Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Grounding LTLf Specifications in Image Sequences

Elena Umili[1] , **Roberto Capobianco**[1,2] , **Giuseppe De Giacomo**[1,3]
[1]Sapienza University of Rome
[2]Sony AI
[3]University of Oxford
{umili, capobianco, degiacomo}@diag.uniroma1.it

Abstract

A critical challenge in neuro-symbolic (NeSy) 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 neuro-symbolic integration is handling the symbol grounding problem without direct supervision. We refer to symbol grounding [Steels2008] 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) [De Giacomo and Vardi2013]. LTLf is used in a big variety of domains, from robotics [He et al.2019] to Business Process Management (BPM) [Giacomo et al.2014], 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 [Kratsch, König, and Röglinger2022] or in non-Markovian Reinforcement Learning [Cai et al.2021], 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 (NN) perform extraordinarily well in perception tasks on raw data [Goodfellow, Bengio, and Courville2016]. 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, 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 [Grill et al.2020] [Caron et al.2021] without using any label. In particular, some approaches can also extract a discrete representation of the input data [Jang, Gu, and Poole2017], which can be considered an interpretation over a symbolic alphabet [Asai and Fukunaga2018, Dittadi, Drachmann, and Bolander2021, Umili et al.2021]. 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 paper, we build upon our previous work [Umili, Capobianco, and Giacomo2022], and 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. 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 neuro-symboilc AI [van Krieken, Acar, and van Harmelen2020], and many framework are based on it, such as Logic Tensor Networks (LTN) [Badreddine et al.2022] and Lyrics [Marra et al.2019]. 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; in section 5 we define when symbols are supposed to be *groundable* through the knowledge of an LTLf formula; we report the experiments evaluating our approach in section 6; and finally we conclude and discuss directions for future work in section 7.

2 Related Works

2.1 Semisupervised Symbol Grounding

Much prior work approaches the symbol grounding problem by assuming prior symbolic logical knowledge [Darwiche2011, Diligenti, Gori, and Saccà2017, Xu et al.2018, Badreddine et al.2022, Manhaeve et al.2018, Yang, Ishay, and Lee2020, Winters et al.2022, Dai et al.2019, Huang et al.2021, Tsamoura, Hospedales, and Michael2021]. These works infer the most probable grounding of a certain set of symbols P in non-symbolic data given prior logical knowledge expressed on the alphabet P and high-level labels on the whole NeSy process. This practice is known as semisupervised symbol-grounding and is tackled mainly with two families of methods.

The first approach [Darwiche2011, Diligenti, Gori, and Saccà2017, Xu et al.2018, Badreddine et al.2022, Manhaeve et al.2018, Yang, Ishay, and Lee2020, Winters et al.2022] consists in using continuous relaxations of the prior logical knowledge. The latter is encoded as a differentiable module $lm()$, while the symbol grounding function is learned through neural classifier $sg()$ mapping observations to symbols. The two functions are combined as two layers of a NN, and their composition $lm(sg(\cdot))$ is trained to maximize the given symbolic knowledge on the high-level labels available.

The second approach [Dai et al.2019, Huang et al.2021, Tsamoura, Hospedales, and Michael2021] instead maintains a crisp boolean representation of the logical knowledge and uses a process of logic *abduction*. Even in this approach, a neural classifier implements the symbol grounding function. The classifier outputs its current belief on the symbol's truth value, and an abduction module corrects its predictions to make them more consistent with the prior symbolic knowledge. After that, the classifier is *retrained* with the corrected symbols in a supervised fashion. This two-step training process has been shown to make the classifier converge to the target symbol grounding.

A benchmark for semisupervised symbol grounding is the digit addition problem [Manhaeve et al.2018], where a system must learn to classify MNIST digits images by knowing only the result of their sum and how addition works. This benchmark is not appropriate for our approach, since the addition task is not extended in time. For this reason, 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 that semi-supervised symbol grounding is applied to temporal specifications.

2.2 LTL and Reinforcement Learning

LTL formulas are widely used in Reinforcement Learning (RL) to specify non-Markovian tasks [Littman et al.2017]. Some work assumes prior knowledge of the LTL task specification [Camacho et al.2019, De Giacomo et al.2021], which is compiled into an automaton, and monitors the automaton state during the task execution to produce non-Markovian rewards for the task. Some other works focus on *learning* the finite machine corresponding to the task from traces of symbolic observations and rewards received by the environment, by using known methods for automata induction [Gaon and Brafman2020, Xu et al.2021, Ronca, Licks, and Giacomo2022].

All these works, however, use symbolic data and do not consider the problem of discovering latent symbols in the data. For this reason, they are applicable only in symbolic-state environments or continuous problems for which a mapping between the continuous state and a symbolic interpretation is known, also called labeled MDP [Cai et al.2021].

Many works assume to know an *imperfect* symbol grounding function for the task [Cai et al.2020, Verginis et al.2022, Li et al.2022]. Namely, a function that sometimes makes mistakes in predicting symbols from states or predicts a set of probabilistic *beliefs* over the symbol set instead of a boolean interpretation. These works can be considered a preliminary step towards integration with non-symbolic domains. However, they do not asses the problem of learning the symbol grounding function, but only how to use a pre-trained imperfect symbol grounder.

One work [Kuo, Katz, and Barbu2020] uses the LTLf specification of the task without assuming any knowledge of the symbol grounding function. This work employs a neural network architecture that is *shaped as the LTL formula* to learn a representation of states, that can be easily transferred to different LTL tasks in the same environment. The capability to transfer the representation to new tasks suggests that the representation can capture the semantics of symbols in some ways. However, the representation does not exactly ground the symbols in the environment observations.

3 Background

3.1 LTLf and DFA

Linear Temporal Logic (LTL) [Pnueli1977] 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) [De Giacomo and Vardi2013]. 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

$$\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \phi_1 U \phi_2 \qquad (1)$$

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) $F\phi \equiv \top U \phi$. 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 [Pnueli1977] for a formal description of the operators' semantics. Any LTLf formula ϕ can be translated in an equivalent Deterministic Finite Automaton (DFA) $A_\phi = (2^P, S, s_0, \delta_t, F)$, where 2^P is the automaton alphabet, S is the set of states, $s_0 \in S$ is the initial state, $\delta_t : S \times 2^P \to S$ is the transition function and $F \subseteq S$ is the set of final states. Let $L(A_\phi)$ be the language composed by all the strings accepted by the A_ϕ we have

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

Despite the size of A_ϕ is double-exponential in ϕ in the worst-case [De Giacomo and Vardi2013], A_ϕ is often quite small in practice, and scalable techniques are available for computing it from ϕ [Zhu et al.2017, Bansal et al.2020, Giacomo and Favorito2021].

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 [Pesic and van der Aalst2006]. The latter are at the base of the system Declare [Pesic, Schonenberg, and van der Aalst2007] and they generate DFAs that are polynomial in the original formula [Westergaard2011]. We use the Declare formulas as a benchmark for evaluating our approach.

3.2 Logic Tensor Networks

Logic Tensor Networks (LTN) [Badreddine et al.2022] are a neuro-symbolic 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 $I = i_0, i_1, ..i_{l-1}$ as compliant or not with a certain specification expressed as an LTLf formula ϕ, that is *not grounded* in the images. Each image is the 'rendering' of a symbolic interpretation over the formula alphabet P. This means that there exists a function $sg : \mathcal{I} \to 2^P$, where $\mathcal{I}$ 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 $\rho = \rho[0], \rho[1], ..., \rho[l-1]$, where l is the sequence length and $\rho[i] = sg(i_t)\ \forall 0 \leq t < l$.

We denote with $A_\phi = (2^P, Q, q_0, \delta_t, F)$ the DFA corresponding to the formula ϕ, where 2^P is the automaton alphabet, Q is the set of states, $q_0 \in Q$ is the initial state, $\delta_t : Q \times 2^P \to Q$ is the transition function and $F \subseteq Q$ is the set of final states. We will consider in the experiments both DFAs and Moore machines as target specifications. A Moore machine is a DFA augmented with a finite output alphabet $O = \{o_0, o_1, ...o_{N_O-1}\}$, where N_O is the number of output symbols, and an output function $\delta_o : Q \to O$. We represent the output function as a partition of the DFA states in N_O parts $\{Q_{o_0}, Q_{o_1}, ..., Q_{o_{N_O-1}}\}$. Note that a DFA is the Moore machine with binary alphabet $\{Accepted, Rejected\}$, with the states partitioned in accepting and non-accepting states. For this reason, we

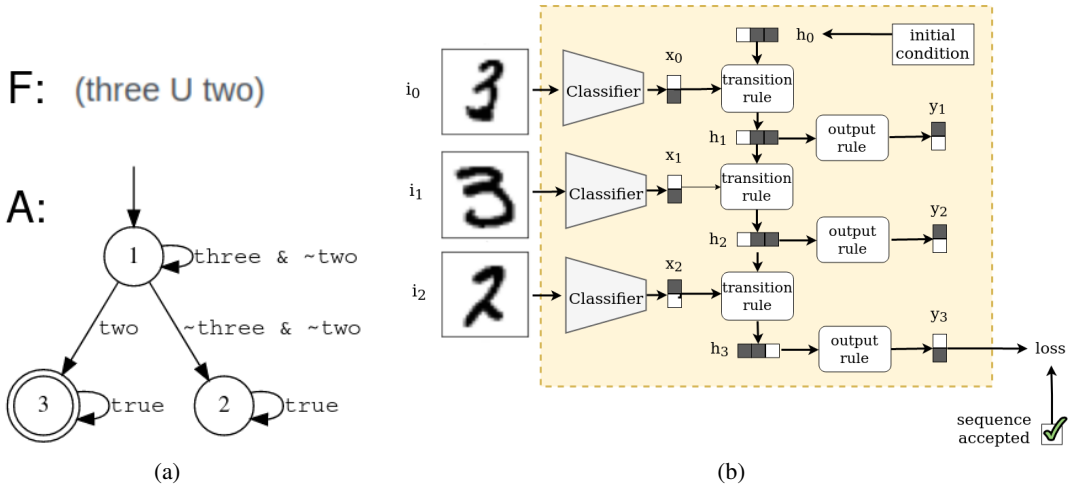


Figure 1: a) An example of LTLf formula with the corresponding equivalent automaton, b) our framework

will explain all the framework considering Moore machines, since they are more general.

If we run the trace ρ in the Moore machine we obtain a sequence of $l+1$ automaton states $q = q[0], q[1], ...q[l]$ and a sequence of $l+1$ output symbols $o = o[0], o[1], ..., o[l]$, where

$$\begin{aligned} q[0] &= q_0 \\ q[i] &= \delta_t(q[i-1], \rho[i-1]) \\ o[i] &= \delta_o(q[i]) \end{aligned} \tag{3}$$

We are interested in the symbol grounding function sg. We assume that we can discover it in a weakly supervised way. We assume therefore to know the following information: (i) the formula ϕ, from which we can build the DFA A_ϕ, eventually augmented with a non-binary output function δ_o, (ii) a set of training data D supervising the whole NeSy process composed of symbol grounding and symbolic processing with the machine.

In particular, we will consider two applications in the experiments: BPM-inspired domains and RL domains. Each of them corresponds to a particular setting for the dataset.

- D_{BPM} is the dataset for binary classification problems inspired by BPM applications. In this setting we want to classify sequences of images as compliant or not with the given formula, therefore we have supervision *only on the last step*. As a consequence D_{BPM} is made of associations $< I, \bar{y} >$, where I is an image sequence $i_0, i_1, ..i_{l-1}$ and $\bar{y} \in \{0, 1\}$ is the ground truth label denoting whether the sequence is accepted or not.
- D_{RL} is the dataset for non-Markovian RL domains. In this setting we consider rewards from the environment as labels, and we have therefore a label *for each step*. We consider reward functions that can be expressed as Reward Machines in the form of Moore machines, taking a finite number N_O of possible values. As a consequence, D_{RL} is made of couples $< I, \bar{Y} >$, where I is an image sequence $i_0, i_1, ..i_{l-1}$ and $\bar{Y}$ is the sequence of ground truth machine output labels $\bar{y_1}, \bar{y_2}, ..., \bar{y_l}$, where $\bar{y_i} \in \{0, 1, ..., N_O - 1\}$.

We will show this information is enough to learn the mapping from images to symbols for both the applications chosen.

4.2 Framework

We consider our framework as a neural network composed of two modules: (i) a *perception module*, implemented as a trainable convolutional neural network $CNN(i; \theta)$ with parameters θ, that classifies symbols from the input image i, approximating the function sg we want to discover; (ii) *a logic module*, implemented as a non-trainable recurrent structure, that is a fuzzy correspondent of the automaton A_ϕ. Figure 1(b) shows an example of the functioning of our framework.

The sequence of images $I = i_0, i_1, ..., i_{l-1}$ 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 $Symbol(p, t)$ denoting whether the t-th image in the sequence belongs to class p. The classifier implements the grounding of $Symbol$. In fact the component p of the CNN prediction for the image i_t in the sequence is the truth value of $Symbol(p, t)$

$$Symbol(p, t) = CNN(i_t; \theta)_p \tag{4}$$

where we denote with $CNN(\cdot; \theta)_p$ component p of the CNN output. We denote as $x_t = [Symbol(p_0, t), Symbol(p_1, t), ..Symbol(p_{|P|-1}, t)]$ the fuzzy interpretation over propositions in P at time t, corresponding to the complete output vector of the CNN. This fuzzy interpretation can be used to proceed on the automaton. Let us notice x_t is a fuzzy relaxation of the t-th point in the trace $\rho[t]$, referring to the background section 3.1.

In particular, at any time t we are in a state q_k of the automaton, we encode this information with another fuzzy predicate $State$, where $State(q_k, t)$ is true

if we are in state q_k at time t. Again, we define as h_t the interpretation at time t over the state symbols $h_t = [State(q_0, t), State(q_1, t), ..., State(q_{|Q|-1}, t)]$, with $|Q|$ equal to the number of states in the DFA. Note that h_t is a fuzzy relaxation of the state at time t denoted for (bolean) DFAs $q[t]$ in the previous section.

We denote the input and the state at time t of the fuzzy DFA x_t and h_t, respectively, to stress the connection with recurrent neural networks.

We start at the initial state q_0 of the automaton, and we have therefore

$$State(q_0, 0) = \top \wedge (State(q_i, 0) = \bot \; \forall 1 \leq i < |Q|) \quad (5)$$

Then we simulate a run of the automaton using the fuzzy symbolic interpretations x_0, x_1, x_{l-1} the classifier has predicted classifying the images.

We recall that for (boolean) DFAs, if at time t we are in a state q_i and we receive a certain interpretation $\rho[t]$ over the set of symbols, at time $t+1$ we transit to the state q_j linked to q_i by the edge $e_{i,j}$ that is made true by the interpretation of symbols in $\rho[t]$. 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 $State(s_j, t+1) = (State(s_i, t) \wedge e_{i,j}(\rho[t]))$, where we denote as $e_{i,j}(\rho[t])$ the truth value of the formula on arc $e_{i,j}$ when evaluated on the interpretation $\rho[t]$.

We apply the same transition rule in fuzzy logic. In particular:

$$State(q_j, t+1) = \bigvee_{i:(i,j) \text{ is an edge of } A_\phi} State(q_i, t) \wedge e_{i,j}(x_t) \quad (6)$$

Finally, we define the fuzzy predicate $Output$ to represent the output of the machine in a given time, denoting with $Output(o_i, t)$ whether the machine gives output o_i at time t. We have therefore

$$Output(o_i, t) = \bigvee_{q_k \in Q_{o_i}} State(q_k, l) \quad (7)$$

We denote the fuzzy relaxation of the machine output at time t with the vector $y_t = [Output(o_0, t), Output(o_1, t), ..., Output(o_{N_O-1}, t)]$. The predicted output must be equal to the ground truth label $\bar{y}$. We impose this by minimizing the loss

$$L = 1 - Output(\bar{y}_t, t) \quad (8)$$

We evaluate this loss on all the steps on which we have supervision, namely, the last step in the classification problem and all the steps in the RL problem. The loss is backpropagated through the network and the classifier parameters are updated accordingly. In summary, our knowledge base is composed of three logical axioms: (i) the initialization condition (Equation 5), (ii) the transition rule (Equation 6), (iii) the output rule (Equation 7). In particular, the initial condition only specifies the initial state and does not depend on the classifier predictions. The transition rule calculates the next state given the current automaton state and the symbol prediction over the current image. The output rule calculates the current output given the current state. These two rules are applied recursively as many times as many images compose the sequence. Finally, the loss applies supervision over the output exploiting the sequence labels present in the dataset.

We ground the truth value of each logical axiom by using the following fuzzy operators: 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. In particular, we use the set of fuzzy operators suggested by [Badreddine et al.2022], since our method is an extension of Logic Tensor Networks.

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

Figure 1(b) shows the network behavior in case of perfect grounding. In this case, the classifier predicts all one-hot encodings, this represents an ideal situation where no uncertainty is present, and symbols are all perfectly true or perfectly false. Consequently, the outputs from the fuzzy transition and output function are also perfectly boolean, and the fuzzy automaton behaves exactly as the original (boolean) machine. The benefit of having a fuzzy automaton is that it can predict the sequence of states even with some uncertainty in the symbol grounding layer, while the original DFA cannot handle any uncertainty. Furthermore, transitions are differentiable, and we can back-propagate error through the model.

Intuitively, at the beginning of training, the classifier does not know how to map images into symbols; therefore, this grounding will initially be random and potentially incorrect, so the automaton states and outputs produced by taking this grounding as inputs. Although we do not know the ground truth sequence of states, the fuzzy automaton produces a sequence of probabilities over these states that are adjusted to be coherent with the output labels, which in turn adjusts the predicted labels over images. For example, if we know that the DFA accepts the sequence, the last state must be a final state. Therefore, the second-last must be linked to one final state; the third-last must be linked to the second-last, and so forth. All of this is automatically optimized through gradient descent.

5 Definition and Examples of Groundability through a Temporal Property

In this section, we formalize the concept of *groundability* of symbols through an LTLf formula.

We say a symbol $p \in P$ is groundable through an LTLf formula ϕ, where P is the formula alphabet, if p is distinguishable from all the other symbols in the formula alphabet.

Undistinguishability of two symbols: We define two propositional symbols $p_1, p_2 \in P$ *indistinguishable* from each other through the LTLf formula ϕ, defined over symbols in P, if and only if, given a generic finite trace ρ,

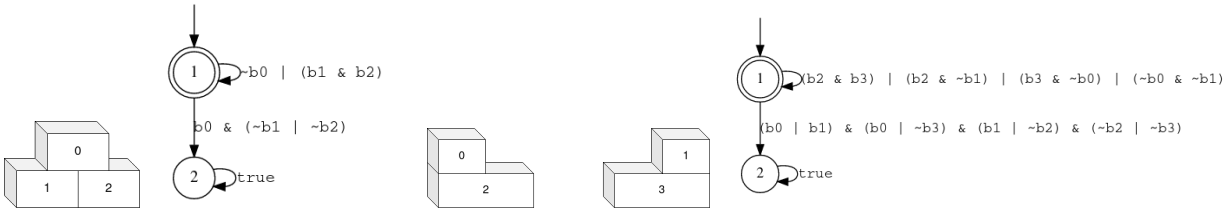


Figure 2: (a) **Pyramid example**. (left) Drawing of the pyramid. (right) DFA corresponding to the instructions to build the pyramid. Bricks are not all groundable through the pyramid instructions, in fact brick 1 and 2 can be confused each other. In this case the classifier can either: (i) learn to ground images of brick1 in symbol b_1 and images of brick2 in the symbol b_2, this lead to 100% accuracy; (ii) learn to ground images of brick1 in symbol b_2 and images of brick2 in the symbol b_1, this lead to 33% (1/3) accuracy in a balanced dataset, because only one brick over 3 is grounded correctly (brick0). (b) **Gate example**. (left) Drawing of the gate. (right) DFA corresponding to the instructions to build the gate. Bricks are *all ungroundable* through the gate instructions, in fact brick 0 and 1 can be confused each other if bricks 2 and 3 are confused each other. Expected symbol grounding accuracy: 100% or 0%.

and the trace $\tilde{\rho}$ obtained by ρ replacing truth values of p_1 with those of p_2 and truth values of p_2 with those of p_1, $\rho \vDash \phi \iff \tilde{\rho} \vDash \phi, \forall \rho \in (2^P)^*$.

Let us consider a generic 'rendering' function r that transforms the trace ρ in a sequence of non-symbolic observations $r(\rho)$. We denote the symbol grounding function as $sg = r^{-1}$. Suppose we want to discover sg supervising non-symbolic traces $r(\rho)$ with the formula acceptance and p_1 and p_2 are indistinguishable. In that case, the symbol grounder will never receive an error different from 0 for grounding renderings of p_1 with p_2 and vice versa, and this *does not depend* on the rendering function or the grounding function, but only on the structure of the formula.

Let us further explain the concept with some examples in the Brick World domain. Consider an assembly process for which certain precedence constraints apply to the assembly of the various parts. A simple example of this is building a brick wall. We must impose that if a brick b_0 rests on other bricks $(b_1, b_2, ..., b_k)$, these must be placed on the wall *before* the brick b_0. During the construction process, we indicate with the propositional variable b_i whether the brick i is placed on the wall. Considering the wall in the figure 2(a), made up of 3 bricks, the precedence constraint corresponds to the following LTLf formula.

$$\phi_{pyramid} = G(b_0 \Rightarrow (b_1 \wedge b_2)) \tag{9}$$

Let us suppose we want to ground the symbols b_i in observations of the wall, while it is under construction, supervising the grounding with compliance with the given assembly instruction $\phi_{pyramid}$. Since the formula accepts both the sequences in which bricks are placed in order b_2-b_1-b_0 and in order b_1-b_2-b_0, only symbol b_0 is 'groundable' through the formula, while b_1 and b_2 are not distinguishable from each other. We can infer this also by looking at the formula $\phi_{pyramid}$ or the equivalent DFA $A_{pyramid}$ shown in Figure 2(a). If we replace symbol b_1 with symbol b_2 and vice versa, the formula does not change. Equivalently, if we exchange the two symbols on all the arcs of the equivalent DFA, the automaton is not modified.

The given definition of indistinguishability is not strong enough to cover all the cases where symbols are ungroundable. Let us clarify why with another example in the Brick World domain. Consider the 'gate' example, shown in Figure 2(b), described by the formula

$$\phi_{gate} = (G(b_0 \Rightarrow b_2)) \wedge (G(b_1 \Rightarrow b_3)) \tag{10}$$

According to the previous definition, the bricks are all groundable in this example because there does not exist a couple of symbols satisfying the definition of indistinguishability of two symbols. However, in this example, none of the bricks is *really* groundable. Because bricks 0 and 1 can be confused by each other *if* bricks 2 and 3 are confused with each other. Therefore a symbol grounder is supposed to achieve either 100% or 0% accuracy, no matter the rendering function.

For this reason, we extend the definition to a *set* of symbols as follows.

Ungroundability of a set of symbols: Given n couples of distinct symbols $(p_{1,1}, p_{2,1}), (p_{1,2}, p_{2,2}), ..., (p_{1,n}, p_{2,n})$, we say the set of symbols $\{p_{1,1}, p_{2,1}, ..., p_{1,n}, p_{2,n}\}$ is ungroundable if and only if $\forall \rho \in (2^P)^*$, let $\tilde{\rho}$ be the trace constructed by replacing truth values of $p_{1,i}$ with those of $p_{2;i}$ for all $1 \leq i \leq n$, $\rho \vDash \phi \iff \tilde{\rho} \vDash \phi$.

Consequently, we define a symbol as *groundable* when it does not exist any set of substitutions that makes it indistinguishable from another symbol.

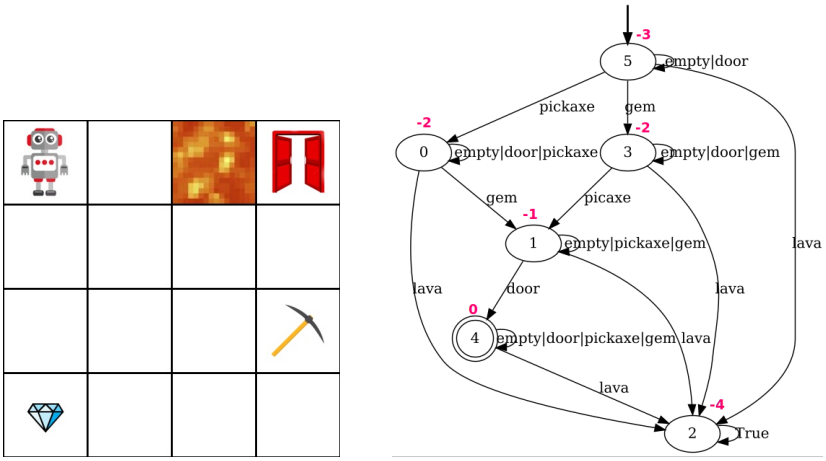
The reader can verify on the gate formula or DFA, in Figure 2(b), that the specification remains the same if symbols b_0 and b_1 replace each other and symbols b_2 and b_3 replace each other, while it changes by performing only one substitution of the two.

6 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.

We conduct experiments in two different domains, inspired by two research areas in which LTLf formulas are popularly used: BPM and non-Markovian RL.

We compare our neuro-symbolic approach (NS) with a classical supervised deep learning approach (DL). We im-



(a) (b)

Figure 3: a) Visual Minecraft environment. b) Task specification as Reward Machine.

plement 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.

In each experiment, we report the sequence classification accuracy, namely the ratio of correctly evaluated sequences; and the image classification accuracy, e.g., the ratio of correctly predicted symbolic interpretation in single images.

6.1 BPM-Inspired Application

Since LTL can be used to specify innumerable constraints in BPM applications, 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 [Pesic, Schonenberg, and van der Aalst2007] 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. We refer the reader to [De Smedt et al.2015] for the 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).

6.2 Non-Markovian-RL-Inspired Application

For the second experiment, we propose the Visual Minecraft environment, an example of a non-Markovian task with non-symbolic states, similar to that considered by [Corazza, Gavran, and Neider2022, Li et al.2022], but with image states. A robotic agent can navigate a grid world environment through 4 movement actions $\{left, right, up, down\}$. Each cell of the grid can be empty or contain one of the following items: *pickaxe*, *gem*, *lava*, *door*

The task consists in collecting a pickaxe *and* a gem (it does not matter in which order the two items are collected) and then going to the door, while always avoiding the lava. The task specification corresponds to the DFA in figure 3(b). The specification is expressed in terms of five symbols

$$P = \{pickaxe, gem, lava, door, empty\} \tag{11}$$

One symbol for each item, that is True when the agent is in a cell containing that item, plus the symbol 'empty' that is True when no items are in the agent cell. We transform the DFA into a Moore machine by defining a reward function on its states that is maximum on the final states. In particular, we define the reward on state q_i as the opposite of the distance to the closest final state, when this distance is smaller than ∞, and equal to a negative constant C for states disconnected from the final states.

$$r(q_i) = max\{\min_{f \in F} dist(q_i, f), C\} \tag{12}$$

In this way, we obtain a quite dense reward function, which is shown in red on the automaton states in Figure 3(b). Figure 3(a) shows an example of image state returned by the environment during the interaction with it.

6.3 MNIST Dataset

For the BPM inspired task, since Declare formulas are defined on the binary alphabet $P = \{0, 1\}$, we create the dataset by rendering symbolic configurations using images of zeros and ones from the MNIST dataset [Lecun et al.1998]. For each formula, all the possible symbolic traces with length between 1 and 4 are created and labeled as accepted or rejected. Note that we supervise only the output in the *last step* of the sequence. Symbolic traces 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,

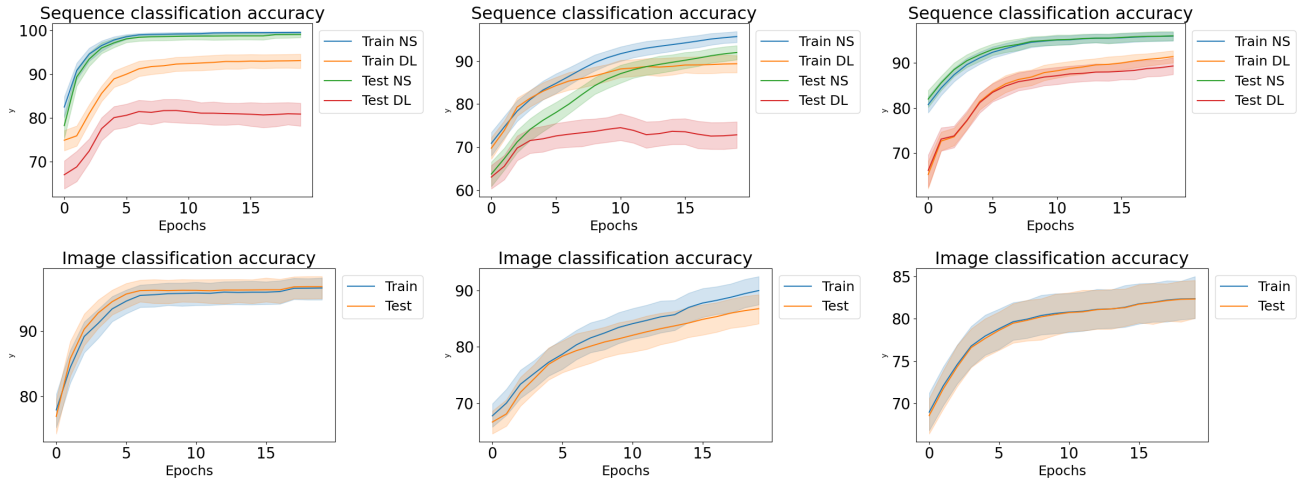


Figure 4: Experiments over 20 Declare formulas. In the first row: **sequence classification** accuracy, in the second row: **image classification** accuracy. They are obtained by training on three different datasets: (first columns) **complete dataset**, (second column) **restricted dataset**, (third column) **complete dataset with non-mutually exclusive symbols**. Solid lines represent mean values, shaded areas represent standard deviations.

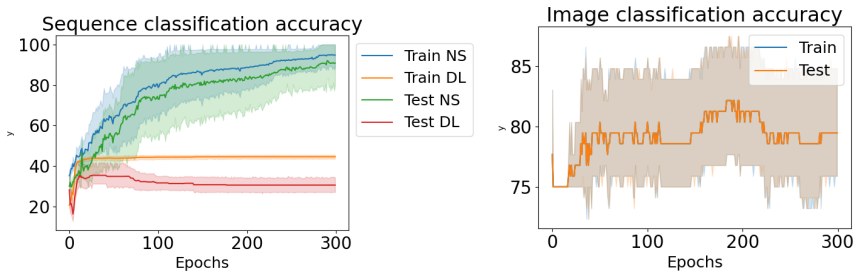


Figure 5: Results in the Visual Minecraft environment. Left) Accuracy over sequences during training of the symbol grounder. Right) Symbol grounding accuracy on single images.

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 a 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 the MNIST dataset: when all symbols are set to false (rendered as a black image), and when both symbols are 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%.

6.4 Minecraft Dataset

For the application to the Minecraft environment, we create a dataset simulating 40 episodes in the environment and collecting the images rendered by the environment. Each episode lasts 30 steps; therefore, it produces a sequence of 30 images corresponding to the environment states and a sequence of 30 reward values, that are used to label the image sequence *at each step*. Episodes are balanced between positive and negative examples. In 20 episodes, the agent correctly collects all the items and goes to the door avoiding the lava; in the other 20, it fails. We split the dataset in a 80% for training and 20% for testing.

6.5 Results on MNIST Dataset

Figure 4 shows the mean results over the 20 different Declare formulas. In all the plots solid line is the mean, and the shaded area represents the standard deviation. In the image sequence classification task, Figure 4 (first row), our approach outperforms the pure deep learning approach in all the 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 on the single formulas. This confirms our intuition that the LTLf knowledge can be exploited to simplify the learning process. In particular, this happens because our method has to learn only to extract the right "visual" features and has the "temporal" ones somehow encoded in the automaton's knowledge, while the DL approach has to learn both the "visual" and "temporal" features from the data.

In the image classification task, Figure 4 (second row), our approach reaches high accuracy on both the test and training sets without exploiting any image label, that is a quite impressive result.

6.6 Results on the Minecraft Environment

Figure 5 shows train and test sequence classification accuracy (left) and symbol grounding accuracy on single images (right). Let us notice the task specification has two ungroundable symbols: gem and $pickaxe$. Since the agent can collect these two items in any order, the framework does not receive enough supervision to distinguish between the two. Therefore image classification does not achieve 100% accuracy. However, sequence classification can still achieve top accuracy even if the symbol grounder confuses the gem for the $pickaxe$ and vice-versa. The end-to-end deep learning approach performs very poorly in this task, obtaining only 40% of sequence accuracy. Investigating the reason for these poor performances, we found that the NN almost never predicts rewards of -1 and -2, corresponding to the scarcest reward labels in the dataset. In fact, even if we balanced the dataset between positive (reward = 0 in the last step) and negative (reward < 0 in the last step) episodes, the reward labels are not balanced within the episodes. Learning classification tasks from highly biased data with neural networks can be very difficult, as experiments in the Minecraft environment show. However, as the figure shows, our approach is unaffected by the label imbalance. Let us notice some differences between the two applications considered. First, we have two different levels of supervision in the two tasks. In the RL domain, the rewards label sequences at each step, while in the BPM domain, labels supervise only the last step. Apart from that, the Minecraft task is more challenging for many reasons. First, the number of symbols to recognize is larger. Furthermore, the image sequences are longer than in the MNIST dataset; we consider traces of length 30 in the Minecraft task and 4 in the MNIST task. Let us also notice that the environment highly biases the distribution of symbols and reward labels in sequences. For example, the 'empty' symbol is much more frequent than the others, and most possible symbolic traces are unfeasible in the environment and, therefore, never observed. For example, the agent cannot jump from one item to the other, but it has to walk and observe many empty cells in between. This can complicate the classification task.

7 Conclusion and Future Work

In conclusion, we propose a framework for exploiting high-level logical knowledge in the form of LTLf formulas in classification task over sequences of images. In particular, we use the temporal 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. We have shown that our framework is applicable in different domains, ranging from BPM-inspired applications to visual non-Markovian RL environments. Furthermore, our approach outperforms the end-to-end approach based on recurrent neural networks in sequence classification: it is more general and can maintain high performances using fewer labels or highly unbalanced labels. In future work, we want to apply this framework to a more realistic scenario in the area of BPM and integrate it with RL algorithms for non-Markovian tasks.

Acknowledgments

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by theEU ICT-48 2020 project TAILOR (No. 952215), by the PRIN project RIPER (No. 20203FFYLK) and by the PNRR MUR project PE0000013-FAIR. Furthermore we thank Francesco Argenziano for his interest in this project.

References

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *Proceedings of the AAAI Conference on Artificial Intelligence* 32(1).

Badreddine, S.; d'Avila Garcez, A.; Serafini, L.; and Spranger, M. 2022. Logic tensor networks. *Artificial Intelligence* 303:103649.

Bansal, S.; Li, Y.; Tabajara, L. M.; and Vardi, M. Y. 2020. 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*, 9766–9774. AAAI Press.

Cai, M.; Xiao, S.; Li, B.; Li, Z.; and Kan, Z. 2020. Reinforcement learning based temporal logic control with maximum probabilistic satisfaction. *2021 IEEE International Conference on Robotics and Automation (ICRA)* 806–812.

Cai, M.; Hasanbeig, M.; Xiao, S.; Abate, A.; and Kan, Z. 2021. Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE Robotics and Automation Letters* 6(4):7973–7980.

Camacho, A.; Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2019. 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*, 6065–6073. International Joint Conferences on Artificial Intelligence Organization.

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

Corazza, J.; Gavran, I.; and Neider, D. 2022. Reinforcement learning with stochastic reward machines. *Proceedings of the AAAI Conference on Artificial Intelligence* 36(6):6429–6436.

Dai, W.-Z.; Xu, Q.; Yu, Y.; and Zhou, Z.-H. 2019. Bridging machine learning and logical reasoning by abductive learning. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Darwiche, A. 2011. Sdd: A new canonical representation of propositional knowledge bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, 819–826. AAAI Press.

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

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2021. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. *Proceedings of the International Conference on Automated Planning and Scheduling* 29(1):128–136.

De Smedt, J.; vanden Broucke, S.; Weerdt, J.; and Vanthienen, J. 2015. A full r/i-net construct lexicon for declare constraints. *SSRN Electronic Journal*.

Diligenti, M.; Gori, M.; and Saccà, C. 2017. Semantic-based regularization for learning and inference. *Artificial Intelligence* 244:143–165. Combining Constraint Solving with Mining and Learning.

Dittadi, A.; Drachmann, F. K.; and Bolander, T. 2021. Planning from pixels in atari with learned symbolic representations. In *AAAI*.

Gaon, M., and Brafman, R. 2020. Reinforcement learning with non-markovian rewards. *Proceedings of the AAAI Conference on Artificial Intelligence* 34(04):3980–3987.

Giacomo, G. D., and Favorito, M. 2021. Compositional approach to translate ltlf/ldlf into deterministic finite automata. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, 122–130. AAAI Press.

Giacomo, G. D.; Masellis, R. D.; Grasso, M.; Maggi, F. M.; and Montali, M. 2014. Monitoring business metaconstraints based on ltl and ldl for finite traces. In *BPM*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.

Grill, J.-B.; Strub, F.; Altché, F.; Tallec, C.; Richemond, P. H.; Buchatskaya, E.; Doersch, C.; Ávila Pires, B.; Guo, Z.; Azar, M. G.; Piot, B.; Kavukcuoglu, K.; Munos, R.; and Valko, M. 2020. Bootstrap your own latent - a new approach to self-supervised learning. In *NeurIPS*.

He, K.; Wells, A. M.; Kavraki, L. E.; and Vardi, M. Y. 2019. Efficient symbolic reactive synthesis for finite-horizon tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, 8993–8999.

Huang, Y.-X.; Dai, W.-Z.; Cai, L.-W.; Muggleton, S. H.; and Jiang, Y. 2021. Fast abductive learning by similarity-based consistency optimization. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 26574–26584. Curran Associates, Inc.

Jang, E.; Gu, S.; and Poole, B. 2017. 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.

Kratsch, W.; König, F.; and Röglinger, M. 2022. Shedding light on blind spots – developing a reference architecture to leverage video data for process mining. *Decision Support Systems* 158:113794.

Kuo, Y.; Katz, B.; and Barbu, A. 2020. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*, 5604–5610.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Li, A. C.; Chen, Z.; Vaezipoor, P.; Klassen, T. Q.; Icarte, R. T.; and McIlraith, S. A. 2022. Noisy symbolic abstractions for deep RL: A case study with reward machines. *CoRR* abs/2211.10902.

Littman, M. L.; Topcu, U.; Fu, J.; Jr., C. L. I.; Wen, M.; and MacGlashan, J. 2017. Environment-independent task specifications via GLTL. *CoRR* abs/1704.04341.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. Deepproblog: Neural probabilistic logic programming. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Marra, G.; Giannini, F.; Diligenti, M.; and Gori, M. 2019. Lyrics: A general interface layer to integrate logic inference and deep learning. In *ECML/PKDD*.

Pesic, M., and van der Aalst, W. M. 2006. A declarative approach for flexible business processes management. In *Business Process Management Workshops*.

Pesic, M.; Schonenberg, H.; and van der Aalst, W. M. 2007. Declare: Full support for loosely-structured processes. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, 287–287.

Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, 46–57. IEEE Computer Society.

Ronca, A.; Licks, G. P.; and Giacomo, G. D. 2022. Markov abstractions for PAC reinforcement learning in non-markov decision processes. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, 3408–3415.

Steels, L. 2008. The symbol grounding problem has been solved. so what's next. *Symbols and embodiment: Debates on meaning and cognition* 223–244.

Tsamoura, E.; Hospedales, T.; and Michael, L. 2021. Neural-symbolic integration: A compositional perspective. *Proceedings of the AAAI Conference on Artificial Intelligence* 35(6):5051–5060.

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

Umili, E.; Capobianco, R.; and Giacomo, G. D. 2022. Grounding ltlf specifications in images. In *Proceedings of the 16th International Workshop on Neural-Symbolic Learning and Reasoning as part of the 2nd International Joint Conference on Learning & Reasoning (IJCLR 2022), Cumberland Lodge, Windsor Great Park, UK, September 28-30, 2022*, 45–63.

van Krieken, E.; Acar, E.; and van Harmelen, F. 2020. Analyzing Differentiable Fuzzy Implications. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, 893–903.

Verginis, C. K.; Köprülü, C.; Chinchali, S.; and Topcu, U. 2022. Joint learning of reward machines and policies in environments with partially known semantics. *CoRR* abs/2204.11833.

Westergaard, M. 2011. Better algorithms for analyzing and enacting declarative workflow languages using ltl. In Rinderle-Ma, S.; Toumani, F.; and Wolf, K., eds., *Business Process Management*, 83–98. Berlin, Heidelberg: Springer Berlin Heidelberg.

Winters, T.; Marra, G.; Manhaeve, R.; and Raedt, L. D. 2022. Deepstochlog: Neural stochastic logic programming. *Proceedings of the AAAI Conference on Artificial Intelligence* 36(9):10090–10100.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Van den Broeck, G. 2018. A semantic loss function for deep learning with symbolic knowledge. In Dy, J., and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 5502–5511. PMLR.

Xu, Z.; Wu, B.; Ojha, A.; Neider, D.; and Topcu, U. 2021. Active finite reward automaton inference and reinforcement learning using queries and counterexamples. In *Machine Learning and Knowledge Extraction - 5th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2021, Virtual Event, August 17-20, 2021, Proceedings*, 115–135.

Yang, Z.; Ishay, A.; and Lee, J. 2020. Neurasp: Embracing neural networks into answer set programming. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 1755–1762. International Joint Conferences on Artificial Intelligence Organization. Main track.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic ltlf synthesis. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 1362–1369.