EasyChair Preprint

№ 8564

Formal Specification for Learning-Enabled Autonomous Systems

Saddek Bensalem, Chih-Hong Cheng, Xiaowei Huang,
Panagiotis Katsaros, Adam Molin, Dejan Nickovic and Doron Peled

August 3, 2022

Formal Specification for Learning-Enabled Autonomous Systems*

Saddek Bensalem[1], Chih-Hong Cheng[2**], Xiaowei Huang[3], Panagiotis Katsaros[4], Adam Molin[5], Dejan Nickovic[6], and Doron Peled[7]

[1] University Grenoble Alpes, VERIMAG, Grenoble, France
[2] Fraunhofer IKS, Munich, Germany
[3] University of Liverpool, Liverpool, L69 3BX, U.K
[4] Aristotle University of Thessaloniki, Thessaloniki, Greece
[5] DENSO AUTOMOTIVE Deutschland GmbH, Eching, Germany
[6] AIT Austrian Institute of Technology, Vienna, Austria
[7] Bar Ilan University, Ramat Gan, Israel

Abstract. The formal specification provides a uniquely readable description of various aspects of a system, including its temporal behavior. This facilitates testing and sometimes automatic verification of the system against the given specification. We present a logic-based formalism for specifying learning-enabled autonomous systems, which involve components based on neural networks. The formalism is based on first-order past time temporal logic that uses predicates for denoting events. We have applied the formalism successfully to two complex use cases.

Keywords: Learning-enabled systems · Formal specification · Neural networks · First-order LTL.

1 Introduction

The application of formal methods to software artefacts requires the use of formal specification. A specification formalism defines the permitted behaviors or the intended architecture of a system in a uniquely readable manner. It can be used as a contract between different project stakeholders, including the customers, designers, developers and quality assurance teams. Common formalisms include temporal logics and various graph structures or state machines. Different formalisms can be combined together to describe different aspects of the system, such as in UML [16]. In addition, some formalisms, such as state-charts, employ visual notation in order to better demonstrate the specification.

The challenge we are undertaking here is to adopt a formalism that can describe systems with, possibly, timing and cyber-physical components, that are *learning-enabled*, or, in other words, include components that involve neural

* Supported by the european project Horizon 2020 research and innovation programme under grant agreement No. 956123.

** The work is primarily conducted during his service at DENSO.

networks (NNs), trained using deep learning. NNs have strongly impacted on the computer applications in the last decade, including object recognition and natural language processing. The structure of a NN is quite simple: layers of components called "artificial neurons", which have some numerical values, feeding values to the next layer through some linear transformation and then applying an *activation function*, which is a nonlinear transformation. Specifying systems that include components based on NNs is challenging, since a NN has different characteristics from the usual state-transition model.

For example, classifying objects in a picture is commonly performed using a NN. If one considers the values of the individual neurons and the constants used to calculate the transformation between the layers, then the number of possible states is astronomical; moreover, there is no known direct connection between the states and the results associated with the NN. Then it is hardly reasonable to specify directly the connection between the values of the different components of the NN and the classification result, e.g., identifying the object that appears in the picture as a pedestrian or bicycle rider.

Learning-enabled systems appear nowadays in a growing number of applications. This stems from the ability of NNs to provide new identification capabilities, e.g., related to vision and speech recognition. Such systems are often intended to interact with people and the environment, most notably, autonomous driving. This makes these applications highly safety-critical.

We introduce a specification formalism that is based on abstracting away the internal structure of the NN, including the internal values of the different neurons; instead, inspired by [4], our specification asserts about objects that are represented using the NN and related values they stand for. We aim to an intuitive yet expressive formal language, which will match the specification requirements for learning enabled autonomous systems. The adequacy of our formalism has been tested against different requirements for the use cases in the FOCETA EU2020 project[8].

The rest of the paper is structured as follows. Section 2 introduces the syntax and semantics of the specification language. Section 3 presents representative formal specifications from two learning-enabled autonomous systems from the FOCETA project. Section 4 reviews the related work and finally, we provide our concluding remarks, most notably the rationale behind the proposed formalism.

2 Formal Specifications

2.1 Event-Based Abstraction

Given the difficulty of specifying a system that includes a NN based on its set of states, we propose an *event-based* abstraction that hides the details of the NN structure [4]. The specification is defined over *relations* or *predicates* of the form $p(a_1, \ldots, a_n)$ over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$, where for $1 \leq i \leq n$, $a_i \in \mathcal{D}_i$ is a value from the domain $\mathcal{D}_i$. These domains can be, e.g., the integers, the

[8] http://www.foceta-project.eu/

reals or strings. One can also use (state-dependent) Boolean variables, which are degenerate predicates with 0 parameters. Formally, let $p \subseteq \mathcal{D}_{i_1} \times \ldots \times \mathcal{D}_{i_m}$ be a relation over subsets of these domains. An *event-based state* or *eb-state* is a set of tuples from these relations. We can restrict each relation to contain *exactly* or *at most one* tuple, or do not restrict them, depending on the type of system that is modeled. An *execution* is a finite or infinite sequence of eb-states. A *trace* is a finite prefix of an execution.

Examples of some conventions one can adopt for modeling of systems include:

- For runtime-verification, each relation consists of at most one tuple. In some cases, there is only one tuple of one relation in a state.
- Representing real-time can be achieved by equiping each state with a single unary relation $time(t)$, where t corresponds to time.
- The output of an object recognition NN can be the following tuples: $object\_type(ob)$, $accurracy(pr)$, $bounding\_box(x_1, y_1, x_2, y_2)$, where ob is the object type, e.g., 'road-sign', 'car'; $0 \leq pr \leq 1$ is the perceived probability of recognition by the NN, and (x_1, y_1), (x_2, y_2) are the bottom left and top right point of the bounding box around the identified object. Object recognition systems can include multiple objects that are recognized in a single frame.
- One can use different units of measurements when referring to time or other physical components, e.g., distance or energy level.

When modeling a system, the assumed conventions, e.g., the number of tuples per relation allowed in a state and the unit of measurements need to be presented separately from the specification formulas.

Syntax. The formulas of the core logic, which is based on a *first-order* extension [17] of the *past* portion of Linear Temporal Logic (LTL) [19] are defined by the following grammar, where a_i is a constant representing a value in some domain $\mathcal{D}$, and x_i denotes a variable over the same domain $domain(x_i)$. The value of x_i must be from the domain associated with this variable.

$$\varphi ::= true \mid false \mid p(x_1, \ldots, x_n) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid \\ \neg\varphi \mid (\varphi \; S \; \varphi) \mid \ominus \varphi \mid \exists x \; \varphi \mid \forall x \; \varphi \mid e \sim e$$

where $\sim \in \{<, \leq, >, \geq, =, \neq\}$ and $e ::= x \mid a \mid e + e \mid e - e \mid e \times e \mid e \; / \; e$. We read $(\varphi S \psi)$ as φ *since* ψ.

Semantics. Let γ be an assignment to the variables that appear free in a formula φ, with $\gamma(x)$ returning the value of the variable x under the assignment γ. Then $(\gamma, \sigma, i) \models \varphi$ means that φ holds for the assignment γ, and the trace σ of length i. We denote the ith event of σ by $\sigma[i]$. Note that by using past operators, the semantics is not affected by states s_j with $j > i$ that appear in longer prefixes than σ of the execution. Let $vars(\varphi)$ be the set of free variables of a subformula φ (i.e., x is not within the scope of a quantifier $\forall$ or $\exists$, as in $\forall x \, \varphi$, $\exists x \, \varphi$). We denote by $\gamma|_{vars(\varphi)}$ the restriction (projection) of an assignment γ to the free variables appearing in φ.

Let $v(e, \gamma)$ be the value assigned to an expression e under the assignment γ:

- $v(a, \gamma) = a$, when a is a constant.
- $v(x, \gamma) = \gamma(v)$, when x is a varaible.
- $v(e_1 + e_2, \gamma) = v(e_1, \gamma) + v(e_2, \gamma)$, and similarly for '$-$', '$\times$' and '$/$'.

Let ϵ be an empty assignment. In any of the following cases, $(\gamma, \sigma, i) \models \varphi$ is defined when γ is an assignment over $vars(\varphi)$, and $i \geq 1$.

- $(\epsilon, \sigma, i) \models true$.
- $(\gamma, \sigma, i) \models p(y_1, \ldots, y_n)$ if $p(v(y_1), \ldots, v(y_n)) \in \sigma[i]$.
- $(\gamma, \sigma, i) \models (\varphi \wedge \psi)$ if $(\gamma|_{vars(\varphi)}, \sigma, i) \models \varphi$ and $(\gamma|_{vars(\psi)}, \sigma, i) \models \psi$.
- $(\gamma, \sigma, i) \models \neg\varphi$ if not $(\gamma, \sigma, i) \models \varphi$.
- $(\gamma, \sigma, i) \models (\varphi\ S\ \psi)$ if for some $1 \leq j \leq i$, $(\gamma|_{vars(\psi)}, \sigma, j) \models \psi$ and for all $j < k \leq i$, $(\gamma|_{vars(\varphi)}, \sigma, k) \models \varphi$.
- $(\gamma, \sigma, i) \models \ominus\varphi$ if $i > 1$ and $(\gamma, \sigma, i-1) \models \varphi$.
- $(\gamma, \sigma, i) \models \exists x\ \varphi$ if there exists $a \in domain(x)$ such that[9] $(\gamma\,[x \mapsto a], \sigma, i) \models \varphi$.
- $(\gamma, \sigma, i) \models e_1 < e_2$ if $v(e_1, \gamma) < v(e_2, \gamma)$, and similarly for the relations '$\leq$', '$>$', '$\geq$', '$=$' and '$\neq$'.

The rest of the operators are defined as syntactic sugar using the operators defined in the above semantic definitions: $false = \neg true$, $\forall x\varphi = \neg\exists x\neg\varphi$, $(\varphi \vee \psi) = \neg(\neg\varphi \wedge \neg\psi)$. We can also define the following useful operators: $P\varphi = (true\, S\, \varphi)$ (for "Previously") and $H\varphi = (false\, R\, \varphi)$ (for "always in the past").

The specification needs to appear in a context that includes the interpretations of the relations that are used. For clarity, we sometimes denote the domain within the formula during quantification. For example, $\exists x \in Obj\ \varphi$ specifies explicitly the domain name Obj for values of the variable x, which may otherwise be understood from the context where the formula appears.

Intended interpretation. We restrict ourselves to *safety properties* [1], and the interpretation of a formula φ is over *every prefix* of the execution sequence. To emphasize that the interpretation is over all the prefixes, we can use the **G** modality from future LTL, writing $\mathbf{G}\varphi$, where φ is a first-order past LTL formula. There are several reasons for this. First, and foremost, safety properties are most commonly used; in many cases, a non-safety property, which guarantees some progress "eventually" as in the future LTL operator $\Diamond$ [19], without specifying a distinct time, can be replaced with a fixed deadline; then it becomes a safety property. For example, instead of expressing that every *request* is eventually followed by an *acknowledge*, specifying that any request must be followed by an acknowledge *within no more than* $10ms$ is a safety property. In addition, safety properties are often more susceptible to the application of formal methods; a notable example for our context is the ability to perform runtime verification on linear temporal logic with data with specification formalisms similar to the one used here [7, 17].

Dealing with Quantitative Progress of Time Temporal logic specification often abstracts away from using real time, where the intended model uses discrete

[9] $\gamma\,[x \mapsto a]$ is the overriding of γ with the binding $[x \mapsto a]$.

progress between states. We make use of time predicates, in particular, *time* with a single integer or real parameter, which can be part of the events in an execution. For example, the term $time(t1)$ can refer to a timer that reports a value of $t1$. By comparing different values of such terms, one refers to the amount of time elapsed between related events.

Examples of specifications.

- $\forall x\,(closed(x) \rightarrow \ominus(\neg closed(x) S open(x)))$. [For each file, if we closed a file, its the first time we close it since it was opened.]
- $\forall x\,\forall t_1\,\exists t_2\,((t_1 - t_2 < 90 \wedge time(t_1) \wedge closed(x)) \rightarrow \ominus(closed(x) S(open(x) \wedge time(t_2))))$ [Every file, cannot remain opened before it is closed more than 90s. Note that the interpretation of 90 as a meausre of seconds and of $time(t)$ as a predicate that holds if t is the current time is a matter of choice.]
- $\forall t1((time(t1) \wedge \neg stopped(car1)) \rightarrow \neg\exists t2(\neg stopped(car1) S(id(stop\_sign, pr) \wedge time(t2) \wedge pr \geq 0.9 \wedge t1 - t2 > 0.3)))$ [At any time, if *car1* is not stopped, no stop sign has been identified in the last 0.3 seconds with probability ≥ 0.9.]

3 Use Case Specifications from Learning-Enabled Autonomous Systems

In this section, we will show how the specification formalism we proposed allows describing the properties of two challenging use cases:

1. A safe and secure intelligent automated valet parking (AVP) system. This is an L4 autonomous driving system with a fixed Operational Design Domain (ODD) on a given set of parking lots. A user owning a vehicle equipped with the AVP functionality stops the car in front of the parking lot entry area. Whenever the user triggers the AVP function, the vehicle communicates with the infrastructure and parks the car at designated regions (assigned by the infrastructure). The system is expected to operate under mixed traffic, i.e., the parking lot will have other road users including pedestrians and vehicles.
2. A life-critical anaesthetic drug target control infusion system. This use case concerns with the manipulation of hypnotic sedative drugs, and the ability to provide new and breakthrough technology to cope with a better control of sedation status in the patient. Since each patient is unique, no single dosage of an anesthetic is likely to be appropriate for all patients. In addition, providing an under or over-dosage of an anesthetic is highly undesirable. The development of this autonomous controller would facilitate the work of the anaesthesiologists and increase patient safety through better control of the depth of anesthesia. For this development, the verification and validation of the controller prior to any clinical investigation with real patients is essential, so a virtual testbench platform with a complete test plan is required for this.

3.1 Automated Valet Parking System

Object detection Object detection is a key component that aims at recognising objects from sensory input. The sensory input can be understood as a sequence of

single inputs such as images. Usually, a deep learning system (such as YOLO [23]) is applied to return a set of detected objects from a single image, although the result may also depend on the detection results of a sequence of images. We consider an object detection component (ODC), for which there are three major classes of specifications:

1. Functional specifications, concerning whether the object detector exhibits the expected behaviour in normal circumstances on a single image frame.
2. Temporal specifications, for the expected behaviour in sequential inputs.
3. Robustness specifications, concerning whether and how the expected behaviour may be affected by perturbations to the input.

Functional Specifications While there may be various specifications, we consider the following as a typical one:
For every object y in the world, if y is a pedestrian that stands within X meters of range, then the ODC will detect some object z as a pedestrian at almost the same position as y (within ϵ).

This can be expressed as the following formula:

$$\forall y \in Obj\ ((pedestrian(y) \wedge range(y)) \rightarrow detect(y)) \tag{1}$$

where: Obj is assumed to be the set of all *objects* occupying the world (this refers to the ground truth); $pedestrian(y)$ is a predicate that is true iff y is a pedestrian (this refers to the ground truth); $range(y)$ is defined as $distance\_ego(y) \leq X$ where X is the "X meters" parameter of the English spec, and $distance\_ego(y)$ returns the distance of y from the ego vehicle (this also refers to the ground truth); $detect(y)$ is defined as ϕ_2 as follows

$$\exists z \in ODC\_Obj\ (ODC\_pedestrian(z) \wedge |ODC\_position(z), position(y)| \leq \epsilon)$$

where: ODC_Obj is the set of objects detected by the ODC (i.e., "the system" that this spec refers to), $ODC\_pedestrian(z)$ is a predicate which is true iff z is classified as a pedestrian by ODC; $ODC\_position(z)$ is the position of z as returned by ODC; $|a, b|$ is a function that returns the distance between positions a and b; ϵ is a parameter that represents how "close" two positions are.

Temporal Specifications While the specification in (1) considers whether the ODC performs correctly in a single frame of the video stream, it is possible that the overall functionality of the ODC may not be compromised by the failure of a single frame. Therefore, we may consider temporal specifications such as

$$\mathbf{G}\phi_1 \tag{2}$$

Besides, we may consider other temporal specifications such as:
In a sequence of images from a video feed, any object to be detected should not be missed more than 1 in X frames.

This property can be formalised with the following formula:

$$\pi := \mathbf{G}(\neg\phi_1 \rightarrow \bigwedge_{t=1}^{X-1} \ominus^t \phi_1). \tag{3}$$

which, intuitively, guarantees that once there is an incorrect detection at time t, the outputs at previous $X-1$ steps should be all correct.

Robustness Specifications The aforementioned classes of specifications do not consider the possible perturbations to the input. However, perturbations such as benign/natural noises or adversarial/security attacks can be typical to an ODC, which works with natural data. We consider the following specification:
For any input x, which contains a pedestrian y, the detection will be the same within certain perturbation δ with respect to the distance measure d.

This can be expressed as follows (*Input* is the set of possible image frames):

$$\begin{aligned}\mathbf{G}\forall x, x' \in \mathit{Input},\, \exists y, y' \in \mathit{ODC\_Obj}, \mathit{ODC\_pedestrian}(y) \wedge \\ d(x, x') \leq \delta \rightarrow \mathit{pedestrian}(y')\end{aligned} \tag{4}$$

Planning and Control *Planning* refers to the task of making decisions to achieve high-level goals, such as moving the vehicle from a start location (e.g. drop-off space for a parking system) to the goal location, while avoiding obstacles and optimizing over some parameter (e.g. shortest path). *Control* is responsible to execute the actions that have been generated by the higher-level planning tasks and generate the necessary inputs to the autonomous system, in order to realize the desired motions.

Planning is usually further decomposed into *mission planning* and *path* planning. Mission planning represents the highest level decisions regarding the route (sequence of way-points on the map) to be followed, whereas the task of path planning refers to the problem of generating a collision-free trajectory based on the high-level mission plan and the current location of the vehicle.

Mission planning The mission planner must ensure that (i) traffic rules are followed (e.g., wait at stop sign) and (ii) obstacles are avoided.

Traffic rule:
At any time, if ego is not stopped, it is not the case that a red_light was sensed within the last second.

$$\begin{aligned}&\mathbf{G}\forall t_1(\mathit{time}(t_1) \wedge \neg \mathit{stopped}(\mathit{ego})) \rightarrow \\ &\neg\exists t_2(\neg \mathit{stopped}(\mathit{ego})\, S\, \mathit{sensed}(\mathit{red\_light}) \wedge \mathit{time}(t_2) \wedge t_1 - t_2 > 1)\end{aligned} \tag{5}$$

where $\mathit{stopped}(\mathit{ego})$ abstracts the respective signal activated by the mission planner and $\mathit{sensed}(\mathit{red\_light})$ abstracts the output of the perception system.

Collision avoidance:

The planner shall calculate a reference trajectory that keeps a distance d_{ttc} (ttc: time-to-collision) [or d_{safety}] to obstacles, e.g., (1 s) [or (1.0 m).

$$\mathbf{G}\forall(p,v) \in traj_{\text{ref}}, \forall o \in Obj \quad (distance(p,v,position(o)) \geq d_{ttc}) \tag{6}$$

or

$$\mathbf{G}\forall(p,v) \in traj_{\text{ref}}, \forall o \in Obj \quad (|p,position(o)| \geq d_{safety}) \tag{7}$$

where p is the position and v is the velocity of the ego vehicle on a waypoint of the reference trajectory $traj_{\text{ref}}$ (a finite set of way point / velocity tuples) and $position(o)$ is the position of object o.

Path planning involves requirements for the computed path to some (intermediate) goal location that may refer to the current location of the system.

Feasible path to the parking lot:
At time C (some constant), the latest, the parking lot (goal) is reached.

$$\mathbf{G}\neg(time(C) \wedge H\ position(ego) \neq goal) \tag{8}$$

Path constraint:
The path to parking lot follows the center line of the driving lane with max. deviation $dev_{max,st}$ in straight road segments and $dev_{max,cu}$ in curves.

$$\mathbf{G}((straight \rightarrow d \leq dev_{max,st}) \wedge (curve \rightarrow d \leq dev_{max,cu})) \tag{9}$$

with: *straight*, *curve* boolean variables, true iff the road segment is straight (resp. curve); d is the distance from the center of the lane.

Control The controller receives the reference path/trajectory from the path planner, and computes the steering and acceleration/deceleration commands, so that the vehicle moves along the path/trajectory. Vehicle should be kept within its dynamical limits with respect to its velocity, acceleration, jerk steering angle etc.

Vehicle moves along the reference path / trajectory:
The tracked path/trajectory shall not diverge from the reference path / trajectory more than d_{max}, e.g., 0.2 m, for any operating condition defined in the ODD.

$$\mathbf{G}\forall t(time(t) \wedge \mathsf{odd}(\mathsf{in})) \rightarrow (d_{\text{error}}(t) \leq d_{\text{max}}) \tag{10}$$

where: $\mathsf{odd}(\mathsf{in})$ states that the condition in is in the ODD; $d_{\text{error}}(t)$ is the maximal deviation between $path_{\text{controlled}}(t)$ up to time t and the reference path $path_{\text{ref}}$.

The vehicle is within its dynamical limits:
The ego vehicle velocity v shall be bounded by v_{max}, and $v_{max,rev}$ for any operating condition in the ODD.

$$\mathbf{G}(\mathsf{odd}(\mathsf{in}) \rightarrow (-v_{max,rev} \leq v \leq v_{max})) \tag{11}$$

3.2 A medical autonomous system

In this section, we focus on the formalization of requirements, for an anaesthetic drug target control infusion system.

A patient's model is a component that predicts the future patient status of depth of anesthesia (site-effect concentration) based on the drug delivered. A model of the patient helps describing what has happened and what will happen with a planned dose for him/her. For intravenous drugs, the plasma concentration will be determined by the dose given (in weight units of drug), the distribution to different tissues in the body and the elimination from the body.

Let $x \in \mathbb{R}$ be the site effect concentration and

$$L := \{\mathsf{Minimal\text{-}sedation}, \mathsf{Sedation}, \mathsf{Moderate\text{-}sedation}, \mathsf{Deep\text{-}sedation}, \mathsf{General\text{-}anaesthesia}\}$$

be the set of all possible levels in sedation in discretized form. The values "none", "less", "more" denote the amount of medicine to be used.
Formulas (12-14) prescribe the level of injection for the required sedation level:

$$\mathbf{G}\forall l \in L, (sedation\_req(l) \wedge x < low\_level(l)) \rightarrow \mathsf{inject(more)} \tag{12}$$

$$\mathbf{G}\forall l \in L, (sedation\_req(l) \wedge x > upper\_level(l)) \rightarrow \mathsf{inject(none)} \tag{13}$$

$$\mathbf{G}\forall l \in L, sedation\_req(l) \wedge low\_level(l) \leq x \leq upper\_level(l) \rightarrow \mathsf{inject(less)} \tag{14}$$

where: $sedation\_req(l)$ is the required sedation level; $low\_level(l)$ and $upper\_level(l)$ are the lowest (resp. upper) level of drug for sedation level l.

The following formula describes how the site-effect concentration x diminishes over time when (via "inject(none)") not injecting medicine.

$$\mathbf{G}\forall t, t', ((time(t) \wedge concentration(x) \wedge (\mathsf{inject(none)}\ \mathcal{S}\ (time(t') \wedge concentration(x')))) \rightarrow x = x' \times e^{(t'-t)/\tau}) \tag{15}$$

where $concentration(x)$ refers to the anaesthetic drug concentration at the current state and τ is a constant characterizing the speed of decay.

Equation (16) specifies how site-effect concentration x is raised over time for the required sedation level l, while further anaesthetic material is injected.

$$\mathbf{G}\forall l \in L,\ \forall t, t',\ \forall inj\_type,\ ((time(t) \wedge concentration(x) \wedge (inject(inj\_type) \wedge (inj\_type = \mathsf{less} \vee inj\_type = \mathsf{more})\ \mathcal{S}\ (time(t') \wedge concentration(x')))) \rightarrow x = x' + (saturation\_level(l, inj\_type) - x') \times e^{(t'-t)/\tau}) \tag{16}$$

where $saturation\_level(l, inj\_type)$ is the desired saturation level of sedation l when injection is of type inj_type.

4 Related Work

In [28], the formalization of requirements for the runtime verification of an autonomous unmanned aircraft system was based on an extension of propositional LTL [29], where temporal operators are augmented with timing constraints. The Timed Quality Temporal Logic (TQTL) [4] has been proposed for expressing monitorable [27] spatio-temporal quality properties of perception systems based on NNs. TQTL is more limited in scope than our specification formalism, which has also the potential to express system-level requirements. The typed first-order logic [25] is an alternative, whose main difference from traditional first-order logic [26] is the explicit typing of variables, functions and predicates. This allows to reason about the domain of runtime control at the abstract level of types, instead of individual objects, thus reducing the complexity of monitoring.

5 Concluding Remarks

We presented a specification approach and a formalism for learning-enabled autonomous systems. To simplify the approach, yet make it general enough, we adopted the past *first-order* LTL, together with the first-order logic capabilities to use variables, functions, predicates and quantification over the domains of variables. We demonstrated the use of formalism on two safety-critical use cases.

Our approach abstracts away from the internals of the involved NNs, since the actual values of neurons, and the relations between them, are not part of the specification. Instead of these values, inspired by [4], we refer to the values and objects they represent. Research on how a NN actually maintains its ability to classify objects or perform other typical tasks is still undergoing, therefore abstracting away from it is a useful feature rather than a handicap.

A notable tradeoff in selecting the specification formalism exists (often related also to the model of execution) between the expressiveness and the ability to utilize it within different formal methods: testing, automatic (model-checking) and manual (theorem-proving) verification and monitoring (i.e., runtime verification). An important case of gaining decidability for scarifying the generality of the model and the formalism is that of the ability to perform automatic verification for propositional LTL (also for Computational Tree Logic CTL) of *finite state systems* [10, 21]. Models for automatic verification hence often abstract the states as a Boolean combination. This helps achieving decidability and also taming down the complexity. Nevertheless, for actual systems, it is often desired to include data in both the specification and the model. For cyber-physical systems, and for learning-enabled autonomous systems in particular, the use of data and parametric specification are often essential. While comprehensive automatic verification needs then to be abandoned, it is still desired to apply testing and monitoring. These methods provide a weaker guarantee for correctness, but are still highly important in the system development process. The formalism that we proposed has the advantage to allow testing and runtime verification [7, 17].

A common constraint on specification that we undertook is focusing on safety properties. Essentially, safety properties assert that "something bad never happens", whereas liveness properties assert, intuitively, that "something good will happen". A formal definition and proof that every property of linear execution sequences can be expressed as a conjunction of a safety and a liveness property appears in [1]. While temporal logic allows expressing safety and liveness properties, recently, there is a growing concentration on safety properties, abandoning liveness. One reason is that safety properties are *monitorable* in the sense that their violation can be detected within finite time. On the other hand, liveness properties may often be non-monitorable [8]: the fact that something "good" will happen can exist forever, not violated, yet that event or combination of states may be deferred forever. It turns out that automatic testability and monitorability for execution sequences with data exist for the kind of specification suggested here [7, 17]. Furthermore, for cyber-physical systems, the requirement often involves setting some actual deadlines: the "good" thing to happen must occur within some given time (physical time or some virtual units of progress). Then, the property becomes a safety property: when that time is expired, a failure of the property happens, and by keeping up with the progress of time, one can monitor the failure after a finite amount of time.

References

1. B. Alpern, F. B. Schneider, Recognizing safety and liveness. Distributed Computing 2(3): 117-126, 1987.
2. M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, U. Topcu, Safe Reinforcement Learning via Shielding. AAAI 2018: 2669-2678
3. K. R. Apt, D. Kozen, Limits for Automatic Verification of Finite-State Concurrent Systems. Information Processing Letters 22(6), 307-309 (1986).
4. A. Balakrishnan, A. G. Puranic, X. Qin, A. Dokhanchi, J. V. Deshmukh, H. Ben Amor, G. Fainekos, Specifying and Evaluating Quality Metrics for Vision-based Perception Systems, DATE 2019, 1433-1438.
5. E. Bartocci, R. Bloem, B. Maderbacher, N. Manjunath, D. Nickovic: Adaptive Testing for CPS with Specification Coverage. ADHS 2021
6. E. Bartocci, Y. Falcone, A. Francalanza, G. Reger: Introduction to Runtime Verification. Lectures on Runtime Verification 2018: 1-33
7. D. A. Basin, F. Klaedtke, S. Müller, E. Zalinescu, Monitoring Metric First-Order Temporal Properties, Journal of the ACM 62(2), 45, 2015.
8. A. Bauer, M. Leucker, C. Schallhart, The good, the bad, and the ugly, but how ugly is ugly?, RV'07, LNCS Volume 4839, Springer, 126-138, 2007.
9. R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Könighofer, M. Roveri, V. Schuppan, R. Seeber: RATSY - A New Requirements Analysis Tool with Synthesis. CAV 2010: 425-429
10. E. M. Clarke, E. A. Emerson, Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic, Logic of Programs 1981, 52-71.
11. M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

12. A. Donzé, O. Maler: Robust Satisfaction of Temporal Logic over Real-Valued Signals. FORMATS 2010: 92-106
13. G. Fainekos, G. Pappas: Robustness of Temporal Logic Specifications. FATES/RV 2006: 178-192
14. Y. Falcone, L. Mounier, J. -C. Fernandez, J. -L. Richier: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods Syst. Des. 38(3): 223-262 (2011)
15. T. Ferrère, D. Nickovic, A. Donzé, H. Ito, J. Kapinski: Interface-aware signal temporal logic. HSCC 2019: 57-66
16. M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison–Wesley.
17. K. Havelund, D. Peled, D. Ulus, First order temporal logic monitoring with BDDs, FMCAD 2017, 116-123.
18. H. S. Hong, I. Lee, O. Sokolsky, H. Ural: A Temporal Logic Based Theory of Test Coverage and Generation. TACAS 2002: 327-341
19. Z. Manna, A. Pnueli, Completing the Temporal Picture, Theoretical Computer Science 83, 91-130, 1991.
20. T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancic, A. Gupta, G. Pappas: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. HSCC 2010: 211-220
21. J. -P. Queille, J. Sifakis, Specification and verification of concurrent systems in CESAR. Symposium on Programming 1982, 337-351.
22. P. Prabhakar, R. Lal, J. Kapinski: Automatic Trace Generation for Signal Temporal Logic. RTSS 2018: 208-217.
23. J. Redmon, S. Divvala, R. Girshick, A. Farhadi: You only look once: Unified, real-time object detection. CVPR 2016: 779-788.
24. H. Roehm, T. Heinz, E. C. Mayer: STLInspector: STL Validation with Guarantees. CAV (1) 2017: 225-232
25. B. Beckert, R. Hähnle, P. H. Schmitt: Verification of Object-Oriented Software. The KeY Approach. Lecture Notes in Computer Science 4334, Springer, 2007
26. R. R. Smullyan: First-Order Logic. Ergebnisse der Mathematik und ihrer Grenzgebiete. 2. Folge, Springer Science & Business Media, 2012
27. A. Balakrishnan, J. Deshmukh, B. Hoxha, T. Yamaguchi, G. Fainekos: PerceMon: Online Monitoring for Perception Systems. RV 2001: 297-308
28. A. Dutle, C. A. Muñoz, E. Conrad. A. Goodloe, L. Titolo, I. Perez, S. Balachandran, D. Giannakopoulou, A. Mavridou, T. Pressburger: From Requirements to Autonomous Flight: An Overview of the Monitoring ICAROUS Project, Proc. of 2nd Workshop on Formal Methods for Autonomous Systems (FMAS), EPTCS, Vol. 329, 2020: 23-30
29. R. Koymans: Specifying real-time properties with metric temporal logic, Real-Time Systems. 2 (4): 255-299 (1990)