Grounding LTLf specifications in images

Elena Umili[1], Roberto Capobianco[1,2] and Giuseppe DeGiacomo[1]

[1]Sapienza University of Rome
[2]Sony AI

Abstract

A critical challenge in neurosymbolic approaches is to handle the symbol grounding problem without direct supervision. That is mapping high-dimensional raw data into an interpretation over a finite set of abstract concepts with a known meaning, without using labels. In this work, we ground symbols into sequences of images by exploiting symbolic logical knowledge in the form of Linear Temporal Logic over finite traces (LTLf) formulas, and sequence-level labels expressing if a sequence of images is compliant or not with the given formula. Our approach is based on translating the LTLf formula into an equivalent deterministic finite automaton (DFA) and interpreting the latter in fuzzy logic. Experiments show that our system outperforms recurrent neural networks in sequence classification and can reach high image classification accuracy without being trained with any single-image label.

1. Introduction

A crucial problem in neurosymbolic integration is handling the symbol grounding problem without direct supervision. We refer to symbol grounding [1] as the process of mapping raw data into an interpretation over a finite boolean symbolic alphabet, where each symbol expresses a meaningful high-level concept. In particular, we focus on grounding symbols in raw data sequences using some prior symbolic knowledge expressed in Linear Temporal Logic interpreted on finite traces (LTLf) [2]. LTLf is used in a big variety of domains, from robotics [3] to Business Process Management (BPM) [4], for specifying temporal relationships, dynamic constraints and performing automated reasoning. It is unambiguous compared to natural language, yet easy to use and understand. Evaluating if a symbolic sequence is compliant with a given LTLf formula is straightforward. In several real-world applications, however, such sequences are not symbolic but appear 'rendered', or grounded in raw data such as images, videos, words, audio, etc. In some application domains, such for example in BPM [5], we could know a high-level specification of the process expressed in terms of symbols, yet exploiting this knowledge is impossible unless it is grounded in the data. Therefore, symbol grounding represents the first preliminary step to be made to perform any logical reasoning, included evaluation.

Deep neural networks perform extraordinary well in perception tasks on raw data [6]. Supervised classification can be seen as grounding a set of classes in the dataset, by training directly on a set of (data, class) examples. Despite the success of deep learning in this area,

NeSy 2022, 16th International Workshop on Neural-Symbolic Learning and Reasoning, Cumberland Lodge, Windsor, UK
umili@diag.uniroma1.it (E. Umili); capobianco@diag.uniroma1.it (R. Capobianco); degiacomo@diag.uniroma1.it (G. DeGiacomo)

the main drawback remains the acquisition of the labeled data necessary for training. State-of-the-art self-supervised approaches have seen enormous progress lately; they can compress high-dimensional data and cluster it in a meaningful way [7] [8] without using any label. In particular, some approaches can also extract a discrete representation of the input data [9], which can be considered an interpretation over a symbolic alphabet [10] [11] [12]. However, we do not know the meaning of these automatically-extracted symbols, and inspecting them or connecting them to some human-designed knowledge remains extremely hard.

In this work, we take a step in the direction of grounding a known meaningful set of symbols in perceptual data, with as little supervision as possible, by exploiting some prior knowledge about expected sequencing expressed as an LTLf fomula in the same symbolic alphabet. Our framework is based on translating the LTLf symbolic knowledge into an equivalent deterministic finite automaton (DFA) and encoding the latter using fuzzy logic. The use of fuzzy logic has seen many successes in neurosymboilc AI [13], and many framework are based on it, such as Logic Tensor Networks (LTN) [14] and Lyrics [15]. Unlike prior work, we focus on grounding knowledge into time-extended data sequences. Our work is similar to LTN, but extends it to temporal logic and time-extended data. LTN extends First Order Logic (FOL) to make it compatible with machine-learning tasks. For example introducing the concept of a *dataset* containing more data samples, and the concept of *feature*. However, the concept of *time* is still missing, in the sense that encoding knowledge on a set of examples (batch dimension), each represented by a sequence of data (time dimension), eventually multidimensional (feature dimension), is not straightforward. In our work, we manage the time dimension by applying recursion over different time steps, in the same way recurrent neural networks do.

In summary, the main contribution of this paper is a framework able to encode temporally extended specifications and ground them on sequences of images of any length, through a recursive structure. Experiments show that our method effectively classifies both sequences and single images. In particular, it is faster, requires less data, and is more robust to overfitting than a classical end-to-end classical neural approach that cannot use high-level knowledge.

The remainder of this paper is organized as follows: in section 2 we report related works; in section 3 we give some preliminaries on Linear Temporal Logic and Logic Tensor Networks; in section 4 we formulate our problem and illustrate in detail the method used to solve it; we report the experiments evaluating our approach in section 5; and finally we conclude and discuss directions for future work in section 6.

2. Related works

Integrating logical knowledge and neural networks Integrating logical knowledge and deep learning is still an open problem, and many different approaches have been proposed. Some works propose embedding logical knowledge and symbolic data in the same feature space and inferring connections between the two using the distance in the feature space as a metric [16] [17]. In this case, the representation quality depends on the training, and obtaining the same exact behavior of the logical knowledge can be hard. Some other approaches use real-valued logic [14] [18], such as fuzzy logic or probabilistic logic, to integrate sub-symbolic perception and symbolic reasoning. The use of real-valued logic is compatible with gradient

descent optimization that is at the base of neural network training. In this work, we use this second approach and in particular we focus on the use of LTLf knowledge.

Machine learning and LTL Many works exploit the synergies between machine learning and LTL in a beneficial way. In reinforcement learning, LTL-based reward machines are used to simplify and automate the creation of reward functions for Markovian and non-Markovian decision processes [19][20]. However, they are applicable only in discrete-state environments or continuous problems for which a mapping between the continuous state and a symbolic interpretation is known, also known as labeled MDP [21]. Some works use neural networks to solve problems related to LTL, generally approached with combinatorial algorithms. Camacho and McIlraith [22] use deep learning to guide research in program synthesis and improve scalability. Walke et al. [23] use recurrent neural networks to learn LTLf formulas from a set of traces. However, these works use symbolic data and do not consider the problem of discovering latent symbols in the data, which is the problem we face in our work.

Exploiting high-level knowledge for vision tasks Previous works have shown that vision tasks can benefit from background knowledge. Stewart and Ermon [24] perform detecting and tracking objects, without any labels, by exploiting known laws of physics. Donadello et al. [25] exploit logical knowledge to increase robustness to noisy datasets with incorrect labels in semantic image interpretation tasks. In particular, our work focuses on classifying images using logical symbolic knowledge instead of image-class labels in a semi-supervised fashion.

Semisupervised symbol grounding A benchmark for semisupervised symbol grounding is the digit addition problem, where a system must learn to classify MNIST digits images by knowing only the result of their sum and how addition works. LTN [14] and DeepProbLog [18] show how their systems can benefit from knowing addition rules. However, they handle the problem only in two settings: single-digit and double-digit addition. Dai et al. [26] use logic abduction to correct the prediction of a CNN, by using a derivative-free optimization. They tested their framework on binary sums of digits, where the two binary numbers can have various lengths. We propose a similar experiment on MNIST digits, that does not concern addition and where we do not know in advance the input sequence length. In particular, we evaluate an LTLf formula over sequences of arbitrary lengths of digits by using a recurrent specification in the form of a fuzzy DFA. We use the same approach of LTN, by adapting it to LTLf formulas. To the best of our knowledge, it's the first time LTN has been used to incorporate temporal logic knowledge into neural networks.

3. Background

3.1. LTLf and DFA

Linear Temporal Logic (LTL) [27] is a language which extends traditional propositional logic with modal operators. With the latter we can specify rules that must hold *through time*. In this work, we use LTL interpreted over finite traces (LTLf) [2]. Such interpretation allows the

executions of arbitrarily long traces, but not infinite, and is adequate for finite-horizon planning problems.

Given a set P of propositions, the syntax for constructing an LTLf formula ϕ is given by

$$\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \phi_1 U \phi_2 \tag{1}$$

where $p \in P$. We use $\top$ and $\bot$ to denote true and false respectively. X (Next) and U (Until) are temporal operators. Other temporal operators are: N (Weak Next) and R (Release) respectively, defined as $N\phi \equiv \neg X \neg \phi$ and $\phi_1 R \phi_2 \equiv \neg(\neg\phi_1 U \neg\phi_2)$; G (globally) $G\phi \equiv \bot R \phi$ and F (eventually) $F\phi \equiv TU\phi$. A trace $\rho = \rho[0], \rho[1], ...$ is a sequence of propositional assignments, where $\rho[x] \in 2^P$ ($x \geq 0$) is the x-th point of ρ. Intuitively, $\rho[x]$ is the set of propositions that are true at instant x. Additionally, $|\rho|$ represents the length of ρ. Since each trace is finite $|\rho| < \infty$ and $\rho \in (2^P)^*$. We refer the reader to [27] for a formal description of the operators' semantics. Any LTLf formula ϕ can be translated in an equivalent Deterministic Finite Automaton (DFA) $A_\phi = (2^P, S, s_0, \delta, F)$, where 2^P is the automaton alphabet, S is the set of states, $s_0 \in S$ is the initial state, $\delta : Sx2^P \rightarrow S$ is the transition function and $F \subseteq S$ is the set of final states. Let be $L(A_\phi)$ the language composed by all the strings accepted by the A_ϕ we have

$$\rho \vDash \phi \text{ iff } \rho \in L(A_\phi) \tag{2}$$

Despite the size of A_ϕ is double-exponential in ϕ in the worst-case [2], A_ϕ is often quite small in practice, and scalable techniques are available for computing it from ϕ [28] [29] [30].

LTLf formulas are widely used in BPM. In particular, the BPM community has selected 21 types of formulas that are particularly significant for describing complex processes declaratively [31]. The latter are at the base of the system Declare [32] and they generate DFAs that are polynomial in the original formula [33]. We use the Declare formulas as a benchmark for evaluating our approach.

3.2. Logic Tensor Networks

Logic Tensor Networks (LTN) [14] are a neurosymbolic framework that can reason and learn by exploiting both structured symbolic knowledge and raw data. It implements a logic called Real Logic, which contains constants, function and predicate symbols, as First Order Logic (FOL). LTN also implements connectives ($\neg, \wedge, \vee, \rightarrow, l \leftrightarrow$) and quantifiers (universal, existential, diagonal universal, and guarded universal and existential). Any logic formula in Real Logic is interpreted using fuzzy logic semantics, namely, it is assigned with a continuous truth-value between 0 and 1. Fuzzy logic has shown to be suitable in several real-world applications where a statement can be only partially true or exceptions can be present. Notably, fuzzy interpretations are based on continuous and differentiable functions, so neural networks can co-exist in the framework and actually implement elements of the logic. Every element of Real Logic is grounded in real tensor, so that it can be an assignment to available data, the output of a neural network, or a satisfaction level of a logic formula between 0 and 1.

LTN can be used for querying, reasoning and learning: here we focus on learning. LTN can learn from both data and symbolic knowledge by imposing the knowledge available, and searching for the groundings that maximize the satisfiability of that knowledge. This is done

by simply defining a loss objective that is inverse to the given formula's satisfaction level and optimizing the system's trainable weights by back-propagation. In our work, we use the same concept of *learning by best satisfiability*, but we apply it to the DFA generated by the LTLf formula. The neural computational graph implementing the automaton has therefore a *recurrent* structure, like a Long short-term memory (LSTM) neural network, and can be applied to sequences of any length. This feature is missing in the current implementation of LTN, and it is very convenient for imposing logic specifications that are extended in the time dimension.

4. Method

In this section, we formulate our problem in detail, and we present the method used to encode the LTLf knowledge and ground the alphabet in the data.

4.1. Problem formulation

We consider the problem of classifying a sequence of images $x = i[0], i[1], ..i[l]$ as compliant or not with a certain specification expressed as an LTLf formula ϕ. Each image is the 'rendering' of a symbolic interpretation over the formula alphabet P. This means that there exists a function $c : I \rightarrow 2^P$, where I is the space of images, that maps each image into the truth values of symbols in P. If we map each image in the symbolic space with this function we obtain a trace $p = p[0], p[1], ..., p[l-1]$, where l is the sequence length and $p[i] = c(i[t])\ \forall 0 \leq t \leq l$. We denote with $A_\phi = (2^P, S, s_0, \delta, F)$ the DFA corresponding to the formula ϕ, where 2^P is the automaton alphabet, S is the set of states, s_0 is the initial state, δ is the transition function and F is the set of final states. If we run the trace in the DFA we obtain a sequence of $l+1$ automaton states $s = s[0], s[1], ...s[l]$, where $s[0] = s0$ is the initial state and the last state $s[l] \in F$ if the sequence of images is accepted or $s[l] \in (S - F)$ otherwise.

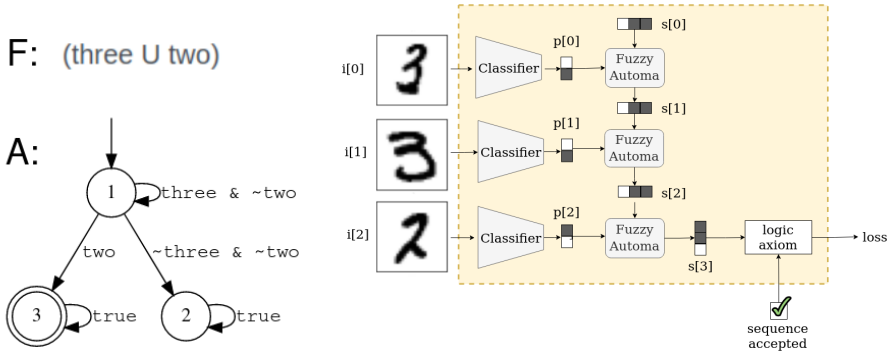
We are interested in the classifying function c. We assume that we can discover it in a weakly supervised way, namely without using any single-image label (image, symbolic interpretation). In particular, we assume to know the following information: (i) the formula ϕ, from which we can build the DFA A_ϕ, (ii) a set of training data $D = \{< x_1, y_1 >, < x_2, y_2 >, ..., < x_n, y_n >\}$ where x_k is an image sequence $i[0], i[1], ..i[l-1]$ and $y_k \in \{0, 1\}$ is the label denoting whether the sequence is accepted or not.

4.2. Framework

We consider our framework as a neural network composed of two parts:(i) a *perception part*, represented by a trainable convolutional neural network that classifies symbols from images, implementing the function c we want to discover; (ii) *a logic part*, represented by a non-trainable recurrent structure, that is a fuzzy correspondent of the automaton A_ϕ. Figure 1 shows an example of the functioning of our framework.

The sequence of images $x = i[0], i[1], ..., i[l-1]$ is passed one by one to the classifier, producing l continuous vectors of dimension $|P|$ where P is the set of propositions used by the formula.

We define a fuzzy predicate $P_c(c_i, t)$ denoting whether the t-th image in the sequence belongs to class i. The classifier implements the grounding of P_c. In fact the component i of the



(a) (b)

Figure 1: a) An example of LTLf formula with the corresponding equivalent automaton, b) our framework

CNN prediction for the image $i[t]$ in the sequence is the truth value of $P_c(c_i, t)$. We denote as $p[t] = [P_c(c_0, t), P_c(c_1, t), ..P_c(c_{|P|}, t)]$ the fuzzy interpretation over propositions in P at time t. This fuzzy interpretation can be used to proceed on the automaton.

In particular, at any time t we are in a state of the automaton, we encode this information with another fuzzy predicate P_s, where $P_s(s_i, t)$ is true if we are in state s_i at time t. As before, we define $s[t]$ the interpretation at time t over the state symbols $s[t] = [P_s(s_0, t), P_s(s_1, t), ..., P_s(s_{|S|}, t)$, with $|S|$ equal to the number of states in the DFA.

If at time t we are in a state s_i of the DFA and we receive a certain interpretation $p[t]$ over the set of symbols, at time $t+1$ we transit to the state s_j linked to s_i by the edge $e_{i,j}$ that is made true by the interpretation. For example, if we are in state 1 of the DFA in Figure 1(a),and we receive the interpretation $['three' = False, 'two' = False]$ we move to state 2 because the interpretation satisfies the formula $\neg three \wedge \neg two$ on the arc $e_{1,2}$. More formally $P_s(s_j, t+1) = (P_s(s_i, t) \wedge e_{i,j}(p[t]))$, where we denote as $e_{i,j}(p[t])$ the truth value of the formula on arc $e_{i,j}$ when evaluated on the interpretation $p[t]$.

We start at the initial state s_0 of the automaton, and we have therefore

$$P_s(s_0, 0) = \top \wedge (P_s(s_i, 0) = \bot \; \forall 1 \leq i \leq |S|) \tag{3}$$

Then we simulate a run of the automaton using the fuzzy symbolic interpretations the classifier has predicted. The transition from a state s_i to a state s_j through an interpretation $p[t]$ follows the rule:

$$P_s(s_j, t+1) = \bigcup_{i:(i,j) \text{ is an edge of } A_\phi} P_s(s_i, t) \wedge e_{i,j}(p[t]) \tag{4}$$

Finally we evaluate the last interpretation over the state symbols $s[l]$ and we impose this must be either: one state in F if the sequence is accepted; or one state not in F if the sequence is negative. For this purpose we define the predicate $Accepted(x)$ that is $\top$ if the label y associated with x in the dataset is 1, and $\bot$ if the label is 0. We know that:

$$\forall x Accepted(x) \leftrightarrow \bigcup_{s_i \in F} P_s(s_i, l) \tag{5}$$

We optimize the network so as to maximize the satisfiability of this formula. In fact the truth value of formula 5 depends on the truth value of the last state, that in turn depends on the previous state and the previous classes and so on. Let $s(x_i, y_i)$ be the truth value of Equation 5 for one particular sample (x_i, y_i) in the dataset. The loss associated with that sample is $1 - s(x_i)$ that is aggregated over all data, since the formula must be true $\forall x$.

$$L = \sum_{i=0}^{i=n} (1 - s(x_i, y_i)) \tag{6}$$

The loss L is backpropagated through the network and the classifier weights are updated with classical gradient descent. In particular Equations 3, 4 and 5 are the axioms in our knowledge base. They are all evaluated in fuzzy logic using the product t-norm T_P for conjunction, its dual t-conorm S_P for disjunction, standard negation N_S , and the Reichenbach implication I_R, as suggested in the LTN paper [14].

$$\begin{aligned} \neg &: N_S(a) = 1 - a \\ \wedge &: T_P(a, b) = a * b \\ \vee &: S_P(a, b) = a + b - a * b \\ \rightarrow &: I_R(a, b) = 1 - a + a * b \end{aligned}$$

In figure 1(b) we show the network behaviour in case of perfect grounding. In this case the classifier predicts all one-hot encodings, this represent an ideal situation where no uncertainty is present, and symbols are all either perfectly true or perfectly false. Also the output from the fuzzy trainsitions is perfectly boolean and the fuzzy automaton behaves exactly as the original DFA. However, the fuzzy automata can predict the sequence of states even with some uncertainty in the symbol grounding layer, while the original DFA cannot handle any uncertainty.

5. Experiments

In this section we report the experiments supporting our method. The implementation code is available online at https://github.com/whitemech/grounding_LTLf_in_image_sequences.

Since LTL can be used to specify innumerable constraints, we test our framework on a subset of formulas that is as complete as possible and, at the same time, useful for practical applications. We choose, therefore, to test it on the Declare constraints. Declare [32] is one of the prime languages of the declarative process modeling paradigm, and is composed of 20 types of activity constraints expressed as LTLf formulas. See the appendix for a complete list of Declare formulas. Declare formulas assume that one and only one proposition is $\top$ at each instant of time, that is symbols are *mutually exclusive.* For each Declare formula, we perform an LTLf evaluation experiment in three settings: (1) training on the complete dataset; (2) training on a restricted dataset; (3) training on the complete dataset by dropping the Declare assumption on mutually exclusive symbols (see the following section for more details about the dataset creation process). We report the sequence classification accuracy, that is the ratio of correctly evaluated sequences, and the image classification accuracy, namely the ratio of correctly predicted symbolic interpretation in single images.

5.1. Dataset

The dataset is created by rendering symbolic configurations using images of zeros and ones from the MNIST dataset. In these experiments, therefore, we used an alphabet composed of only two symbols. However, we can apply the framework to an alphabet of any size by changing the classifier output layer. For each formula, all the possible symbolic traces with length between 1 and 4 are created. The latter are randomly split in train traces and test traces, we denote as $p_{traces,train}$ the percentage of traces used for training. In the same way images in the MNIST dataset are randomly divided in train and test images, we denote as $p_{images,train}$ the percentage of images used for training.

We construct the training dataset by rendering train traces with train images and the test dataset by rendering test traces with test images. In this way, the test contains symbolic traces never observed in the training, in which each symbolic interpretation is rendered with an image never observed during training.

We test our approach on three dataset: (i) complete, (ii) restricted, (iii) complete with non mutually exclusive symbols. The complete dataset is built as described above with parameters $p_{traces,train}$ = 50% and $p_{images,train}$ =85%. The restricted dataset is constructed by using parameters $p_{traces,train}$=40% and $p_{images,train}$ = 15%. Achieving good perception performances on the restricted dataset is therefore more difficult, since a big percentage of possible renderings are not observed during training.

Images from MNIST dataset render only one digit at time (mutually exclusive symbols), however our framework can be tested also for multilabel classification, as needed when symbols are not mutually exclusive. For this purpose we create also a dataset rendering interpretations non in MNIST: when all symbols are set to false (rendered as a black image), when both symbols set to true (rendered as a 'zero' image and a 'one' image superimposed on each other). We create a dataset for multilabel classification by modifying MNIST images as described above and using the same parameters values used for the complete dataset, namely $p_{traces,train}$ = 50% and $p_{images,train}$ =85%.

5.2. Results

We compare our neurosymbolic approach (NS) with a classical supervised deep learning approach (DL). We implement the latter with a convolutional neural network (the same used by NS) followed by an LSTM. For each approach, each formula, and each dataset, we perform 10 experiments with different seeds, and we keep the best 8 ones. For space reasons, we report the results obtained with each single Declare formula in the appendix and the mean results over the 20 different Declare formulas in Figure 2. In all the plots solid line is the mean, and the shaded area represents the standard deviation. In the sequence classification task, Figure 2 (first row), our approach outperforms the deep learning approach in all three datasets, even in the non-mutually exclusive symbol case, although Declare formulas are not designed for this kind of interpretation. The lstm-based approach struggles to reach the top accuracy on the test set, and this is even more evident in the experiment on the restricted image dataset. It also happens because in some formulas the lstm tends to overfit the training data, which is visible in the results in the appendix.

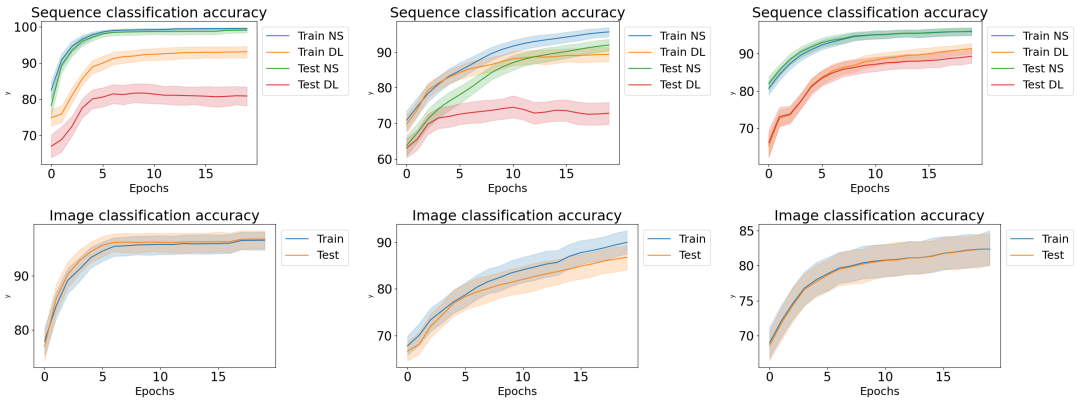


Figure 2: Experiments over 20 Declare formulas. In the first row: **sequence classification** accuracy, in the second row: **image classification** accuracy. They are obtained by training on three different datasets: (first columns) **complete dataset**, (second column) **restricted dataset**, (third column) complete dataset with **non-mutually exclusive symbols**. Solid lines represent mean values, shaded areas represent standard deviations.

In the image classification task, Figure 2 (second row), our approach reaches high accuracy on both the test and training sets without exploiting any image label.

5.3. Discussion on 'groundability'

Our system does not need any single-data label to ground the symbols of a given formula into data, however, correctly grounding single data using only sequence labels and the formula is not always possible for any arbitrary formula. In particular, if there exists a redenomination of the symbols in the alphabet that maintains all the accepted traces still accepted and all the unaccepted traces still not accepted, multiple groundings are possible. When trained on these formulas, our system can still distinguish one class from the other, but can choose the wrong names for symbols. For example, in our experiment on MNIST digits, the classifier may assign all images of zeros the label 1 and all the images of ones the label 0. In this case, we observe that the accuracy on single image classification approaches 0% while the sequence classification accuracy still approaches 100%. In order to aggregate results from these formulas, which can do either 100% or 0%, we do not plot the value x of image classification accuracy, but the distance from 50%, that is (50 + |x - 50|)% in Figure 2 (second row). This value goes to 100%, which means the system correctly clusters all images of zeros together and all images of ones together.

This happens for all the formulas except one: choice(c0,c1), that is $Fc0 \vee Fc1$ LTL (see the appendix). This formula accepts any trace of mutually exclusive symbols, it is therefore not specific enough, even to cluster images in the correct way. In fact, this formula achieves the highest sequence classification accuracy and the lowest image classification accuracy. We would obtain the same results with a tautology or an unsatisfiable formula.

Let us notice that this is not a problem with our implementation or the specific experiment we made, but it is a problem with the process of abduction in general. However, our system is compatible with the use of single image labels, which can be employed to ensure the assignment

of the correct class names to the clusters in case the LTLf specification is not informative enough to infer them.

6. Conclusion and future work

In conclusion, we propose a framework for exploiting high-level logical knowledge in the form of LTLf formulas. In particular, we use this knowledge to map images into a set of symbols with a known meaning without any image label. We have shown that discovering this mapping is possible by using only sequence-level labels and the logical knowledge. Furthermore, in sequence classification, our approach outperforms the end-to-end approach based on recurrent neural networks: it is more general and can maintain high performances using fewer labels.

In the future, we want to apply this framework to a more realistic scenario in the area of BPM or natural language processing, and we want to investigate how it can perform in the case of nonperfect symbolic knowledge.

Acknowledgments

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), by the PRIN project RIPER (No. 20203FFYLK)".

References

[1] L. L. Steels, The symbol grounding problem has been solved, so what's next?, 2008.

[2] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI '13, AAAI Press, 2013, p. 854–860.

[3] K. He, A. M. Wells, L. E. Kavraki, M. Y. Vardi, Efficient symbolic reactive synthesis for finite-horizon tasks, in: 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 8993–8999. doi:10.1109/ICRA.2019.8794170.

[4] G. D. Giacomo, R. D. Masellis, M. Grasso, F. M. Maggi, M. Montali, Monitoring business metaconstraints based on ltl and ldl for finite traces, in: BPM, 2014.

[5] W. Kratsch, F. König, M. Röglinger, Shedding light on blind spots – developing a reference architecture to leverage video data for process mining, Decision Support Systems 158 (2022) 113794. URL: https://www.sciencedirect.com/science/article/pii/S0167923622000653. doi:https://doi.org/10.1016/j.dss.2022.113794.

[6] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.

[7] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. Ávila Pires, Z. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, M. Valko, Bootstrap your own latent - a new approach to self-supervised learning, in: NeurIPS, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/f3ada80d5c4ee70142b17b8192b2958e-Abstract.html.

[8] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, A. Joulin, Emerging properties in self-supervised vision transformers, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 9650–9660.

[9] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017. URL: https://openreview.net/forum?id=rkE3y85ee.

[10] M. Asai, A. Fukunaga, Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary, Proceedings of the AAAI Conference on Artificial Intelligence 32 (2018). URL: https://ojs.aaai.org/index.php/AAAI/article/view/12077.

[11] A. Dittadi, F. K. Drachmann, T. Bolander, Planning from pixels in atari with learned symbolic representations, in: AAAI, 2021.

[12] E. Umili, E. Antonioni, F. Riccio, R. Capobianco, D. Nardi, G. De Giacomo, Learning a symbolic planning domain through the interaction with continuous environments, in: Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL), 2021.

[13] E. van Krieken, E. Acar, F. van Harmelen, Analyzing Differentiable Fuzzy Implications, in: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, 2020, pp. 893–903. URL: https://doi.org/10.24963/kr.2020/92. doi:10.24963/kr.2020/92.

[14] S. Badreddine, A. d'Avila Garcez, L. Serafini, M. Spranger, Logic tensor networks, Artificial Intelligence 303 (2022) 103649. URL: https://www.sciencedirect.com/science/article/pii/S0004370221002009. doi:https://doi.org/10.1016/j.artint.2021.103649.

[15] G. Marra, F. Giannini, M. Diligenti, M. Gori, Lyrics: A general interface layer to integrate logic inference and deep learning, in: ECML/PKDD, 2019.

[16] Y. Xie, F. Zhou, H. Soh, Embedding symbolic temporal knowledge into deep sequential models, 2021. URL: https://arxiv.org/abs/2101.11981. doi:10.48550/ARXIV.2101.11981.

[17] Y. Xie, Z. Xu, M. S. Kankanhalli, K. S. Meel, H. Soh, Embedding symbolic knowledge into deep networks, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/7b66b4fd401a271a1c7224027ce111bc-Paper.pdf.

[18] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. De Raedt, Deepproblog: Neural probabilistic logic programming, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 31, Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.

[19] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. Valenzano, S. A. McIlraith, Ltl and beyond: Formal languages for reward function specification in reinforcement learning, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 6065–6073. URL: https://doi.org/10.24963/ijcai.2019/840. doi:10.24963/ijcai.2019/840.

[20] G. De Giacomo, L. Iocchi, M. Favorito, F. Patrizi, Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications, Proceedings of the Inter-

national Conference on Automated Planning and Scheduling 29 (2021) 128–136. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3549.

[21] C. Wang, Y. Li, S. L. Smith, J. Liu, Continuous motion planning with temporal logic specifications using deep neural networks, 2020. URL: https://arxiv.org/abs/2004.02610. doi:10.48550/ARXIV.2004.02610.

[22] A. Camacho, S. A. McIlraith, Towards neural-guided program synthesis of Linear Temporal Logic specifications, in: Workshop on Knowledge Representation and Reasoning Meets Machine Learning (KR2ML) at NeurIPS, 2019.

[23] H. Walke, D. Ritter, C. Trimbach, M. Littman, Learning finite linear temporal logic specifications with a specialized neural operator, 2021. URL: https://arxiv.org/abs/2111.04147. doi:10.48550/ARXIV.2111.04147.

[24] R. Stewart, S. Ermon, Label-free supervision of neural networks with physics and domain knowledge, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[25] I. Donadello, L. Serafini, A. d'Avila Garcez, Logic tensor networks for semantic image interpretation, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 1596–1602. URL: https://doi.org/10.24963/ijcai.2017/221. doi:10.24963/ijcai.2017/221.

[26] W.-Z. Dai, Q. Xu, Y. Yu, Z.-H. Zhou, Bridging machine learning and logical reasoning by abductive learning, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/9c19a2aa1d84e04b0bd4bc888792bd1e-Paper.pdf.

[27] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, IEEE Computer Society, 1977, pp. 46–57. URL: https://doi.org/10.1109/SFCS.1977.32. doi:10.1109/SFCS.1977.32.

[28] S. Zhu, L. M. Tabajara, J. Li, G. Pu, M. Y. Vardi, Symbolic ltlf synthesis, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 1362–1369. URL: https://doi.org/10.24963/ijcai.2017/189. doi:10.24963/ijcai.2017/189.

[29] S. Bansal, Y. Li, L. M. Tabajara, M. Y. Vardi, Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 9766–9774. URL: https://aaai.org/ojs/index.php/AAAI/article/view/6528.

[30] G. D. Giacomo, M. Favorito, Compositional approach to translate ltlf/ldlf into deterministic finite automata, in: S. Biundo, M. Do, R. Goldman, M. Katz, Q. Yang, H. H. Zhuo (Eds.), Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021, AAAI Press, 2021, pp. 122–130. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/15954.

[31] M. Pesic, W. M. van der Aalst, A declarative approach for flexible business processes management, in: Business Process Management Workshops, 2006.

[32] M. Pesic, H. Schonenberg, W. M. van der Aalst, Declare: Full support for loosely-structured processes, in: 11th IEEE International Enterprise Distributed Object Computing Conference

(EDOC 2007), 2007, pp. 287–287. doi:10.1109/EDOC.2007.14.
[33] M. Westergaard, Better algorithms for analyzing and enacting declarative workflow languages using ltl, in: S. Rinderle-Ma, F. Toumani, K. Wolf (Eds.), Business Process Management, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 83–98.
[34] J. De Smedt, S. vanden Broucke, J. Weerdt, J. Vanthienen, A full r/i-net construct lexicon for declare constraints, 2015. doi:10.2139/ssrn.2572869.

A. Appendix

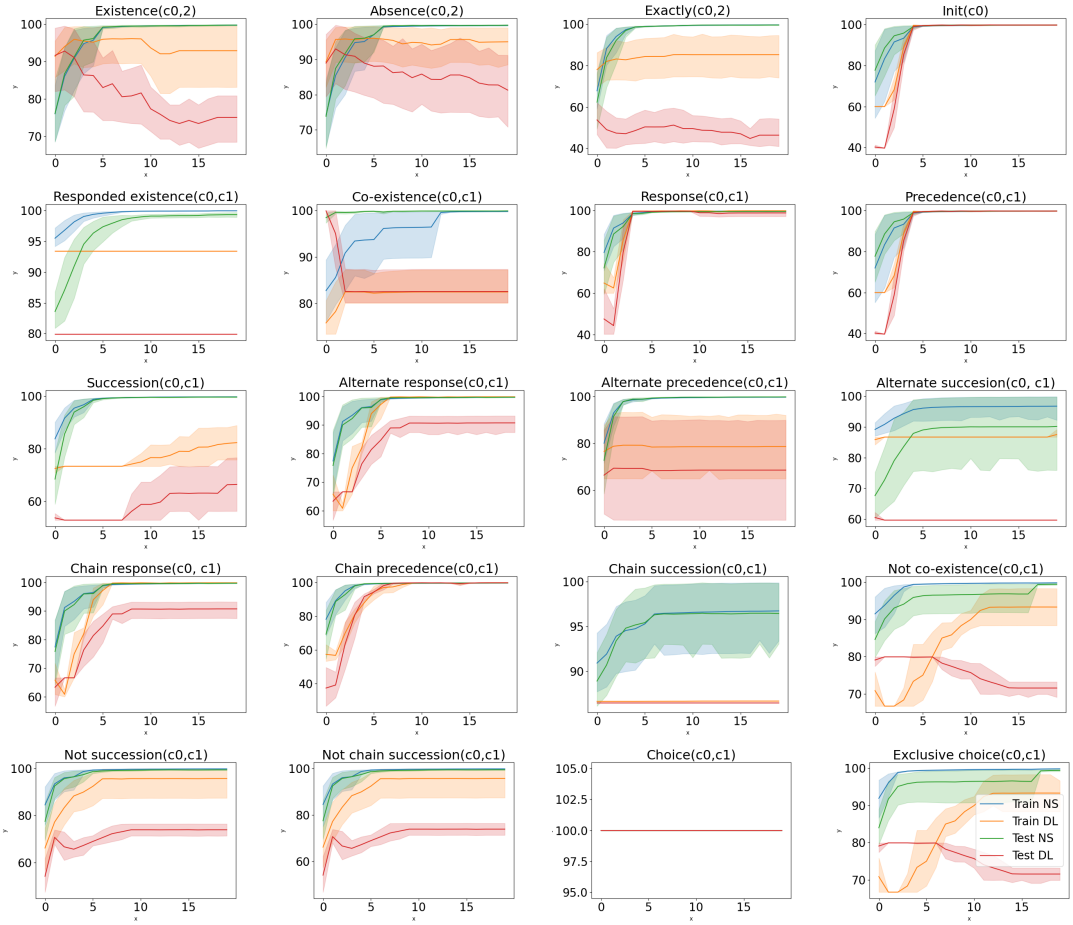


Figure 3: Experiments over 20 Declare constraints training on the **full dataset** in **mutually exclusive symbols**. On the y axis: **sequence classification** accuracy ; on the x axis: epochs of training. Solid lines represent mean values, shaded areas represent standard deviations.

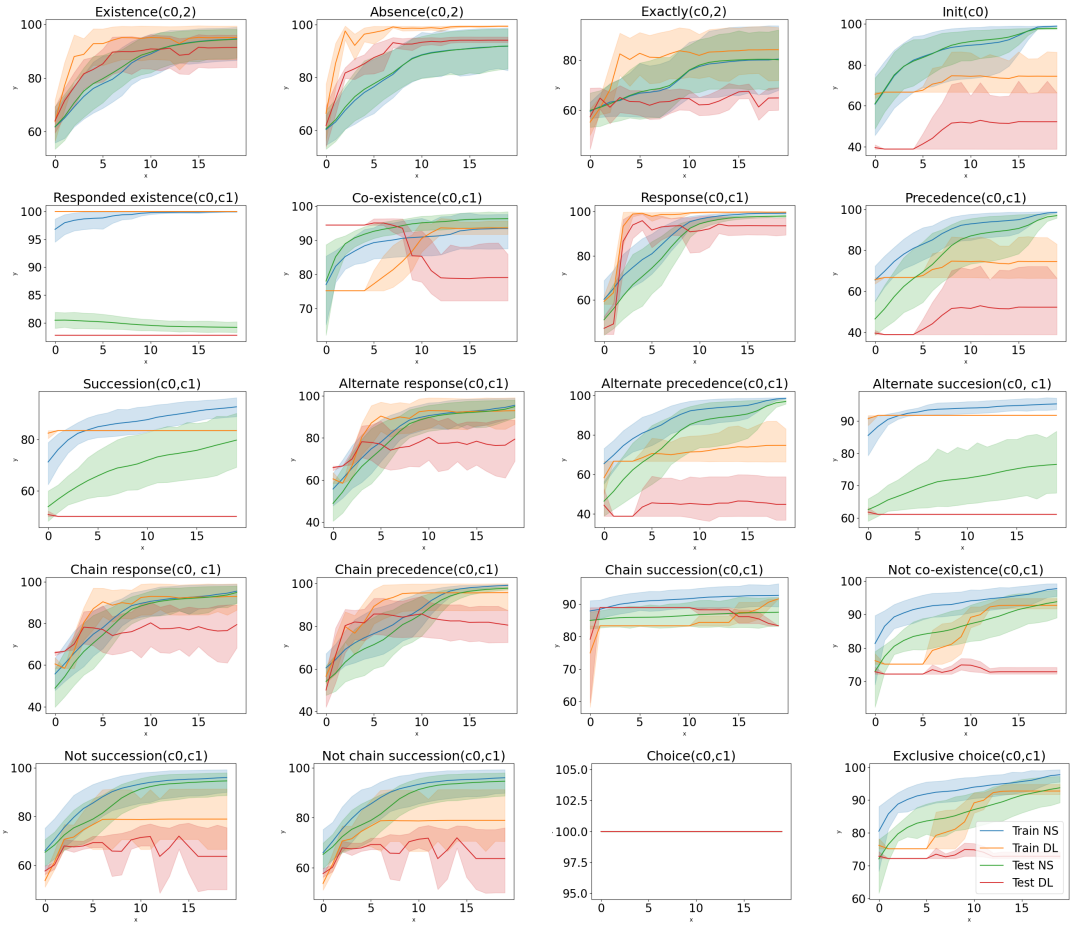


Figure 4: Experiments over 20 Declare constraints training on a **restricted dataset** in **mutually exclusive symbols** setting. On the y axis: **sequence classification** accuracy ; on the x axis: epochs of training. Solid lines represent mean values, shaded areas represent standard deviations.

A.1. Mutually exclusive symbols setting

Figure 3 shows the sequence classification accuracy of our approach (NS) and a convolutional+LSTM neural network (DL) when trained on the full dataset for the different Declare formulas. The results show that our approach outperforms the LSTM in all the formulas in the full-dataset settings. Our approach reaches the top accuracy in a couple of epochs for all the formulas. In contrast, the only-neural system struggles to reach the top, reaches it slower than our method, or completely overfits. Figure 3shows the same experiment conducted on fewer data, as described in the section Dataset. Performances of both methods degrade if we train on the restricted dataset; however, our method is more robust and maintains higher performances than the LSTM in 19 formulas over 20.

Figures 5 and 6 show the image classification accuracy obtained training on the full dataset and the small one, respectively. We observe that our system achieves top classification accuracy

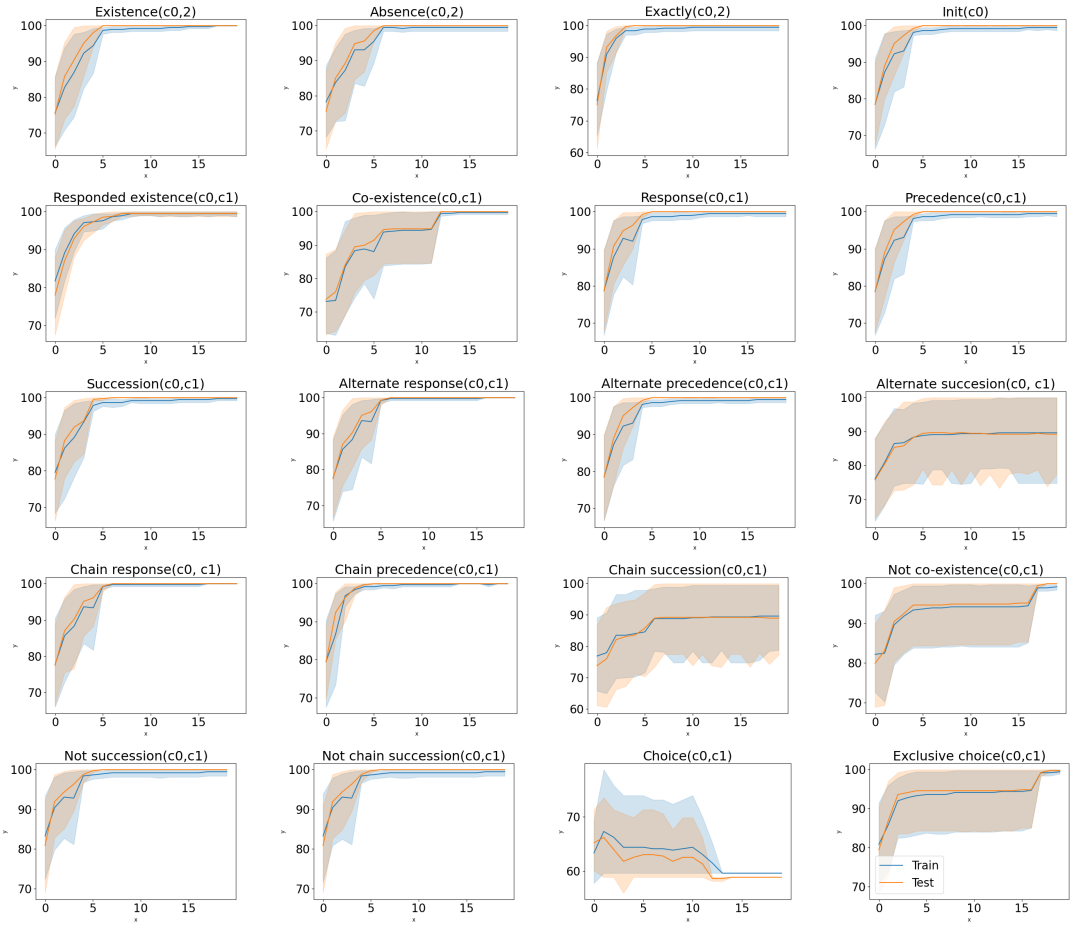


Figure 5: Experiments over 20 Declare constraints training on the **full dataset** in **non mutually exclusive symbols** setting. On the y axis: **image classification** accuracy ; on the x axis: epochs of training. Solid lines represent mean values, shaded areas represent standard deviations.

on the single images without being trained with any single-image label. This is particularly evident in the experiments over the full dataset.

A.1.1. Non mutually exclusive symbols

In this section, we describe results obtained in non-mutually exclusive symbol configurations. Figures 7 and 8 show the classification accuracy obtained on sequences and single images, respectively. Our approach remains better than the LSTM in 18 over 20 formulas, even if the formulas where designed for mutually exclusive symbol interpretations.

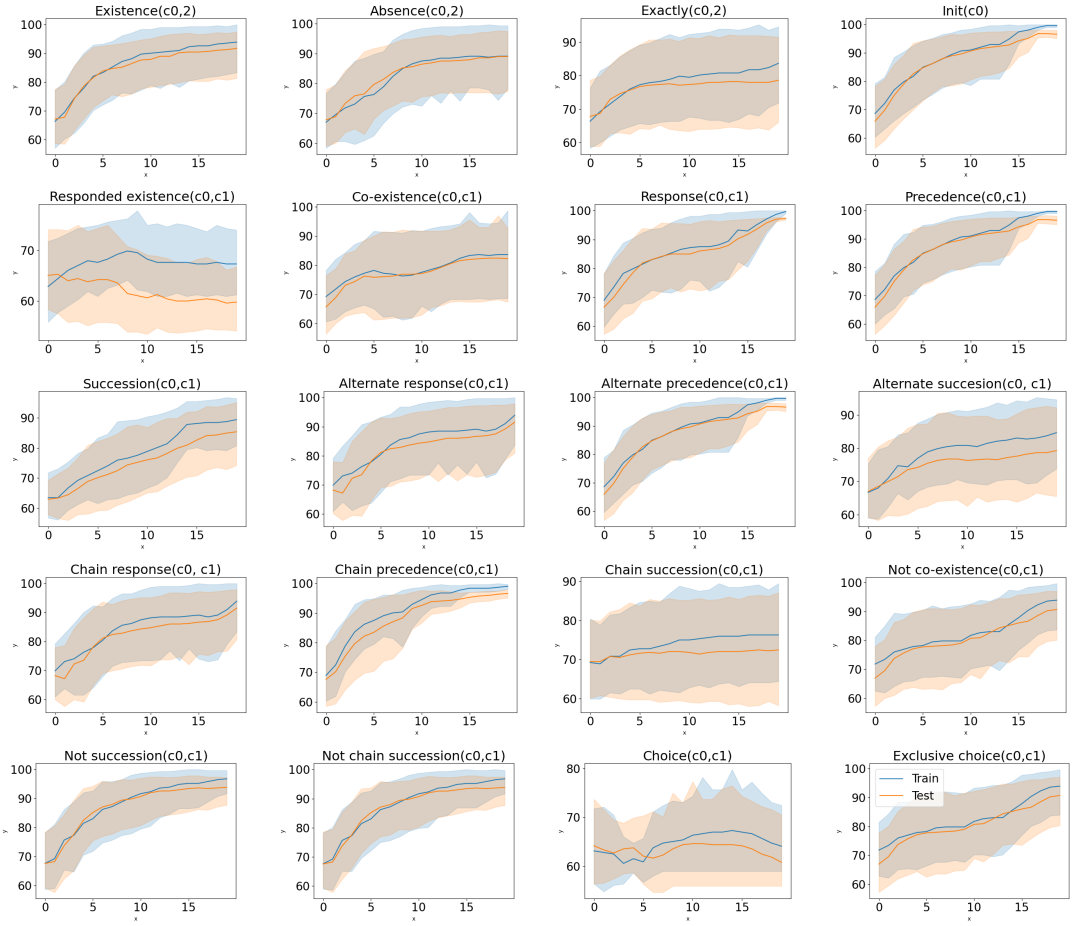


Figure 6: Experiments over 20 Declare constraints training on a **restricted dataset** in **mutually exclusive symbols** setting. On the y axis: **image classification** accuracy ; on the x axis: epochs of training. Solid lines represent mean values, shaded areas represent standard deviations.

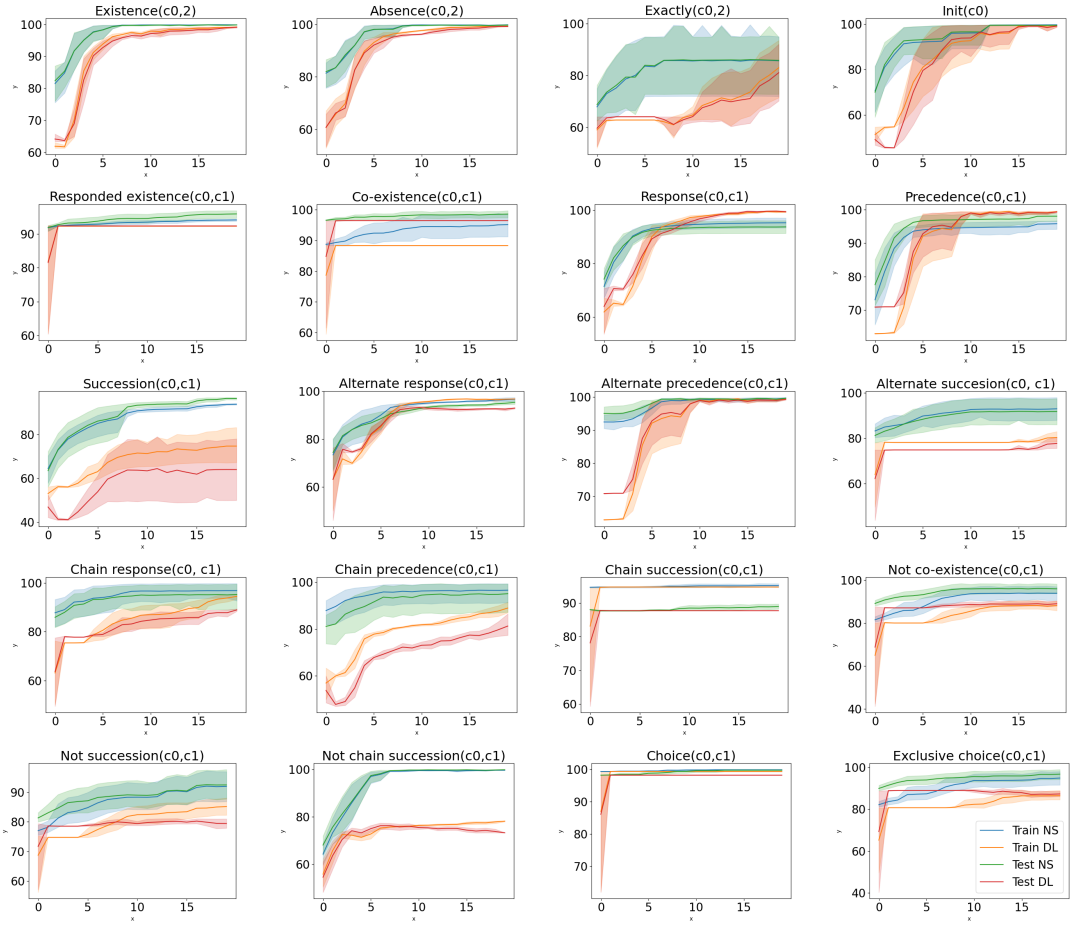


Figure 7: Experiments over 20 Declare constraints training on the **full dataset** in **non mutually exclusive symbols** setting. On the y axis: **sequence classification** accuracy ; on the x axis: epochs of training. Solid lines represent mean values, shaded areas represent standard deviations.

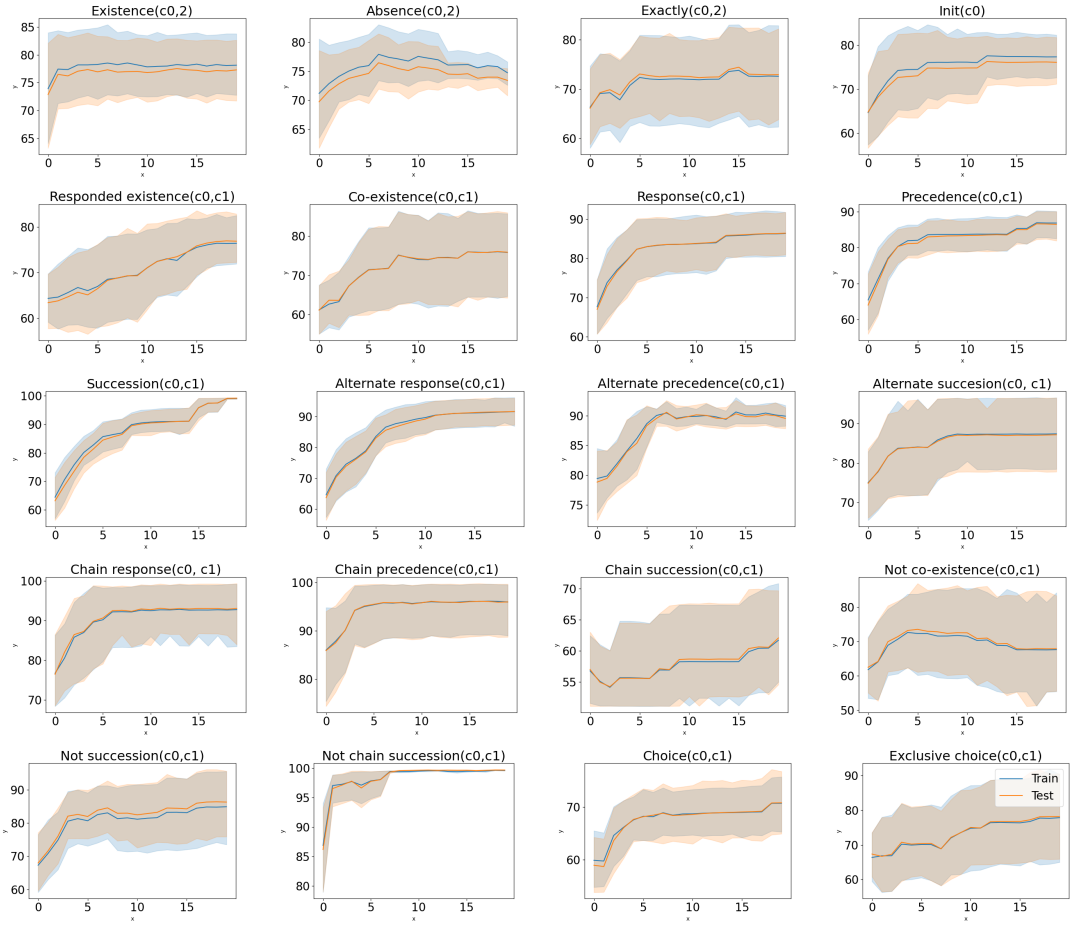


Figure 8: Experiments over 20 Declare constraints training on the **full dataset** in **non mutually exclusive symbols** setting. On the y axis: **image classification** accuracy ; on the x axis: epochs of training. Solid lines represent mean values, shaded areas represent standard deviations.

Template	LTL Formula [29]	Regular Expression [38]	R/I-Net Constructs	Description
Existence(A,n)	$\Diamond(A \wedge \bigcirc(existence(n-1, A)))$	.*(A.*){n}	$F = \{(A, p_1), (p_1, t_{sink})\}, W(p_1, t_{sink}) = n$	Activity A happens at least n times.
Absence(A,n)	$\neg existence(n, A)$	[^A]*(A?[^A]*){n-1}	$F = \{(t_{source}, p_1), (p_1, t_A)\}, W(t_{source}, p_1) = n - 1$	Activity A happens at most n times.
Exactly(A,n)	$existence(n, A) \wedge absence(n + 1, A)$	[^A]*(A[^A]*){n}	$F = \{(t_{source}, p_1), (p_1, t_A)\}, W(t_{source}, p_1) = n, I(t_{sink}) = p_1$	Activity A happens exactly n times.
Init(A)	A	(A.*)?	$F = \{(t_{source}, p_1)\}, \forall t \in T \setminus t_A, I(t) = p_1$	Each instance has to start with activity A.
Last(A)	$\Box(A \implies \neg X \neg A)$	.*A	$F = \{(t_A, p_1), (p_1, t_{sink})\}, \forall t \in T \setminus t_A, R(t) = p_1$	Each instance has to end with activity A.
Responded existence(A,B)	$\Diamond A \implies \Diamond B$	[^A]*((A.*B.*) |(B.*A.*))?	$F = \{(t_A, p_1), (t_B, p_2), (p_2, t_\lambda), (t_\lambda, p_3)\}, I(t_A) = p_3, I(t_B) = p_3, I(t_{sink}) = p_1, R(t_\lambda) = \{p_1, p_2\}$	If A happens at least once then B has to happen or happened before A.
Co-existence(A,B)	$\Diamond A \iff \Diamond B$	[^AB]*((A.*B.*) |(B.*A.*))?	$F = \{(t_A, p_1), (t_A, p_4), (t_B, p_2), (t_B, p_4), (p_1, t_\lambda), (p_2, t_\lambda), (t_\lambda, p_3)\}, I(t_A) = p_3, I(t_B) = p_3, I(t_{sink}) = p_4, R(t_\lambda) = \{p_1, p_2, p_4\}$	If A happens then B has to happen or happened after after A, and vice versa.
Response(A,B)	$\Box(A \implies \Diamond B)$	[^A]*(A.*B)*[^A]*	$F = \{(t_A, p_1)\}, I(t_{sink}) = p_1, R(t_B) = p_1$	Whenever activity A happens, activity B has to happen eventually afterward.
Precedence(A,B)	$(\neg B \, U \, A) \vee \Box(\neg B)$	[^B]*(A.*B)*[^B]*	$F = \{(t_A, p_1), (t_B, p_1), (p_1, t_B)\}$	Whenever activity B happens, activity A has to have happened before it.
Succession(A,B)	$response(A, B) \wedge precedence(A, B)$	[^AB]*(A.*B)*[^AB]*	$F = \{(t_A, p_1), (t_B, p_1), (p_1, t_B), (t_A, p_2)\}, I(t_{sink}) = p_2, R(t_B) = p_2$	Both Response(A,B) and Precedence(A,B) hold.
Alternate response(A,B)	$\Box(A \implies \bigcirc(\neg A \, U \, B))$	[^A]*(A[^A]*B[^A]*)*	$F = \{(t_{source}, p_1), (p_1, t_A), (t_A, p_2), (t_B, p_1)\}, I(t_{sink}) = p_2, R(t_A) = p_1, R(t_B) = p_2$	After each activity A, at least one activity B is executed. A following activity A can be executed again only after the first occurrence of activity B.
Alternate precedence(A,B)	$precedence(A, B) \wedge \Box(B \implies \bigcirc(precedence(A, B))$	[^B]*(A[^B]*B[^B]*)*	$F = \{(t_A, p_1), (p_1, t_B)\}, R(t_B) = p_1$	Before each activity B, at least one activity A is executed. A following activity B can be executed again only after the first next occurrence of activity A.
Alternate succession(A,B)	$altresponse(A, B) \wedge precedence(A, B)$	[^AB]*(A[^AB]*B[^AB]*)*	$F = \{(t_{source}, p_1), (p_1, t_A), (t_A, p_2), (p_2, t_B), (t_B, p_1)\}, I(t_{sink}) = p_2$	Both alternative response(A,B) and alternate precedence(A,B) hold.
Chain response(A,B)	$\Box(A \implies \bigcirc B)$	[^A]*(AB[^A]*)*	$F = \{(t_A, p_1)\}, I(t_A) = p_1, I(t_C) = p_1, R(t_B) = p_1$	Every time activity A happens, it must be directly followed by activity B (activity B can also follow other activities).
Chain precedence(A,B)	$\Box(\bigcirc B \implies A)$	[^B]*(AB[^B]*)*	$F = \{(t_{source}, p_1), (t_B, p_1), (t_C, p_1)\}, I(t_B) = p_1, R(t_A) = p_1$	Every time activity B happens, it must be directly preceded by activity A (activity A can also precede other activities).
Chain succession(A,B)	$\Box(A \iff \bigcirc B)$	[^AB]*(AB[^AB]*)*	$F = \{(t_{source}, p_2), (t_A, p_1), (t_B, p_2)\}, I(t_A) = p_1, I(t_B) = p_2, I(t_C) = p_1, R(t_A) = p_2, R(t_B) = p_1$	Activities A and B can only happen directly following each other.
Not co-existence(A,B)	$\neg(\Diamond A \wedge \Diamond B)$	[^AB]*((A[^B]*) |(B[^A]*))?	$F = \{(t_A, p_1), (t_B, p_2)\}, I(t_A) = p_2, I(t_B) = p_1$	Either activity A or B can happen, but not both.
Not succession(A,B)	$\Box(A \implies \neg(\Diamond B))$	[^A]*(A[^B]*)*	$F = \{(t_A, p_1)\}, I(t_B) = p_1$	Activity A cannot be followed by activity B, and activity B cannot be preceded by activity A.
Not chain succession(A,B)	$\Box(A \implies \neg(\bigcirc B))$	[^A]*(A+[^AB][^A]*)*A*	$F = \{(t_A, p_1)\}, I(t_B) = p_1, R(t_C) = p_1$	Activities A and B can never directly follow each other.
Choice(A,B)	$\Diamond A \vee \Diamond B$	.*[AB].*	$F = \{(t_{source}, p_1)\}, I(t_{sink}) = p_1, R(t_A) = p_1, R(t_B) = p_1$	Activity A or activity B has to happen at least once, possibly both.
Exclusive choice(A,B)	$(\Diamond A \vee \Diamond B) \wedge \neg(\Diamond A \wedge \Diamond B)$	([^B]*A[^B]*) |.*[AB].*([^A]*B[^A]*)	$F = \{(t_{source}, p_2), (t_A, p_1), (t_B, p_3)\}, I(t_A) = p_3, I(t_B) = p_1, I(t_{sink}) = p_2, R(t_A) = p_2, R(t_B) = p_2$	Activity A or activity B has to happen at least once, but not both.

Figure 9: List of Declare formulas as in [34]. We tested on all except last(a). Meaning of modal operators symbols:○=X, ◇=F, □=G