Logical and statistical approaches to XAI

Valentyn Boreiko
IST Austria

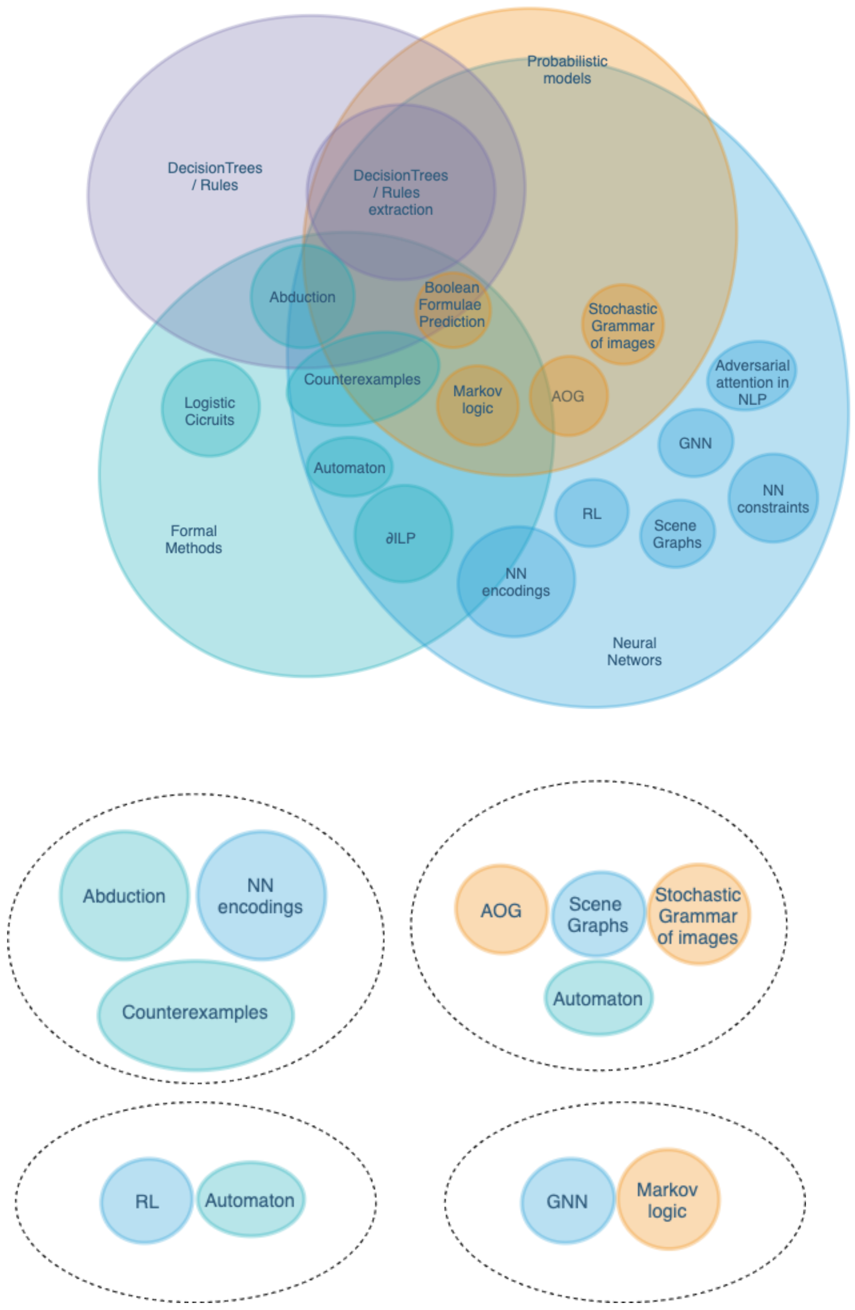
June 14, 2020

Abstract

Exlainable AI (XAI) is a hot trend that requires some unification as it is applied in different ways to each of the domains. This work provides an overview of the domains and respective models of ML, XAI methods used for them, and some methods that try to generalize across methods and / or domains. The domains under consideration are: NNs in general, graph neural networks (GNNs), computer vision, robotics and reinforcement learning (RL), and methods that generalize across domains.

1 Introduction

We will describe both the domain-specific XAI models as well as the models that can generalize across domains.
Specific domains (Computer vision (CV), RL, etc.) require some sort of ”preprocessing” to provide explanations to make them global and white-box.
Neuran networks (NNs) in general use approaches such as encodings of the network (e.g., mixed-integer linear programming (MILP)), which are used in the Abduction model that generalizes across other models as well, or combination of NNs with inductive logic programming (ILP);
Graph neural networks (GNNs) and their embeddings can be combined with Markov logic networks and their predicates.
Schematically, the complexity of the XAI approaches can be represented in the diagram 1 and 2 below


Probabilistic models
DecisionTrees / Rules
DecisionTrees / Rules extraction
Abduction
Boolean Formulae Prediction
Stochastic Grammar of images
Adversarial attention in NLP
Counterexamples
Logistic Cicruits
Markov logic
AOG
GNN
Automaton
NN constraints
RL
Scene Graphs
∂ILP
Formal Methods
NN encodings
Neural Networs
Abduction
NN encodings
Counterexamples
AOG
Scene Graphs
Stochastic Grammar of images
Automaton
RL
Automaton
GNN
Markov logic

2 XAI for different domains

2.1 NNs

2.1.1 NN encoding (MILP)

Here we will focus on the mixed-integer linear programming (MILP) encoding of a NN with an activation function represented via a rectified linear unit (ReLU). Another encoding can be a binarized neural network (e.g., [20]) or a prime compilation of Boolean formulas ([2]), or a compilation into a binary decision diagram (BDD).
Let $A \in \mathbb{R}^m \times \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Having input $x = \{x_1, ..., x_n\}, x \in \mathbb{R}^n$ and output $y = \{y_1, ..., y_m\}, y \in \mathbb{R}^m_{\geq 0}$ each of the blocks $y = ReLU(Ax + b)$ in the NN in case of MILP for $i \in \{1, ..., m\}$ will be encoded as follows:

$$\sum_{j=1}^{n} a_{i,j} x_j + b_i = y_i - s_i \tag{1}$$

$$z_i = 1 \rightarrow y_i \leq 0 \tag{2}$$

$$z_i = 0 \rightarrow s_i \leq 0 \tag{3}$$

$$y_i \geq 0, s_i \geq 0, z_i \in \{0, 1\} \tag{4}$$

This encoding is used for example in 2.4.2 to generate explanation with guarantees and is *not* used for the training of NNs ([21]).

2.1.2 ∂ILP Learning Explanatory Rules from Noisy Data

In this paper ([22]), the authors attempt to combine inductive logic programming (ILP) (a collection of techniques to construct logic programs from the examples) and neural-network-based systems to produce a differentiable inductive logic programming (∂ILP) architecture.
Advantages of ILP are:

- output is an explicit symbolic structure, that can be understood and verified easier than NNs,
- it is more data-efficient, because small programs are preferred over overfitted long ones,
- it allows for transfer learning as learned programs can be shared.

Disadvantages of ILP and the reasons why we need to move to a differentiable architectures are:

- ILP are not robust to noise and mislabeled data,
- ILP cannot process raw data, only symbolic input.

Generally, in logic programming the central component are if-then rules. They are also called clauses. A **definite clause** is a rule of the form:

$$\alpha \leftarrow \alpha_1, ..., \alpha_m$$

consisting of the head atom (n-ary predicate $p(t_1, ..., t_n)$, where t_i is a term - either variable or constant) α, and the body $\alpha_1, ..., \alpha_m$, where $m \geq 0$. These rules are read from right to left: if all of the atoms on the right are true, then the atom on the left is also true.

Example 2.1
Let us consider the following program that defines connected relation as transitive closure of the edge relation:

$$connected(X, Y) \leftarrow edge(X, Y) \tag{5}$$
$$connected(X, Y) \leftarrow edge(X, Z), connected(Z, Y) \tag{6}$$

The **consequences** of a set R of clauses is generated by repeatedly applying the rules in R until no more consequences can be derived. Formally, the set of immediate consequences $cn_R(X)$ of rules R applied to ground atoms X is defined as:

$$cn_R(X) = X \cup \{\gamma \mid \gamma \leftarrow \gamma_1, ..., \gamma_m \in ground(R), \bigwedge_{i=1}^{m} \gamma_i \in X\},$$

where $ground(R)$ are the ground rules (clauses in which all variables have been substituted by constants) of R.
Now define a series of consequences $C_{R,0}, C_{R,1}, ...$:

$$C_{R,0} = \{\}, C_{R,i+1} = cn_R(C_{R,i}).$$

Then the consequences after T time steps is the union of these series:

$$con(R) = \bigcup_{i \geq 0}^{T} C_{R,i}.$$

Now given a set of clauses R and a ground atom (atom that is grounded - i.e. contains no variables) γ, to chek if $R \models \gamma$, we need to check if $\gamma \in con(R)$. This method is called **forward chaining**.
Given these notations, ∂ILP can be considered as a **binary classification** and divided in the **input data**, **binary output data**, **cross entropy loss**, and the **differentiable function**, which gives the probability of the output.
Input data consists of:

- *axioms* $\mathcal{B}$ - set of ground atoms,

- *language* $\mathcal{L} = (P_e, P_i, arity_e, arity_i, C)$, where P_e, P_i are respectively extensional (are wholly defined by a set of ground atoms. In the example above *edge* is an extensional predicate defined by the set of atoms:

  $$\{edge(a,b), edge(b,c), edge(c,a)\}$$

) and intensional predicates (are defined by a set of clauses. In the example above *connected* is an intensional predicate defined by the clauses:

  $$connected(X,Y) \leftarrow edge(X,Y) \tag{7}$$
  $$connected(X,Y) \leftarrow edge(X,Z), connected(Z,Y) \tag{8}$$

). $arity_e, arity_i$ are the maps that specify the arity of each predicate. C is a set of constants.

- *program template* $\Pi = (P_a, arity_a, rules, T)$ describes the range of programs that can be generated. Here *rules* is a map from each intentional predicate p to a pair of rule templates (τ_p^1, τ_p^2) (w.l.o.g. a pair of clauses defines each intensional predicate) and T is a maximum number of steps in forward chaining inference. Rule template τ describes a range of clauses that can be generated. It is a pair (ν, int) where:
  - $\nu \in \mathbb{N}$ defines a number of existentially quantified variables allowed in the clause,
  - $int \in \{0, 1\}$ specifies whether the atoms in the body of the clause can use intentional predicates ($int = 1$) or only extensional ones ($int = 1$).

- *target atoms* - target intentional predicates, labels for which we are trying to learn.

Output data consists of *true labels* - the correct labels for the target atoms. **Cross entropy loss** is computed as

$$loss = -\mathbb{E}_{(\alpha,\lambda)\sim\Lambda}(\lambda \log p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) + (1-\lambda) log(1 - p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}))).$$

Differentiable function is a conditional probability of the label λ for a ground atom α:

$$p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) = f_{extract}(f_{infer}(f_{convert}(\mathcal{B}), f_{generate}(\Pi, \mathcal{L}), W, T), \alpha),$$

where the parameters are defined as follows:

- function $f_{extract}$ - given the set of all ground atoms G, the function $f_{extract} : [0,1]^n \times G \to [0,1]$ takes a valuation x and an atom γ and extraxts the value for that atom:

  $$f_{extract}(x, \gamma) = x[index(\gamma)],$$

- function $f_{infer} : [0,1]^n \times Cl \times W \times \mathbb{N} \to [0,1]^n$ - is the most important part in the process. It performs T steps of forward-chaining inference using the clause weights $W = \{W_1, ..., W_{|P_i|}\}$ (set of matrices) and generated clauses Cl from $f_{generate}$. Here each matrix $W_p \in \mathbb{R}^{|cl(\tau_p^1)| \times |cl(\tau_p^2)|}$ in the set W is for each intensional predicate $p \in P_i$,
- function $f_{convert} : 2^G \to [0,1]^n$ is a valuation mapping:

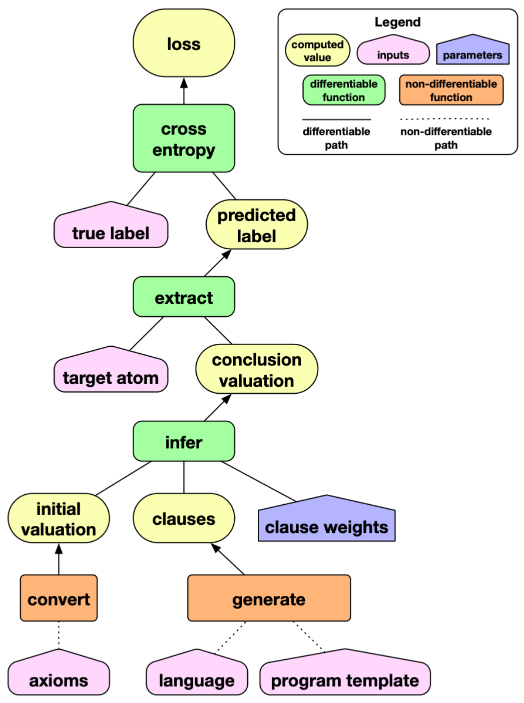
$$f_{convert}(G) = y, \text{ where } y[i] = \begin{cases} 1 \text{ if } \gamma_i \in \mathcal{B} \\ 0 \text{ otherwise} \end{cases}$$

  and where γ_i is the i^{th} ground atom in G for $i \in \{1, ..., n\}$,
- function $f_{generate}$ - produces a set of clauses from a program template Π and a language $\mathcal{L}$:

$$f_{generate}(\Pi, \mathcal{L}) = \{cl(\tau_p^i) \mid p \in P_i, i \in \{1,2\}\},$$

  where $cl(\tau_p^i)$ is a set of clauses that satisfy the template τ_p^i and has the following restrictions:
  - all clauses consist of free variables and no constants,
  - all predicates are of arity $0, 1$, or 2,
  - all clauses w.l.o.g. have exactly two atoms in the body,
  - variables used in the head must be used in the body (safe clauses),
  - the head atom must not appear in the body (no circularity),
  - no clause should be equivalent to any other with the body atoms permuted (no duplicates),
  - the intensional flag *int* must be respected.

This process is schematically represented as follows:



This model achieves state-of-the-art (SOA) performance on 19 out of 20 ILP tasks, taken from four domains: arithmetic, lists, group-theory, and family tree relations. It also performs well the tasks that are very difficult either for ILP or NNs alone, such as learning ”less than” from the pair of pictures.

2.2 Computer vision

2.2.1 Learning by Abstraction: The Neural State Machine

The size and statistical nature of deep learning methods worsens their interpretability and modularity. This paper ([23]) tackles it in the domain of computer vision by introducing an abstraction on top of the deep-learning methods called Neural State Machine (NSM). To show its efficiency the authors apply this concept to the task Visual Question Answering (VQA), which answers the question that involves reasoning over the image — therefore we present theory in context of VQA.
NSM usage, that simulates the computation of a finite automaton, has two stages: *modeling*, to create NSM, and *inference*, to simulate its operation.
First, we describe the modeling part.
Formally, NSM is defined as a tuple $(C, S, E, \{r_i\}_{i=0}^{N}, p_0, \delta)$, where

- C is the model's *concept vocabulary*, embedded as learned vectors. It consists of $L+2$ semantic properties (SP): $C_O = C_0$ is object's identity (e.g. dog, house), $C_A = \bigcup_{i=1}^{L} C_i$ are different types of attributes (e.g.

color, material), and $C_R = C_{L+1}$ are relations (e.g. holding, in front of). They are derived from the Visual Genome dataset [24].

- S are the states represented by the *object nodes* from the image, each together with a collection of discrete probability distributions $\{P_i\}_{i=0}^{L}$ for each of the SP.
  For each $s \in S$, we define a set of property variables $\{s^j\}_{j=0}^{L}$ via:
  $$s^j = \sum_{c_k \in C_j} P_j(k) c_k.$$
  That is, each element in the set is represented via weighted sum of the embedded concepts that describe each of the SP.

- E are the directed *relation edges* that specify valid transitions between the states. Each of the edges is associated with a probability distribution P_{L+1} of its semantic type (e.g. holding, in front of).
  Edge representations are computed similarly to the ones of states (here e' stands for a representation of an edge e):
  $$e' = \sum_{c_k \in C_{L+1}} P_{L+1}(k) c_k \; \forall e \in E.$$

- $\{r_i\}_{i=1}^{N}$ is a sequence of instructions in terms of C translated from the question.
  The translation consists of the following components:

  – tagging, when for each embedded via embedding known as "Global Vectors" (GloVe), which maps words into a meaningful vector space based on the word-word co-occurrence statistics, (dimension $d = 300$) word w_i, we compute a similarity-based distribution:
  $$P_i = \text{softmax}(w_i^T W C),$$
  where W is initially an identity matrix, and C is a matrix that encodes all the concepts plus the additional learned default embedding c' to consider for non-content words.
  Then each word is translated into "normalized" concept-based representation:
  $$v_i = P_i(c') w_i + \sum_{c \in C \setminus \{c'\}} P_i(c) c.$$

  – decoding (based on [25] and [26]), where given a question of P normalized words $V^{P \times d} = \{v_i\}_{i=1}^{P}$, we first apply LSTM encoder on it, and obtain a context vector q to represent the question. Then we extract $N+1$ hidden states from the recurrent decoder $\{h_i\}_{i=0}^{N}$ and transform each of them into a reasoning instruction:
  $$r_i = \text{softmax}(h_i V^T) V.$$

- $p_0 : S \to [0, 1]$ is a probability distribution of the initial state.
- $\delta_{S,E} : p_i \times r_i \to p_{i+1}$ is a *state transition function* represented by some neural module.

Now we describe the inference part.
Formally, at each step i, $\delta_{S,E}$ predicts the distribution p_{i+1} over states given the distribution and the instruction of the current step i. This distribution tells us, which steps to traverse to p_{i+1} given the states we are currently attending p_i. We proceed as follows:

- first, we need to find the instruction type. For this, we compute the distribution $R_i = \text{softmax}(r_i^T D)$ over the $L + 2$ embedded properties D.
- next, we compute for each of the states and edges the relevance score, based on the distribution R_i (here σ is a non-linear function):

$$\gamma_i(s) = \sigma(\sum_{j=0}^{L} R_i(j)(r_j^T W_j s^j)), \tag{9}$$

$$\gamma_i(e) = \sigma(r_i^T W_{L+1} e'). \tag{10}$$

- having relevance scores, we compute the distribution of the next states and edges:

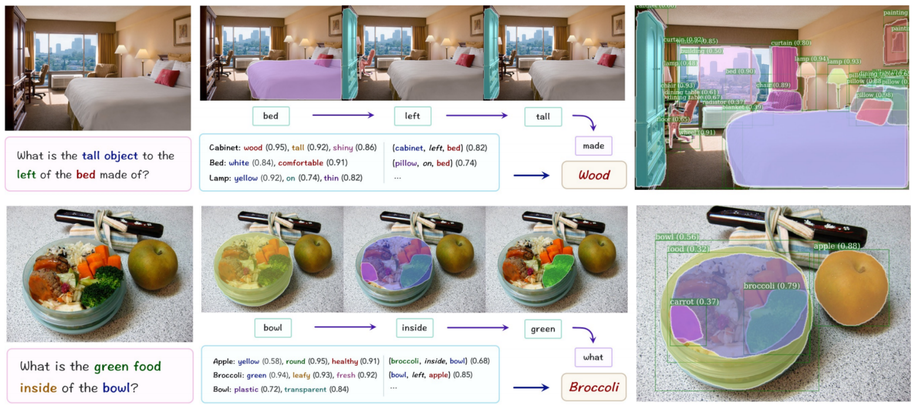
$$p^s_{i+1} = \text{softmax}_{s \in S}(W_s \gamma_i(s)), \tag{11}$$

$$p^r_{i+1} = \text{softmax}_{s \in S}(W_r \sum_{(s',s) \in E} p_i(s')\gamma_i((s', s))), \tag{12}$$

$$p_{i+1} = R_i(L + 1)p^r_{i+1} + (1 - R_i(L + 1))p^s_{i+1}. \tag{13}$$

  The last equation computes the final distribution over the next states and edges as a weighted sum of the distribution of the edges times the similarity of the instruction type with the relation property and the distribution of the states times the similarity of the instruction type with any of the state properties.

- repeating this process for N steps simulates the iterative computation of the NSM. Example can be found below:


What is the tall object to the left of the bed made of?
bed
left
tall
made
Cabinet: wood (0.95), tall (0.92), shiny (0.86)
Bed: white (0.84), comfortable (0.91)
Lamp: yellow (0.92), on (0.74), thin (0.82)
(cabinet, left, bed) (0.82)
(pillow, on, bed) (0.74)
...
Wood
What is the green food inside of the bowl?
bowl
inside
green
what
Apple: yellow (0.58), round (0.95), healthy (0.91)
Broccoli: green (0.94), leafy (0.93), fresh (0.92)
Bowl: plastic (0.72), transparent (0.84)
(broccoli, inside, bowl) (0.68)
(bowl, left, apple) (0.85)
...
Broccoli
bowl (0.56)
food (0.32)
apple (0.88)
broccoli (0.79)
carrot (0.37)

- last step related to the task of VQA consists of concatenating the question context vector q with the vector m that aggregates the information from the last step of the NSM:

$$m = \sum_{s \in S} p_N(s) \sum_{j=0}^{L} (R_N(j) s^j)$$

  and passing it to the 2-layer fully-connected softmax classifier.

  This model achieves state-of-the-art (SOA) performance both on the GQA ([35]) (it tests real-world visual reasoning and compositional question answering) and the VQA-CP ([36]) (a version of the VQA dataset that tests generalization skills via changes in the answer distribution from the training to the test sets) datasets.

2.3 Reinforcement Learning

2.3.1 FSPA - A formal methods approach to interpretable reinforcement learning for robotic planning

This paper ([27]) presents a formal specification language to generate interpretable task descriptions of the steps involved. This work is also very important for safety of RL agents, we will focus however only on the interpretability. First, we will introduce following definitions.

Definition 2.1 (Markov decision process (MDP) and RL)
MDP can be defined as a tuple $\mathcal{M} = (S, A, P, r)$, where

- $S \subseteq \mathbb{R}^n$ *it the state space which can be discrete,*
- $A \subseteq \mathbb{R}^m$ *is the action space which can be discrete,*
- $P : S \times S \times A \to [0, 1]$ *is the transition function and* $P(s_{t+1}|s_t, a_t)$ *specifies the probability of transitioning to the state* s_{t+1} *after being in the state* s_t *and taking action* a_t*,*

- *$r : S \times A \times S \to \mathbb{R}$ is the reward function and $r(s_t, a_t, s_{t+1})$ is the reward of being in the state s_t, executing action a_t and transitioning to the state s_{t+1}.*

The goal of MDP is to find an optimal policy $\pi^* : S \to A$ (or $\pi^* : S \times A \to [0,1]$ for stochastic policies) that maximizes the expected return under all possible policies π:

$$\pi^* = \arg\max_{\pi}(\mathbb{E}^{\pi}\big[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1})\big]),$$

where T is the maximal number of the steps of each execution.
The state-action value function which is often used to evaluate the quality of a policy π is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}^{\pi}\big[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}) \mid s_0 = s, a_0 = a\big].$$

Informally, $Q^{\pi}(s,a)$ is the expected return of choosing action a at the state s and following π onward.
We can say that the RL methods differ in the way they search for π^*, 3 common approaches being:

- *optimizing for Q^{π} and using ϵ-greedy policy (such policy follows greedy action steps with probability $1-\epsilon$ and selects random actions with the probability ϵ). One such algorithm is a Deep Q-Network (DQN) ([29]),*
- *optimizing for π^* directly - direct policy search. One such approach are policy gradient methods ([30]),*
- *alternating between the previous two methods - known as actor-critic methods ([31]).*

In this article authors used an actor-critic method called proximal policy gradient ([32]).

Definition 2.2 (Truncated linear temporal logic (TLTL))
According to [33] (using notations from [27]) a TLTL formula ϕ has the following syntax:

$$\phi := \top \mid f(s) < c \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathcal{F}\phi \mid \mathcal{U}\phi \mid \mathcal{X}\phi$$

and all the operators that can be derived from the TLTL formula:
finite time "always" ($\mathcal{G}\phi = \neg\mathcal{F}\neg\phi$) and "then" ($\phi_1 \mathcal{T} \phi_2 = \phi_1 \wedge \mathcal{X}\mathcal{F}\phi_2$).
In the syntax of TLTL formula above the predicates $f : \mathbb{R}^n \to \mathbb{R}$ are of the form $f(s) < c$ for c constant, standard logical operator are used together with the temporal operators such as "eventually" ($\mathcal{F}$), "until" ($\mathcal{U}$), and "next" ($\mathcal{X}$).

Example 2.2
One valid TLTL formula for $s \in \mathbb{R}$ is $\mathcal{G}(s < 1) \wedge \mathcal{F}(s < 0) \wedge \mathcal{X}(s = -1)$ which reads as: for all the times $s < 1$, eventually $s < 0$ and after this s becomes -1.

Remark 2.1 (TLTL vs linear temporal logic (LTL) ([34]))

- *TLTL is evaluated against finite traces (sequences of valid atomic propositions),*
- *TLTL is specified over predicates of MDP states and LTL - over atomic propositions (express simple known facts about the states).*

Problem 2.1
The task that the paper solves is defined as follows: given (i) a robotic system with states $S \in \mathbb{R}^n$ and actions $A \in \mathbb{R}^m$, (ii) a TLTL specification ϕ_{task} over S, (iii) a set of safety requirements, and (iv) knowledge base (KB) $K = \{\psi_1, ..., \psi_k\}$ with $\phi_K = \mathcal{G} \bigwedge_{i=0}^{k} \psi_i$, generate a trajectory $s_{0:T}$ that satisfies $\phi_{task} \wedge \phi_K$.

As a special type of automaton will be used in the solution, we introduce it first together with the relevant components.

Definition 2.3 (Finite-state predicate automaton (FSPA))
An FSPA is a tuple $\mathcal{A} = (\mathcal{Q}, S, \mathcal{E}, \Psi, q_0, b, F, Tr)$, where:

- *$\mathcal{Q}$ is a finite set of automaton states,*
- *$S \subseteq \mathbb{R}^n$ is a state space,*
- *$\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$ is a set of directed edges,*
- *Ψ is the input alphabet - predicate Boolean formulas, where the predicates are evaluated over S,*
- *$q_0 \in \mathcal{Q}$ is the initial state,*
- *$b : \mathcal{E} \to \Psi$ is a map from the transitions to predicate Boolean formulas,*
- *$F \subseteq \mathcal{Q}$ is the set of final states,*
- *$Tr \subset \mathcal{Q}$ is the set of trap states.*

FSPA functions as follows:
at time 0 automaton is in state q_0, then it evaluates the formulas assigned via b to the edges associated to q_0 at s_0 (the connection between $\mathcal{Q}$ and S will be more clear in the definition below) by calculating robustness $\rho(s_0, b(q_0, q))$ (it quantifies the degree of satisfaction of a formula $b(q_0, q)$ by a trajectory starting at s_0).
The automaton then will choose the edge with the largest robustness and repeat the operation above until it arrives at the state in F.

Remark 2.2
Because every nondeterministic automaton can be translated into a (bigger) deterministic one, we choose a deterministic version of an automaton to avoid the situation, where two edges are satisfied at the same time.
If in the deterministic version, however, several edges will have the same robustness, one of them will be chosen uniformly at random.

The following definition establishes the connection between FSPA with TLTL and RL by guiding the learned policy with the help of FSPA through an automaton state $q \in \mathcal{Q}$.

Definition 2.4 (FSPA-augmented MDP)
Given FSPA $\mathcal{A}$ and MDP $\mathcal{M}$ an FSPA-augmented MDP is defined as:

$$\mathcal{M}_{\mathcal{A}} = (\tilde{S}, A, \tilde{P}, \tilde{r}, \mathcal{E}, \Psi, q_0, b, F, Tr),$$

where:

- $\tilde{S} \subseteq S \times \mathcal{Q}$ *is the product state space,*
- $\tilde{P} : \tilde{S} \times \tilde{S} \times A \to [0,1]$ *is the new transition function defined as:*

  $$\tilde{P}(\tilde{s}_{t+1} \mid \tilde{s}_t, a_t) = \begin{cases} \frac{1}{C} P(s_{t+1} \mid s_t, a_t) \chi_{\{\rho(s_t, b(q_t, q_{t+1}))\}} & (q_t, q_{t+1}) \in \mathcal{E} \\ 0 & \textit{otherwise} \end{cases}$$

  Here C is a normalizing constant.
- $\tilde{r} : \tilde{S} \times A \times \tilde{S} \to \mathbb{R}$ *is the new reward. Define first the set of non-trap FSPA states that are connected to q_t and are not equal to q_t as $\Omega_{q_t} = \{q'_t \mid (q_t, q'_t) \in \mathcal{E}, q'_t \neq q_t, q'_t \notin Tr\}$ and let*

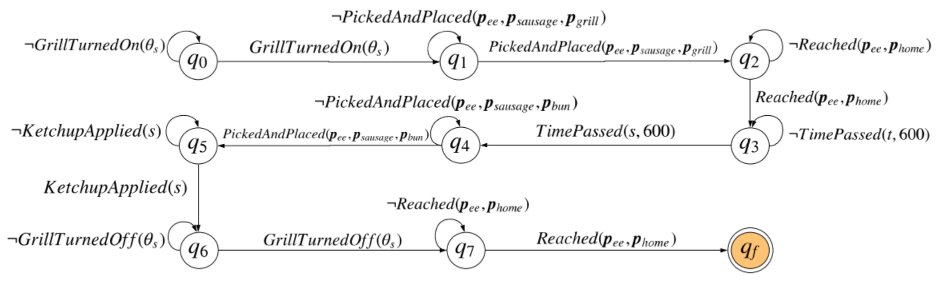
  $$D(q_t) = \vee_{q'_t \in \Omega_{q_t}} b(q_t, q'_t) = \vee_{i=0}^{n} \wedge_{j=0}^{m_i} (\neg) p_{q_t}^{ij}$$

  in its original and DNF forms respectively. Here $(\neg)$ represents a possible negation and $p_{q_t}^{ij}$ is the indexed predicate of $D(q_t)$ in its DNF form.
  Then $\tilde{r}$ is defined as:

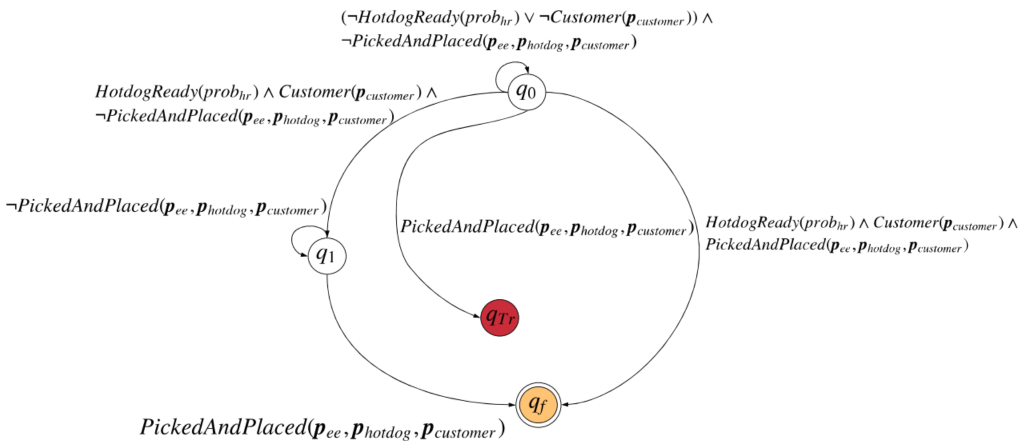
  $$\tilde{r}(\tilde{s}_t, a_t, \tilde{s}_{t+1}) = \begin{cases} \max_{i \in \{0,\dots,n\}} \left[\min_{j \in \{0,\dots,m_i\}} L(p_{q_t}^{ij}) \rho(s_{t+1}, (\neg) p_{q_t}^{ij}) \right] & \exists i,j \textit{ s.t. } L(p_{q_t}^{ij}) = 1 \\ -\|a_t\| & \textit{otherwise} \end{cases}$$

  Here $L : \mathcal{P} \to \{0,1\}$, where $\mathcal{P}$ denotes a set of predicates that are of the form of a bounded function $f : S \to \mathbb{R}_+$, is a labeling function. It assigns 1 to an actionable predicate p ($\exists a_t \in A$ s.t. $\rho(s_{t+1}, p) > \rho(s_t, p)$) and 0 to a non-actionable one.

The *solution*, which is based on the FSPA-augmented MDP, uses the new reward and transition function to move from one state (s_t, q_t) to another (s_{t+1}, q_{t+1}) until reaching one of the final states F (in terms of the second argument of the tuple). In other words it is an RL agent that is being guided by the automaton.



A


(¬HotdogReady(prob_hr) ∨ ¬Customer(p_customer)) ∧
¬PickedAndPlaced(p_ee, p_hotdog, p_customer)
q0
HotdogReady(prob_hr) ∧ Customer(p_customer) ∧
¬PickedAndPlaced(p_ee, p_hotdog, p_customer)
¬PickedAndPlaced(p_ee, p_hotdog, p_customer)
q1
PickedAndPlaced(p_ee, p_hotdog, p_customer)
HotdogReady(prob_hr) ∧ Customer(p_customer) ∧
PickedAndPlaced(p_ee, p_hotdog, p_customer)
q_Tr
q_f
PickedAndPlaced(p_ee, p_hotdog, p_customer)

2.4 Methods that try to generalize

2.4.1 Explanations via adversarial examples

As the first generalization method we consider the work of A. Ignatiev et al. [1] that uses a formula representation $\mathcal{M}$ (e.g., via prime compilation of Boolean formulas [2], or a compilation into a BDD and subsequent extraction of prime implicants and implicates [3]) of a model $\mathbb{M}$.
Here we are essentially using a *MaxSAT* framework, where we consider formulas $\mathcal{F}$ in the form of conjunctive normal form (CNF), in which each clause has an associated *weight*, which allows to model the task of optimization, where some clauses (*hard*) must be satisfied and some may or may not be satisfied (*soft*). A *core* is then as subset of soft clauses, that when combines with hard clauses results in unsatisfiable set of clauses.
For a hard clause C in this setting weight $wt(C) = \infty$ and for a soft one: $wt(C) < \infty$. Further, a *cost* under a *feasible solution* π (truth assignment,

which satisfies all the hard clauses) is a sum of weights of soft clauses from the set of soft clauses S *not satisfied* by π: $cost(\pi) = \sum_{C \in S | \pi \not\models C} wt(C)$. Therefore the task is to find an assignment which minimizes the cost.
Before we state the theorem, let us formalize, what is a *hitting set* according to [4].

Definition 2.5 (Hitting set)
Let K be a set of ***cores****, i.e., a set of subsets of* ***soft clauses****. A* ***hitting set*** *$HS(K)$ of set K is then a set of soft clauses that has a non-empty intersection with every set in $K : \forall C \in K : HS(K) \cap C \neq \emptyset$.*

This work uses a hitting set duality stated in the theorem below.

Theorem 2.1 (Hitting set duality)
Let $\mathbb{M}$ be an ML model, $\mathcal{M}$ – its Boolean formulas representation, and π – a prediction. Then every explanation $\mathcal{E}$ of π breaks every counterexample $\mathcal{C}$ of π and vice versa.

Thus, the set of explanations can be computed as the consistent minimal breaks of the set of the counterexamples.
The algorithm for this is as follows.

Algorithm 1: Computation of explanations as a cost minimal hitting set of the set of counterexamples $\mathbb{C}$

```
Input  : formula $\mathcal{M}$ and prediction $\pi$
Output: set $\mathbb{E}$ of all explanations of prediction $\pi$
$(\mathbb{C}, \mathbb{E}, \mathcal{E}) \leftarrow (\emptyset, \emptyset, \emptyset)$;
do
    if $\mathcal{E} \models (\mathcal{M} \to \pi)$ then
        $\mathbb{E} \leftarrow \mathbb{E} \cup \{\mathcal{E}\}$                # Initially, $\mathcal{E}$ is empty
         # we proceed to the generation of the counterexample
    else
        $(\mathcal{C}, \rho) \leftarrow$ ExtractInstance()
         # Generate an instance $\mathcal{C}$ with a prediction $\rho, \rho \neq \pi$
        for $l \in \mathcal{C}$ do
            if $(\mathcal{C} \backslash \{l\}) \models (\mathcal{M} \to \rho)$ then
                $\mathcal{C} \leftarrow \mathcal{C} \backslash \{l\}$
            end
        end
        $\mathbb{C} \leftarrow \mathbb{C} \cup \mathcal{C}$
    end
    $\mathcal{E} \leftarrow$ MinimumHS($\mathbb{C}$)           # Get a cost-minimal hitting set of $\mathbb{C}$
while $\mathcal{E} \neq \emptyset$;
return $\mathbb{E}$
```

2.4.2 Abduction-Based Explanations for Machine Learning Model

In comparison to many empirical approaches, this paper [16] focuses on the global model-agnostic explanations with the formal guarantees, such as cardinality-

or subset-minimality of the explanations.
In comparison to the model before, we generate the explanations with the guarantee either by top-down or bottom-up approach directly, without using the connection to the counterexamples.
To continue, we need to define an **propositional abduction problem (PAP)** *Abduction*(Γ), a **set of explanations** $\mathcal{S}$ of P, and a **cube** C.

Definition 2.6 (*Abduction*(Γ))
*Given a (finite) constraint language Γ over $\{0,1\}$ (an arbitrary (finite) set of (finitary) relations over $\{0,1\}$) an instance P of a **PAP** is a 4-tuple $(\mathcal{V},\mathcal{H},\mathcal{E},\mathcal{F})$. Here*

- *$\mathcal{V}$ is a finite set of variables,*
- *$\mathcal{H} \subseteq Lits(V)$ is a set of hypotheses,*
- *$\mathcal{E}$ is a propositional formula (the manifestation) with $Vars(\mathcal{E}) \subseteq V$,*
- *$\mathcal{F}$ is theory (a conjuction of constraints) of Γ.*

*The question is whether there exists an **explanation** for P, i.e., a set $\mathcal{S} \subseteq \mathcal{H}$ such that $\mathcal{F} \wedge \bigwedge \mathcal{S}$ is satisfiable (there is a mapping from variables of the formula to $\{0,1\}$ that evaluates formula to 1) and $\mathcal{F} \wedge \bigwedge \mathcal{S} \models \bigwedge \mathcal{E}$*

For subset- and cardinality-minimality or respectively two algorithms are used:

```
Algorithm 2: Computation a subset-minimal explanation
  Input  : formula $\mathcal{F}$, initial cube $C$ and prediction $\mathcal{E}$
  Output: subset-minimal explanation $C_m$
  foreach $l \in C$ do
      if $\mathcal{M}, \mathcal{C}\backslash\{l\} \models (\mathcal{F} \rightarrow \mathcal{E})$ then
          $C \leftarrow C\backslash\{l\}$
      end
  end
  return C
```

```
Algorithm 3: Computation of cardinality-minimal explanation
  Input  : formula $\mathcal{F}$, initial cube $C$ and prediction $\mathcal{E}$
  Output: cardinality-minimal explanation $C_m$
  $\Gamma \leftarrow \emptyset$
  while $true$ do
      $h \leftarrow$ MinimumHS($\Gamma$)
      if $\mathcal{M}, h \models (\mathcal{F} \rightarrow \mathcal{E})$ then
          return $h$
      else
          $\mu \leftarrow$ GetAssignment()
          $C' \leftarrow$ PickFalseLits($C\backslash h, \mu$)
          $\Gamma \leftarrow \Gamma \cup C'$
      end
  end
```

An application to a NN is based on the MILP encoding of the NN.

2.4.3 Markov logic networks (MLN)

It is stated in [11] that "Relational reasoning is a central component of generally intelligent behavior, but has proven difficult for neural networks to learn.", which is what Markov logic networks model.
To formulate the MLN, let us first remind ourselves of Markov networks.

Definition 2.7 (Markov network (a.k.a. Markov random field))
*Let $G = (V, E)$ be an undirected graph, $X = (X_v)_{v \in V}$ - a set of random variables. Then G and X form a Markov network, if they satisfy **global Markov property**:*

$$X_A \perp\!\!\!\perp X_B \mid X_S,$$

which means that if after removing all the nodes from S, there are no paths connecting any node from A to any node from B, then then the sets random variables corresponding to A and B are conditionally independent (CI).

Remark 2.3
In case of the strictly positive joint distribution ($P(X = x) > 0$ for any network configuration x) of X, we have that the global Markov property is equivalent both to the
***pairwise Markov poperty**: for any two not connected nodes $u, v \in V$, the respective random variables X_u, X_v are CI given all other variables:*

$$X_u \perp\!\!\!\perp X_v \mid X_{V \setminus \{u,v\}}$$

and to the
***local Markov poperty**: given the neighbor nodes $N(u)$ of u, the respective random variable X_u is CI given all other variables:*

$$X_u \perp\!\!\!\perp X_{V \setminus (u \cup N(u))} \mid X_{N(u)}.$$

Now we are ready to state a convenient factorization of the joint distribution of X. Note that as we do not have a topological ordering associated with the undirected graph, we cannot use chain rule to represent $P(X = x \mid \theta)$.

Theorem 2.2 (Hammersley–Clifford theorem)
A positive distribution $P(X = x) > 0$ satisfies the above mentioned CI properties of the undirected graph G, iff P can be represented as a product of factors:

$$P(X = x \mid \theta) = \frac{1}{Z(\theta)} \prod_{c \in \mathcal{C}} \psi_c(x_c \mid \theta_c),$$

*where $\mathcal{C}$ is a set of all (maximal) cliques of G, $\psi_c(x_c \mid \theta_c)$ – a **potential function** or **factor** associated to clique c (it can be any non-negative function), and $Z(\theta)$*

is the normalization factor, called ***partition function****, which is given by*

$$Z(\theta) \triangleq \sum_{x \in \mathcal{X}} \prod_{c \in \mathcal{C}} \psi_c(x_c \,|\, \theta_c).$$

Here $\mathcal{X}$ denotes the set of all possible assignments of values to all the network's random variables.

Remark 2.4
Markov networks are often represented as ***log-linear models****:*

$$\log(P(X = x \,|\, \theta)) = \sum_{c \in \mathcal{C}} \phi_c(x_c)^T \theta_c - Z(\theta),$$

where $\phi_c(x_c)$ is a feature vector derived from the values of x_c.

As Markov networks represent a probability distribution over propositional domain with a fixed number of variables, we need to find a way to extend this approach to a **variable number of variables** with **relationships between them**.
To do so, one would use **first-order knowledge base (KB)**. First we represent the KB as a set of clauses, then we attach weights to each of them. These weights are the parameters of the aforementioned Markov network with the clique potentials defined as follows:

$$\exp(\phi_c(x_c)^T w_c),$$

where $\phi_c(x_c)$ is a logical expression, which evaluates clause c applied to the variable x_c, and w_c is the weight that we attach to that clause. w_c specifies the log-probability of the clause: the higher this number is, the greater is the difference between the log-probability of the world that satisfies the formula and the one that violates it.

2.4.4 Probabilistic Logic Neural Networks for Reasoning (pLogicNet)

Our motivation here is that many learning tasks can be expressed via graph representations that serves as a generalization of both sequential (RNN) and grid-based representations (CNN) [14], [15] and has an arbitrary relational inductive bias.
To facilitate inference in GNN and add entity and relationship embeddings, while having access to domain knowledge, M. Qu et al. proposed the pLogicNet [9].
In this papers, the authors focus on the context of probabilistic triplet prediction in the knowledge graph, or more formally:

Definition 2.8 (Probabilistic triplet prediction)
Given a knowledge graph (E, R, O), where E is a set of entities, R – set of relations, O – set of observed triples (a, r, b) with the associated indicator variable

$v_{(a,r,b)}$ which is equal 1 when the triplet is true and 0 otherwise, the task is to predict the hidden triplets $v_H = \{v_{(a,r,b)}\}_{(a,r,b)\in H}$ by reasoning with the observed true triplets $v_O = \{v_{(a,r,b)} = 1\}_{(a,r,b)\in O}$. Eventually, we would like to model the joint distribution of the observed and hidden triplets $p(v_O, v_H)$.

To leverage MLN, we formulate $p(v_O, v_H)$ as follows:

$$p_w(v_O, v_H) = \frac{1}{Z(w)} \exp(\sum_{l\in L} w_l \sum_{g\in G_l} \mathbb{1}_{\{g\ is\ true\}}),$$

where w_l is the weight of the logic rule l between the indicator variables, and G_l is a set of all possible groundings for this rule. An example (more on it can be found in [9]) of such a logic rule is a **subrelation rule**: "a relation r_j is a subrelation of r_i" formally means $\forall x, y \in E,\ v_{(x,r_i,y)} \implies v_{(x,r_j,y)}$.
This model can be trained by the maximization of $\log p_w(v_O)$. This is infeasible, as we have to integrate over all hidden indicator variables v_H. Thus we maximize the following lower bound of it:

$$\log p_w(v_O) \geq \mathcal{L} = \mathbb{E}_{q_\theta(v_H)}(\log p_w(v_O, v_H) - \log(q_\theta(v_H))),$$

where $q_\theta(v_H)$ is a variational distribution of the hidden variables v_H. The equation holds when $q_\theta(v_H) = p_w(v_H \mid v_O)$. To maximize the lower bound we use the EM algorithm. In the E-step, we fix p_w and update $q_\theta(v_H)$ to minimize the KL divergence between $q_\theta(v_H)$ and $p_w(v_H \mid v_O)$. In the M-step, we fix q_θ and update p_w. To leverage knowledge graph embedding, the variational distribution $q_\theta(v_H)$ is approximated as follows:

$$q_\theta(v_H) = \prod_{(a,r,b)\in H} q_\theta(v_{(a,r,b)}) = \prod_{(a,r,b)\in H} Ber(v_{(a,r,b)} \mid f(x_a, x_r, x_b)),$$

where Ber means Bernoulli distribution and f is a scoring function that computes the probability of the triplet (a, r, b) being true. One example of such scoring function is the one used in the model TransE - there it can be formulated as $sig(\gamma - \|x_a + x_r - x_b\|)$.
This model achieves state-of-the-art (SOA) performance for the task of knowledge graph completion on 4 datasets - FB15k, WN18, FB15k-237, WN18RR. More details can be found in the paper [9].

3 Conclusions

References

[1] A. Ignatiev et al. On Relating Explanations and Adversarial Examples. In Advances in Neural Information Processing Systems, 2019.

[2] A. Previti et al. Prime compilation of non-clausal formulae. In IJCAI, pages 1980–1988. AAAI Press, 2015.

[3] A. Shih et al. A symbolic approach to explaining Bayesian network classifiers. In IJCAI, pages 5103–5111, 2018.

[4] K. Fazekas et al. Implicit hitting set algorithms for maximum satisfiability modulo theories. In Proc. IJCAR, volume 10900 of LNCS, 134–151. Springer., 2018.

[5] A. Morgado et al. Iterative and core-guided MaxSAT solving: A survey and assessment. Constraints 18(4), 478–534, 2013.

[6] P. Domingos et al. Unifying logical and statistical AI with markov logic. Commun. ACM, 62(7):74–83,2019.

[7] M. Qu et al. GMNN: Graph markov neural networks. ICML, 2019.

[8] M. Asai. Unsupervised grounding of plannable first-order logic representation from images. arXiv preprint arXiv:1902.08093, 2019.

[9] M. Qu et al. Probabilistic logic neural networks for reasoning. NeurIPS, 2019.

[10] R. B. Palm et al. Recurrent relational networks. NeurIPS, 2018

[11] A. Santoro et al. A simple neural network module for relational reasoning. NeurIPS, 2017.

[12] M. Richardson et al. Markov logic networks. Machine Learning Journal, vol. 62, no. 1-2, pp. 107–136, 2006.

[13] K. Murphy. Machine Learning: A Probabilistic Approach. MIT Press, 2012.

[14] J. Zhou et al. Graph Neural Networks: A Review of Methods and Applications. ArXiv, 2018.

[15] L. Lamb et al. Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective. ArXiv, 2020.

[16] A. Ignatiev et al. Abduction-Based Explanations for Machine Learning Models. AAAI, 2019.

[17] G. Nordh et al. Propositional Abduction is Almost Always Hard. IJCAI, 2019.

[18] G. Nordh et al. What Makes Propositional Abduction Tractable. ACM, 2008.

[19] C. Barett et al. Handbook of Satisfiability. vol. 185 of Frontiers in Artificial Intelligence and Applications, 2009.

[20] N. Narodytska et al. In Search for a SAT-friendly Binarized Neural Network Architecture. ICLR, 2020.

[21] M. Fischetti et al. Deep neural networks and mixed integer linear optimization. Constraints, 2018.

[22] R. Evans et al. Learning explanatory rules from noisy data. JAIR, 2018.

[23] D. Hudson et al. Learning by Abstraction: The Neural State Machine. NeurIPS, 2019.

[24] R. Krishna et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. IJCR, 2017.

[25] D. Hudson et al. Compositional attention networks for machine reasoning. ICLR, 2018.

[26] M. Luong et al. Effective Approaches to Attention-based Neural Machine Translation. EMNLP, 2015.

[27] X. Li et al. A formal methods approach to interpretable reinforcement learning for robotic planning. Science Robotics, 2019.

[28] X. Li et al. Reinforcement learning with temporal logic rewards. IEEE IROS, 2017.

[29] V. Mnih et al. Human-level control through deep reinforcement learning. Nature, 2015.

[30] R. Sutton et al. Policy gradient methods for reinforcement learning with function approximation. NeurIPS, 2000.

[31] T. Degris. Linear off-policy actor-critic. ICLR, 2012.

[32] J. Schulman. Proximal policy optimization algorithms. CoRR, 2017.

[33] X. Li. Reinforcement learning with temporal logic rewards. IEEE/RSJ IROS, 2017.

[34] C. Baier et al. Principles of Model Checking. MIT Press, 2007.

[35] D. Hudson et al. GQA: A new dataset for real-world visual reasoning and compositional question answering. CVPR, 2019.

[36] A. Agrawal et al. Don't just assume; look and answer: Overcoming priors for visual question answering. CVPR, 2018.