{LG} that eliminate the element-wise subtraction and the feature coordinates respectively. PNMHA^{LG} uses an MHA in the CM2 of PrediNet^{LG}, while MHAPN^{LG} uses an MHA in CM1.

Map size	PrediNet$^{LG}_{noSub}$		PrediNet$^{LG}_{noPos}$		PNMHA^{LG}		MHAPN^{LG}	
	Train	Test	Train	Test	Train	Test	Train	Test
7x7	41(8)	22.8(2.8)	50.6(9.9)	29.3 (5)	39.3(7.6)	29.2(6.2)	49(10)	36.6(7.5)
14x14	64.7(7.9)	42.3(6.9)	68.2(10.8)	48.1(5.4)	52.8(3.9)	46.5 (7.9)	59.9(9.8)	45.6(4.9)
22x22	63.7(7.8)	46.4(8.1)	70.6(9.3)	56.4(9.2)	64.6(6.7)	45.2(10)	57.4(6.4)	51.6(7.6)

Table 4. Study on compositional learning with zero-shot objects and instructions. Results show the performance obtained by each network in 200 test maps when rewards are given according to the "real goal", while the symbolic module provides the "given instruction" to the neural module. An agent learning compositionally should have 1^{st} row $> 2^{nd}$ row $> 3^{rd}$ row for the values within its column. Also, values lower than random are only acceptable in the third row (deceptive instruction).

Real goal: $-cU+p$									
Given instruction	MLP	M-MLP	RN	M-RN	MHA	M-MHA	PrediNet	M-PrediNet	Random
$-cU+p$	44.6(10.5)	70.6(11.9)	14(1.1)	13.9 (1)	55.8(6.1)	43.8(7.3)	17.5(1.2)	43.6(9.3)	12.5(0.7)
true $U+p$	20.3(5.7)	18.0(4.6)	13.3(0.9)	13.9(1.4)	16.9(3.0)	19.9 (6.3)	19.9(3.4)	17.5(3.8)	12.5(0.7)
$+cU+p$	8.7(1.6)	11.0(2.6)	7.9(2.2)	5.6(1.2)	10.8(2.6)	15.0(3.6)	8.2(1.8)	8.4(1.2)	12.5(0.7)

the PrediNet^{LG} advantage over the MHA in maps of OOD size that we saw in Table 1 comes from a better ability of the former to satisfy constraints when compared with the later. Note that the larger the map the harder it is to find p; thus the bigger chances of accumulating penalizations due to "violations" of the task.

Discussion To the best of our knowledge, the only two works that include some empirical evidence of emergent compositional learning with negation, i.e., achieving a performance 50% better than chance are (Hill et al., 2020; León et al., 2020). However, in those works negated instructions are interpreted as "something different from", e.g., "not p" intuitively meant "get something different from p". Instead, here we have an interpretation of negated atoms more aligned with propositional and temporal logic, and also to natural language, where "not p" intuitively means that p must be false. Hence, we believe this is the first work to show that DRL agents are capable of learning the abstract operator of negation in its classical interpretation and successfully apply it to new utterances.

B. The Symbolic Module

Here we detail the symbolic module (SM) and the main functions within it.

Symbolic Module. Formally, the first functionality of the SM is the extractor $\mathcal{E}$ (see Algorithm 1), which transforms the complex formula T into a list $\mathcal{K}$ consisting of all the possible sequences of (Markovian) atomic tasks α that satisfy T. As is common in the literature (Icarte et al., 2018; De Giacomo et al., 2018; Icarte et al., 2019; León et al., 2020; Kuo et al., 2020) we assume that the SM have access to an internal labelling function $\mathcal{L}_{\mathcal{I}} : Z \to 2^{AP}$, that acts as an *event detector* by firing when the propositions in AP hold in the environment. The second functionality is the progression function $\mathcal{P}$ (see Algorithm 2), which given the output of $\mathcal{L}_{\mathcal{I}}$ and the current $\mathcal{K}$ selects the next task α for the NM and updates $\mathcal{K}$. Once α is selected, the NM aims to follow the given instruction until it is satisfied or until the time horizon is reached, while the role of the SM is to evaluate the fulfillment of the task based on the outputs from $\mathcal{L}_{\mathcal{I}}$. This is used to generate the internal reward function R_I detailed in the main document.

C. Experiment details

In this section we provide further details to facilitate the reproduction of the experiments we performed in the main document. Code is available at `https://github.com/bgLeon/Latent-Goal-Architectures.git`.

C.1. Experiment Architecture

This is devoted to give more details about the experiment procedure. All the tested deep learning architectures share the same encoder, a 3-layer convolutional network with a kernel of 3x3 in the first and second layers and 1x1 in the third one. The number of channels in each layer are 8, 16 and 1 while the strides are 3, 3 and 1 respectively. We made use of our knowledge of the tile resolution and, consequently, the encoder is designed to generate an embedding were the instruction is independent from the observation.

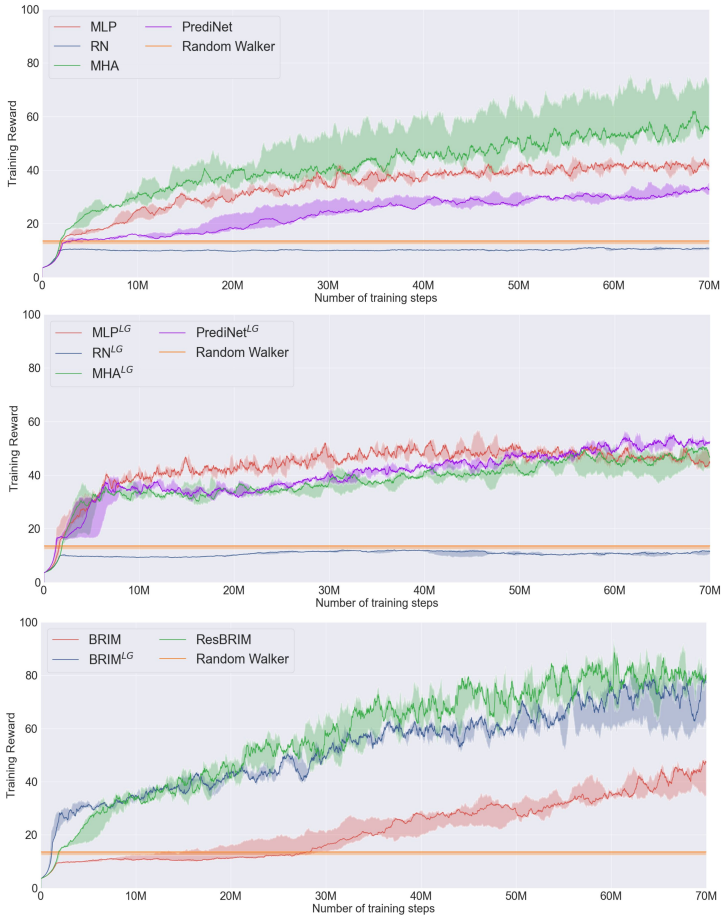

Figure 3. Training plots of the networks studied in Sec 4. Continuous lines correspond to the 50^{th} percentiles while the shadowed areas are the 25^{th} and 75^{th} percentiles. Note that good train performances do not yield to similarly good test results as detailed in the main text.

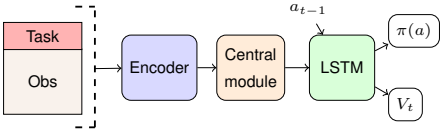

Figure 4. Standard neural network architecture. The encoder is a 3-layer CNN that receives observations with tasks as input. The central module can be either a fully connected layer (MLP), an RN, an MHA or a PrediNet. Outputs $\pi(a_t)$ and V_t refer to the actor and critic respectively (Mnih et al., 2016).

While pretraining we use only reachability goals of the form $\diamond l = \mathit{true} U l$, with objects from $\mathcal{X}_1$ (the train set). This pretraining stage helps the encoder to differentiate objects within the environment. Once pretrained, the encoder remains frozen to prevent overfitting. The training stage is 70M time steps in maps of training size and populated with items from $\mathcal{X}_2$. For testing, all the training parameters are frozen and the agents are evaluated in sets of 500 maps populated with OOD objects from $\mathcal{X}_3$. At the beginning of each episode, a task α is procedurally generated (e.g., α_1 or α_2 above). Tasks are generated with objects either from $\mathcal{X}_2$ in training or $\mathcal{X}_3$ in testing. Hence, tasks used for testing are also OOD (they combine learned SATTL operators with unseen objects). Once we selected a task, a new map is generated where the agent is placed in an aleatory position, some or all the goal objects are randomly placed and some or all the objects for the constraint are also randomly placed. Additional items from the corresponding set (either $\mathcal{X}_2$ or $\mathcal{X}_3$) are also included in the map. As stated in the main text, objects are represented by a matrix of 9x9 values. These values were procedurally generated with pseudo-aleatory numbers (we used a fixed seed) to ensure that each new object was different from the previous one.

While we do not have optimal performance metrics given the procedural nature of the benchmark, visualizing the agents we observe that their behavior is not always optimal. Still we believe that with further hyperparameter exploration and given a larger training time the agents could achieve optimal

Algorithm 1 Extractor function that obtains the the list of all the possible sequences of tasks

Function $\mathcal{E}$, Input: T
Initialize the list for the sequences of tasks $\mathcal{K}$
for each atomic task $\alpha \in T$ **do**
  if α is atomic positive or negative **then**
    for all $Seq \in \mathcal{K}$ **do**
      Seq.append(α)
    end for
  else
    // This is a non-deterministic choice
    Initialize choice list: CL
    $LK \leftarrow length(\mathcal{K})$
    for all $Seq \in \mathcal{K}$ **do**
      for all choice $\in \alpha$ **do**
        CL.append(choice)
        Generate a clone per choice $Seq' \leftarrow Seq$
        $\mathcal{K}$.append(Seq')
      end for
    end for
    Initialize counter $c \leftarrow -1$
    for i in **range**($length(\mathcal{K})$) **do**
      if $i\%\ LK == 0$ **then**
        $c+ = 1$
      end if
      We append a different choice to each sequence cloned $\mathcal{K}[i]$.append($CL[c]$)
    end for
  end if
end for
return $\mathcal{K}$

Algorithm 2 Progression function that returns the next task to be solved.

Function $\mathcal{P}$, Input:$\mathcal{K}, p_\alpha$
if $p_\alpha \neq \emptyset$ **then**
  for all $Seq \in \mathcal{K}$ **do**
    if p_α fulfills $Seq[0]$ **then**
      Seq.pop($Seq[0]$)
    else
      $\mathcal{K}$.pop(Seq)
    end if
  end for
end if
$\alpha' \leftarrow \emptyset$
if $\mathcal{K} == \emptyset$ **then**
  return α'
else
  Select the head of the first sequence: $\alpha' \leftarrow \mathcal{K}[0][0]$
  for all $Seq \in \mathcal{K}$ **do**
    // look for a non-deterministic choice
    if α' & $Seq[0]$ can be combined **then**
      $\alpha' \leftarrow \alpha' \cup Seq[0]$
    end if
  end for
end if
return α'

performance. Still, optimality in these benchmarks was not within our goals. Instead we assess the general ability of different deep learning architectures towards OOD task generalisation.

C.2. RL hyperparameters

All of our agents were trained for 70M time-steps with A2C and all using the same hyperparameters. Most of the selected hyperparameters were extracted from (León et al., 2020). Specifically, we used a discount factor $\gamma = 0.99$, a value loss weight of 0.5 and a batch size of 128. The we used an scheduled learning rate with a starting value of $8e^{-5}$, $6e^{-5}$ at the step 30 M, and $4e^{-5}$ at 55 M. These scheduled values offered the best results when testing in a range values between $1e^{-5}$ and $1e^{-3}$. The entropy-exploration term was fixed to $H = 1e^{-3}$ after testing with values in the range $[1e^{-5}, 1e^{-2}]$.

C.3. Networks

In Sec.4 we stated that all the central modules (CMs) were designed to have the same input and output size. The specific values selected were the same as in (Shanahan et al., 2020). Particularly, in the case of the MLP we used a single fully-connected layer with 640 units. For the RN we used a central hidden size of 256 units and an output size of 640 units. The MHA employs 32 head with a key size of 16 and value size of 20. Last, the PrediNet has 32 heads, with key and relation sizes of 16.

We did not perform any hyperparameter exploration with BRIMs and used directly the same configuration from the RL experiments in Mittal et al. (2020). Specifically, each BRIM layer has 6 modules with 50 units and four modules active (k) at a given time step.

C.4. Training complexity and computing time

The number of procedurally generated training tasks is ∼200000 and ∼65000 in testing. We used gridworlds where the state-space size is $\sim 1^{50}$ in training and $\sim 1^{64}$ in testing (as a guideline, chess has $\sim 1^{46}$ states). The number of procedurally generated training tasks is ∼200000 and ∼65000 in testing.

Figure 3 shows the training plots of the standard (top), latent-goal (middle) and BRIM (bottom) configurations. For train-

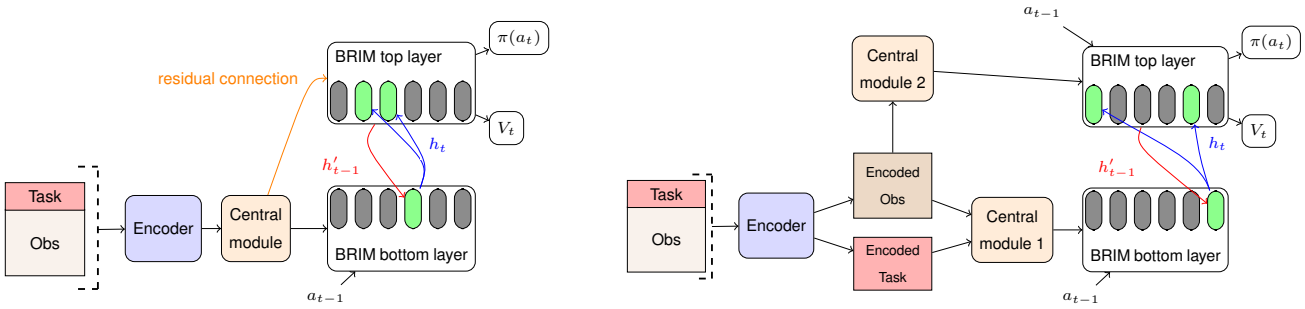

Figure 5. **Left:** vanilla BRIM and ResBRIM (without and with the residual connection respectively). Green cells within the BRIM layer refer to activated modules while gray cells are deactivated. The bottom layer receives as input the output of the central module (a PrediNet) and the hidden state from the upper layer in the previous time step (h'_{t-1}). The upper layer receives the output of the from the bottom layer (h_t). Only strong connections (the ones between activated modules) are shown. In the case of ResBRIM the upper layer also receives the output of the PrediNet. **Right:** BRIM$^{\text{LG}}$. The architecture is similar to the ResBRIM but the top layer receives an input an embedding from the observation only. Consequently, the top layer has only access to the current goal through the lower BRIM layer.

ing we used various computing clusters with GPUs such as Nvidia Tesla K80, Tesla T4 or TITAN Xp. We employed Intel Xeon CPUs and consumed 5 GB of RAM per every three independent runs working in parallel. Running concurrently, each experiment typically takes 4 days to train, 5 days in the case of multi-module configurations.

D. SATTL and LTL$_f$

We finish by giving formal details about SATTL. Given a trace λ, i.e., a sequence of states and actions, we denote time steps, i.e., instants, on the trace as $\lambda[j]$, for $0 \leq j < |\lambda|$, where $|\lambda|$ is the length of the trace. $\lambda[i, j]$ is the (sub)trace between instants i and j. 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 sets of atoms in AP.

Definition 2 (Satisfaction)**.** *Let $\mathcal{N}$ be a model and λ a finite path. We define the satisfaction relation $\models$ for literals l, atomic tasks α, and temporal formulas T in path λ as follows:*

$$
\begin{array}{ll}
(\mathcal{N}, \lambda) \models +p & \textit{iff } p \in L(\lambda[0]). \\
(\mathcal{N}, \lambda) \models -p & \textit{iff } p \notin L(\lambda[0]). \\
(\mathcal{N}, \lambda) \models l \vee l' & \textit{iff } (\mathcal{N}, \lambda) \models l \textit{ or } (\mathcal{N}, \lambda) \models l. \\
(\mathcal{N}, \lambda) \models lUl' & \textit{iff for some } 0 \leq j < |\lambda|, (\mathcal{N}, \lambda[j, |\lambda|]) \\
 & \quad \models l', \textit{and for every } t \in [0, j), \\
 & \quad (\mathcal{N}, \lambda[t, j-1]) \models l. \\
(\mathcal{N}, \lambda) \models T; T' & \textit{iff for some } 0 \leq j < |\lambda|, (\mathcal{N}, \lambda[0, j]) \\
 & \quad \models T \textit{ and } (\mathcal{N}, \lambda[j+1, |\lambda|]) \models T'. \\
(\mathcal{N}, \lambda) \models T \cup T' & \textit{iff } (\mathcal{N}, \lambda) \models T \textit{ or } (\mathcal{N}, \lambda) \models T'.
\end{array}
$$

Intuitively, by Def. 2 an atomic task $\alpha = c_\alpha U g_\alpha$ is satisfied if the "safety" condition c_α remains true until g_α is fulfilled. In the context of this work we are not interested in strict safety warranties but in agents that are trying to reach a goal while trying to minimize the violation of additional conditions. Thus, we may say our agent has satisfied α while it has not fulfilled c_α at every time step before g_α. Formally, in those cases we are considering traces that are not starting at the beginning of the episode. Still, the agent is penalised for this behaviour through its reward function (Sec. 3)

We now demonstrate that SATTL is a fragment of the widely-used LTL$_f$. The syntax of LTL$_f$ is defined as follows:

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathrm{U} \varphi_2$$

Since both LTL$_f$ and SATTL are defined ofer finite traces, we can directly introduce the following translations:

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathrm{U} \varphi_2$$

Definition 3. *Translations τ from Task Temporal Logic to LTL$_f$ are defined as follows:*

$$
\begin{array}{lcl}
\tau(\alpha) & = & lUl' \\
\tau(T \cup T') & = & \tau(T) \vee \tau(T') \\
\tau(T; T') & = & \begin{cases} lU(l' \wedge \tau(T')) & \textit{if } T = lUl' \\ \tau(T_1; (T_2; T')) & \textit{if } T = T_1; T_2 \\ \tau((T_1; T') \cup (T_2; T')) & \textit{if } T = T_1 \cup T_2 \end{cases}
\end{array}
$$

We immediately prove that translation τ preserve the interpretation of formulae in SATTL.

Proposition 1. *Given a model $\mathcal{N}$ and trace λ, for every formula T in SATTL,*

$$(\mathcal{N}, \lambda) \models T \quad \textit{iff} \quad (\mathcal{N}, \lambda) \models \tau(T)$$

Proof. The proof is by induction on the structure of formula T. The base case follows immediately by the semantics of SATTL and LTL$_f$.

As for the induction step, the case of interest is for formulae of type $T; T'$. In particular, $(\mathcal{N}, \lambda) \models T; T'$ iff for some $0 \leq j < |\lambda|$, $(\mathcal{N}, \lambda[0, j]) \models T$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models T'$. By induction hypothesis, this is equivalent to $(\mathcal{N}, \lambda[0, j]) \models lUl'$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models \tau(T')$ in the case of $T = lUl'$. Finally, this is equivalent to $(\mathcal{N}, \lambda) \models lU(l' \wedge \tau(T'))$. The cases for $T = T_1; T_2$ and $T = T_1 \cup T_2$ are dealt with similarly.

Finally, the case for $T \cup T'$ follows by induction hypothesis and the distributivity of $\vee$. $\square$