Monitoring of Spatio-Temporal Properties with Nonlinear SAT Solvers

André de Matos Pedro[1(✉)], Tomás Silva[1,2], Tiago Sequeira[1], João Lourenço[2], João Costa Seco[2], and Carla Ferreira[2]

[1] VORTEX-CoLab, Vila Nova de Gaia, Portugal
andre.pedro@vortex-colab.com
[2] NOVA-LINCS, NOVA University Lisbon, Lisbon, Portugal

Abstract. The automotive industry is increasingly dependent on computing systems with variable levels of critical requirements. The verification and validation methods for these systems are now leveraging complex AI methods, for which the decision algorithms introduce non-determinism, especially in autonomous driving. This paper presents a runtime verification technique agnostic to the target system, which focuses on monitoring spatio-temporal properties that abstract the evolution of objects' behavior in their spatial and temporal flow. First, a formalization of three known traffic rules (from the Vienna convention on road traffic) is presented, where a spatio-temporal logic fragment is used. Then, these logical expressions are translated to a monitoring model written in the first-order logic, where they will be processed by a non-linear satisfiability solver. Finally, the translation allows the solver to check the validity of the encoded properties according to an instance of a specific traffic scenario (a trace). The results obtained from our tool that automatically generates a monitor from a formula show that our approach is feasible for online monitoring in a real-world environment.

1 Introduction

Autonomous Driving System (ADS) is a field of study that belongs to the Cyber-Physical Systems (CPSs) domain, partially seen as safety-critical systems due to the high impact that a hazard can have [27]. Correctness and validation of an ADS are crucial, as any error or malfunction of the system may lead to loss of life, environmental damage, or financial impact on trust and reputation [22]. Challenges on the verification and validation methodologies for these systems are being introduced by sub-symbolic AI methods, for which the decision algorithms are known to introduce non-determinism [2,6,7,15].

Runtime Verification (RV) is a lightweight verification method commonly used in safety-critical systems [16,19,30] performed during runtime, which offers the possibility to act whenever a fault is observed. In RV, a formal requirement is used to automatically generate a monitor that checks if the target system is compliant with it. In this paper, we are interested in formally representing how ADSs interact with the environment, hence, we use Linear Temporal Logic

(LTL), a tool widely used in RV [16], to describe the evolution over time, and Modal Metric Spaces (MS), which allows us to formally reason about the surrounding space of the system [18]. By combining these two logical frameworks, we enable a full description of the ADS in space at all time instants.

The traffic safety rules that driving systems, and more specifically ADS, are subjected to, usually specify temporal and spatial features. The spatio-temporal languages (e.g., [12,14,20]) provide the adequate formalization and fulfillment of the ADS requirements [24], which are specified over time and space. In the present work, we consider the safety requirements of an ADS to be expressed by sets of spatial constraints along a discrete linear time frame.

This paper proposes an RV approach that can deal with different autonomous systems and focus on the monitoring of their spatio-temporal properties. These properties are safety requirements that represent road safety constraints over objects that are specified by their distances or topological relations. From a macro perspective, Fig. 1 schematizes our architecture, where the relations between simulator, monitor and vehicle can be seen. The simulator implements the scenario described using ASAM standard [11] and the Ego vehicle implements the set of requirements. Then the Monitor Block that runs a solver checks whether the requirement is met and draws a verdict. Step 1 starts with the formalization of the requirements. From a micro perspective, the verification of a *LTL combined with a fragment of MS* (LTL $\times$ MS) [12] formula consists on the construction of a monitoring model and a decision procedure. Given a trace (step 4) that comes from the ADS, the decision procedure inside the Monitor Block answers whether a trace satisfies the monitoring model (step 5) and draws a verdict (step 6). As shown in Fig. 1, the scenario (step 3) and the corresponding formalized traffic rules (step 2) are given as input to the Translation and Model Construction, where the translation to a set of *first order language of the real numbers* ($\text{FOL}_{\mathbb{R}}$) constraints is done. This engine creates a monitoring model in $\text{FOL}_{\mathbb{R}}$, which is interpreted by the non-linear satisfiability solver that is provided by the SMT solver Z3 [8] and runs inside the Monitor Block. Parallel to the monitoring model, a trace at runtime feeds the Monitor Block, and a Trace Encoder is provided to encode it to $\text{FOL}_{\mathbb{R}}$. So, the monitor block can produce a verdict based on a trace that came from the ADS, a scenario, and a requirement.

Problem Statement. Consider monitoring the behaviour of an ADS, while driving at an urban intersection, that must comply with road safety rules defined by the international Vienna convention [24]. The present work focuses on presenting a logic fragment, expressive enough to describe a specific set of road traffic rules. Thanks to this fragment we were able to build an inline monitor that verifies if these legal requirements [24] are being met. In simple terms the road safety requirement '*the car shall stop when it reaches a stop sign and then carries on when the path is clear*' is a spatio-temporal property. When encoded as a $\text{FOL}_{\mathbb{R}}$ formula, nonlinear SAT solvers are able to verify its satisfiability.

Paper Contributions. First, we present a formalization of three traffic rules, taken from the Vienna convention [24] using LTL $\times$ MS, and applied to the

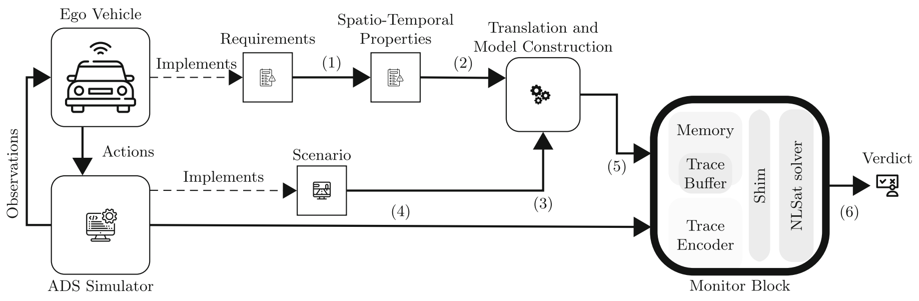


Fig. 1. Ego vehicle spatio-temporal monitoring architecture.

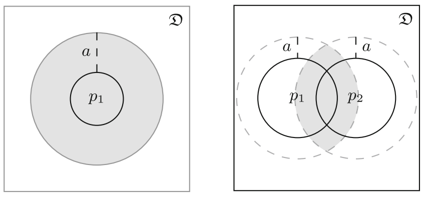
context of these three traffic rules, the construction of the traffic T-shaped junction scenario. Second, we encoded these rules written in LTL × MS and our scenario in $\mathrm{FOL}_{\mathbb{R}}$, the language interpretable by the SMT solver Z3 [8]. Then, to encapsulate the encoding, our tool automatically generates runtime monitor blocks that can verify whether the requirements check in the simulated environment. Finally, we show evidence of the feasibility and scalability of online monitoring.

Paper Structure. Section 2 introduces some important concepts and definitions of the LTL × MS language. Section 3 presents the formalization of three road traffic rules in terms of LTL × MS and a T-shaped traffic junction, where the aforementioned rules are applicable. Moreover, the scenario is abstracted to $\mathrm{FOL}_{\mathbb{R}}$ and the trace is introduced as well as its encoding to $\mathrm{FOL}_{\mathbb{R}}$. Section 4 introduces the monitor generation approach, while Sect. 5 shows the feasibility of the monitor approach. Finally, Sects. 6 and 7 present the related work and draw conclusions and directions for future work, respectively.

2 Preliminaries

The combination of temporal logic with spatial logic has been exhaustively explored [1,12,13,23]. LTL is a propositional discrete linear temporal logic, adequate for model checking of reactive systems and RV [19]. The time flow in LTL is a set of points that are strictly ordered by the precedence relation $<$ [10], and is restricted to the usage of propositions and how they are sequenced. Furthermore, LTL has the temporal operators 'Until', $\alpha\mathbf{U}\omega$—α has to hold until ω becomes true—and 'Since', $\alpha\mathbf{S}\omega$—α has been true since ω was true.

Regarding the spatial logic, Kuts et al. [18] introduced MS, which includes the bounded distance operators: $\exists^{=a}$, $\exists^{<a}$, $\exists^{>a}$, and $\exists^{<b}_{>a}$. As an example, Fig. 2 gives a visual description of $\exists^{\leq a}p_1$ and $\exists^{\leq a}(p_1 \sqcap p_2)$ in a metric space, where p_1, p_2 are spatial variables, expanded by a units. Wolter and Zakharyaschev [33] presented a restricted version named $\mathrm{MS}^{\leq,<}$ that just considers the operators $\exists^{\leq,<}$. Marco et al. [1, p. 545] showed that the satisfiability and the computational complexity of the combination of LTL with $\mathrm{MS}^{\leq}$ is decidable. However, despite the expressiveness of LTL × MS, decision procedures for spatio-temporal



(a) Distance operator $(\exists^{\leq a} p_1)$ (b) $\exists^{\leq a} (p_1 \sqcap p_2)$

Fig. 2. Examples of distance term operators on a metric space $\mathfrak{D}$.

languages are scarce [14]. As far as we know, in this paper we introduce the first decision procedure for LTL × MS.

Definition 1 (LTL × MS - Syntax). *The terms and formulas are inductively defined by*

$$\varrho ::= p \mid \overline{\varrho} \mid \varrho_1 \sqcap \varrho_2 \mid \varrho_1 \sqcup \varrho_2 \mid \exists^{\leq a} \varrho \mid \varrho_1 \mathfrak{U} \varrho_2 \qquad \textit{(terms)}$$
$$\varphi ::= \varrho_1 \sqsubseteq \varrho_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \varphi_1 \,\mathcal{S}\, \varphi_2, \qquad \textit{(formulas)}$$

where $p \in \mathfrak{P}$ is a spatial variable (or proposition), a is a rational number (distance), and $\mathfrak{P}$ a nonempty set of variables. $\mathfrak{U}$ and $\mathcal{U}$ stand for the binary operator 'Until' for terms and formulas, respectively. While $\mathcal{S}$ is the 'Since' operator. Moreover, ρ is denoted as an instance of ϱ and ϕ as an instance of φ.

Definition 2 (LTL × MS - Terms Semantics). *A metric temporal model is a pair of the form $\mathfrak{M} = (\mathfrak{D}, \mathfrak{N})$ [1], where $\mathfrak{D} = (\Delta, d)$ is a metric space, Δ represents a nonempty set of points that reproduce the entire universe, d is a function of the form $\Delta \times \Delta$ describing the distance between every two points in Δ, satisfying the axioms identity of indiscernibles, symmetry and triangle inequality [17]. The valuation $\mathfrak{N}$ is a map associating each spatial variable p and time instant n to a set $\mathfrak{N}(p, n) \subseteq \Delta$. The valuation can be inductively extended to arbitrary LTL × MS terms such as*

$$\begin{aligned}
\mathfrak{N}(\overline{\varrho}, n) &= \Delta - \mathfrak{N}(\varrho, n),\\
\mathfrak{N}(\varrho_1 \sqcap \varrho_2, n) &= \mathfrak{N}(\varrho_1, n) \cap \mathfrak{N}(\varrho_2, n),\\
\mathfrak{N}(\exists^{\leq a} \varrho, n) &= \{x \in \Delta \mid \text{there exists a } y \in \mathfrak{N}(\varrho, n) \text{ such that } d(x, y) \leq a\},\\
\mathfrak{N}(\varrho_1 \mathfrak{U} \varrho_2, n) &= \bigcup_{m>n} \left(\mathfrak{N}(\varrho_2, m) \cap \bigcap_{k \in]n,m[} \mathfrak{N}(\varrho_1, k) \right)
\end{aligned}$$

The shorthands 'Eventually' $\diamondsuit$, 'Always' $\boxdot$, and 'Next' $\odot$ are defined using $\mathfrak{U}$, $\diamondsuit\, \varrho \equiv \top \mathfrak{U}\, \varrho$, $\boxdot \varrho \equiv \overline{\diamondsuit \overline{\varrho}}$, and $\odot\, \varrho \equiv \bot \mathfrak{U}\, \varrho$, where $\top$ and $\bot$ denote the universe and the empty set. $\odot$ is the next operator and its semantics is $\mathfrak{N}(\odot, n) = \mathfrak{N}(\varrho, n+1)$, while $\diamondsuit$ stands for the eventually operator with $\mathfrak{N}(\diamondsuit, n) = \bigcup_{m>n} \mathfrak{N}(\varrho, m)$, and $\boxdot$ means the always operator where $\mathfrak{N}(\boxdot, n) = \bigcap_{m>n} \mathfrak{N}(\varrho, m)$.

Definition 3 (LTL $\times$ MS - Formulas Semantics [12]**).** *An LTL $\times$ MS formula φ is said satisfiable if there exists a model $\mathfrak{M}$ such that $(\mathfrak{M}, n) \models \varphi$ for some time point $n \in \mathbb{N}$. $\mathfrak{M}$ is equipped with the following properties*

$$
\begin{array}{lll}
(\mathfrak{M}, n) \models \varrho_1 \sqsubseteq \varrho_2 & \textit{iff} & \mathfrak{N}(\varrho_1, n) \subseteq \mathfrak{N}(\varrho_2, n), \\
(\mathfrak{M}, n) \models \neg\varphi & \textit{iff} & (\mathfrak{M}, n) \not\models \varphi, \\
(\mathfrak{M}, n) \models \varphi_1 \wedge \varphi_2 & \textit{iff} & (\mathfrak{M}, n) \models \varphi_1 \quad \text{and} \quad (\mathfrak{M}, n) \models \varphi_2, \\
(\mathfrak{M}, n) \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \textit{iff} & \text{there is a } m > n \text{ such that } (\mathfrak{M}, m) \models \varphi_2 \text{ and} \\
 & & (\mathfrak{M}, k) \models \varphi_1 \text{ for all } k \in (n, m), \\
(\mathfrak{M}, n) \models \varphi_1 \,\mathcal{S}\, \varphi_2 & \textit{iff} & \text{there is a } m < n \text{ such that } (\mathfrak{M}, m) \models \varphi_2 \textit{ and} \\
 & & (\mathfrak{M}, k) \models \varphi_1 \text{ for all } k \in (n, m).
\end{array}
$$

Regarding temporal modalities, $\Diamond$ stands for ‘Eventually’, $\Box$ for ‘Always’ and $\bigcirc$ for ‘Next’, which can be defined using $\mathcal{U}$: $\Diamond\varphi \equiv \top\,\mathcal{U}\,\varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$ and $\bigcirc\varphi \equiv \bot\,\mathcal{U}\,\varphi$. When talking about past, the connectors are defined in an analogous way using $\mathcal{S}$. Thus, $\diamondminus\varphi \equiv \top\,\mathcal{S}\,\varphi$ for ‘Once’, $\boxminus\varphi \equiv \neg\diamondminus\neg\varphi$ for ‘Historically’ and $\ominus\varphi \equiv \bot\,\mathcal{S}\,\varphi$ for ‘Yesterday’. Note that the traditional universal modalities $\forall$ and $\exists$ are expressible in our language. $\forall\varrho$ can be seen as an abbreviation for $\top \sqsubseteq \varrho$ and $\exists\varrho$ for $\neg(\varrho \sqsubseteq \bot)$. Along our work we will use the symbol := to denote ‘is defined’. Also, to construct complex formulas we introduce four spatial patterns over terms ρ_1, ρ_2, where the atomic formula $\varrho_1 = \varrho_2$ stands for $(\varrho_1 \sqsubseteq \varrho_2) \wedge (\varrho_2 \sqsubseteq \varrho_1)$, as follows:

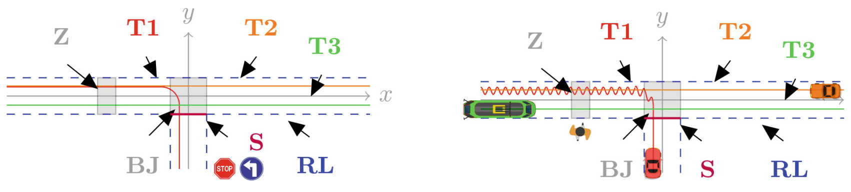
$$
\begin{array}{lll}
\mathtt{DC}\,(\rho_1, \rho_2) & := & \rho_1 \sqcap \rho_2 = \bot, \ \text{(disconnected)} \\
\mathtt{EQ}\,(\rho_1, \rho_2) & := & (\rho_1 \sqsubseteq \rho_2) \wedge (\rho_2 \sqsubseteq \rho_1), \ \text{(equally connected)} \\
\mathtt{O}\,(\rho_1, \rho_2) & := & \neg\big(\mathtt{DC}\,(\rho_1, \rho_2)\big) \wedge \neg(\rho_1 \sqsubseteq \rho_2) \wedge \neg(\rho_2 \sqsubseteq \rho_1), \ \text{(overlapped)} \\
\mathtt{I}\,(\rho_1, \rho_2) & := & (\rho_1 \sqsubseteq \rho_2) \wedge \neg(\rho_2 \sqsubseteq \rho_1). \ \text{(included)}
\end{array}
$$

Encoding Language FOL$_{\mathbb{R}}$

The FOL$_{\mathbb{R}}$ denotes the first-order logic defined over the structure $(\mathbb{R}, <, +, \times, 1, 0)$ that consists of the set of all well-formed sentences of first-order logic that involve quantifiers and logical combinations of polynomial expressions over real variables. The first-order language FOL$_{\mathbb{R}}$ forms the set $\mathbb{L}$, and $\mathbb{P}$ means the set of real variables in FOL$_{\mathbb{R}}$.

3 Running Example

The concrete traffic scenario studied throughout this work is depicted in Fig. 3a. It consists of a T-shaped junction where the vehicle **C**, from a one-way road, approaches the intersection where faces a stop sign in order to enter a bi-directional road. In this road there is a tram going one way in its rails named as *Tram* and a car going the other way identified as **C′**. In the junction of these roads there is a box junction that, according to the Vienna convention on road



(a) Model with all reference trajectories and without actors.

(b) Unsatisfiable model with actors and the oscillating Ego vehicle trajectory.

Fig. 3. Running Example: An urban T-shaped junction scenario.

traffic, is an area where it is prohibited to stop. In addition, there is a pedestrian zebra crossing in the bi-directional road. It is also possible to see three different solid lines noted as **T1** (red), **T2** (orange), and **T3** (green) that represent the reference trajectories the vehicles may take in this specific use case.

The goal of this running example is to provide validation for complex Ego vehicles. To this end, we start by introducing the formalization of the traffic rules in LTL $\times$ MS, the encoding of the traffic scenario, and later the trace definition and encoding. Note that a scenario describes static objects while a trace describes dynamic objects that live within a scenario. Objects are entities such as pedestrians, cyclists, vehicles, trajectories, or horizontal/vertical traffic signs (e.g., crosswalk, stop sign).

Informally, an Ego vehicle shall follow a reference trajectory when at a cross region with a safety-margin of at least one meter. In LTL $\times$ MS, we write

$$\Box\Big(\mathsf{0}\,(\mathbf{T1}, \exists^{\leq 1}\mathbf{C})\Big), \tag{1}$$

where **T1** corresponds to the reference trajectory, and **C** to the Ego vehicle. The model in Fig. 3b does not satisfy (1) since the oscillation of the Ego vehicle along the reference trajectory **T1** is above the accepted threshold of one meter.

3.1 Formalization of Road Traffic Rules with LTL × MS

According to the Vienna convention [24], road traffic rules describe the way in which pedestrians and vehicles should behave in a street environment. Without loss of generality, we identify three specific rules of interest to describe in LTL $\times$ MS language. These rules translate general autonomous driving system safety requirements to check a given scenario.

Rule 1 (vehicle safety-margin). *To simplify the presentation, this rule is divided into two parts: (a) a vehicle should maintain a safety-margin relative to the walkways (based on article 13 [24]) while following its trajectory, and (b) a vehicle should maintain a safety-margin from the vehicle in front of it. In LTL $\times$ MS, the (a) part of this rule can be described by*

$$\neg\Diamond\Big(\mathsf{0}\,(\mathbf{RL}, \exists^{\leq 1}\mathbf{C})\Big), \tag{2}$$

where **RL** *means the road limits. Informally, it reads as the vehicle* **C** *should maintain a safety-margin of at least one meter ($\exists^{\leq 1}\mathbf{C}$) between the car and the road limit, while following its predefined trajectory. Moreover* (2) *can be written in terms of temporal connectors and predicates, by expanding* $\mathtt{O}$ *and* $\square$, *we arrive to the following expression:*

$$\neg\Big[\top\,\mathcal{U}\,\Big(\neg\big(\mathbf{RL}\sqcap(\exists^{\leq 1}\mathbf{C})=\bot\big)\wedge\neg\big(\mathbf{RL}\sqsubseteq\exists^{\leq 1}\mathbf{C}\big)\wedge\neg\big((\exists^{\leq 1}\mathbf{C})\sqsubseteq\mathbf{RL}\big)\Big)\Big]. \quad (3)$$

The safety-margin (b) of at least two meters between two vehicles, can be expressed as:

$$\neg\Diamond\left(\mathtt{O}\left(\exists^{\leq 2}\mathbf{C}',\exists^{\leq 2}\mathbf{C}\right)\right), \quad (4)$$

where $\mathbf{C}'$ *corresponds to an external car. The overall rule is the conjunction of formulas* (2) *and* (4). *The second term of the conjunction is transformed in*

$$\neg\Big[\top\,\mathcal{U}\,\Big(\neg\big(\exists^{\leq 2}\mathbf{C}'\sqcap\exists^{\leq 2}\mathbf{C}=\bot\big)\wedge\neg\big(\exists^{\leq 2}\mathbf{C}'\sqsubseteq\exists^{\leq 2}\mathbf{C}\big)\wedge\neg\big(\exists^{\leq 2}\mathbf{C}\sqsubseteq\exists^{\leq 2}\mathbf{C}'\big)\Big)\Big]. \quad (5)$$

Rule 2 (stop-on-forbidden areas). *A vehicle should not stop on top of (a) a box junction, based on the Portuguese road marks M17b and article 18 of the Vienna convention; (b) a crosswalk, based on article 23 al.3 [24]; (c) tram rails, based on article 23 al.3 [24].*

Regarding part (a), it is mandatory that a vehicle must never stop on top of a box junction, that is, from instant n, *when the vehicle overlaps the delimited region, at* $n+1$ *it cannot be in the exact same position as it was in the previous moment. Writing in LTL × MS we have:*

$$\square\left(\mathtt{I}\left(\mathbf{C},\mathbf{BJ}\right)\vee\mathtt{O}\left(\mathbf{C},\mathbf{BJ}\right)\rightarrow\neg\mathtt{EQ}\left(\mathbf{C},\odot\mathbf{C}\right)\right), \quad (6)$$

where **BJ** *corresponds to the box junction. The previous implication is extended by using the logical equivalence* $\varphi_1\rightarrow\varphi_2\equiv\neg\varphi_1\vee\varphi_2$. *First we expand* $\odot$ *and* $\square$ *operators,*

$$\top\,\mathcal{U}\left[\neg\Big(\mathtt{I}\left(\mathbf{C},\mathbf{BJ}\right)\vee\mathtt{O}\left(\mathbf{C},\mathbf{BJ}\right)\Big)\vee\neg\mathtt{EQ}\left(\mathbf{C},\bot\mathfrak{U}\mathbf{C}\right)\right],$$

then the predicates $\mathtt{O}$, $\mathtt{EQ}$ *and* $\mathtt{I}$,

$$\begin{aligned}\top\,\mathcal{U}\,\Big[\Big(\big(\neg(\mathbf{C}\sqsubseteq\mathbf{BJ})\vee\mathbf{BJ}\sqsubseteq\mathbf{C}\big)\wedge\big((\mathbf{C}\sqcap\mathbf{BJ}=\bot)\vee\mathbf{C}\sqsubseteq\mathbf{BJ}\vee\mathbf{BJ}\sqsubseteq\mathbf{C}\big)\Big)\\ \vee\neg\big(\mathbf{C}\sqsubseteq\bot\mathfrak{U}\mathbf{C}\wedge\bot\mathfrak{U}\mathbf{C}\sqsubseteq\mathbf{C}\big)\Big].\end{aligned} \quad (7)$$

This rule is now ready for the monitor generation. Parts (b) and (c) have an analogous encoding but with crosswalk and tramway regions, respectively.

Rule 3 (stop-sign). *According to the road traffic laws, a vehicle shall stop at a stop sign within a maximum distance of one meter. In LTL × MS, this rule can be described in a compact form by*

$$\square\Big[\Big(\mathtt{O}\left(\mathbf{S},\exists^{\leq 1}\mathbf{C}\right)\wedge\neg\diamondminus\mathtt{EQ}\left(\mathbf{C},\odot\mathbf{C}\right)\Big)\rightarrow\Diamond\Big(\mathtt{EQ}\left(\mathbf{C},\odot\mathbf{C}\right)\wedge\mathtt{DC}\left(\mathbf{S},\diamonddot\mathbf{C}\right)\Big)\Big], \quad (8)$$

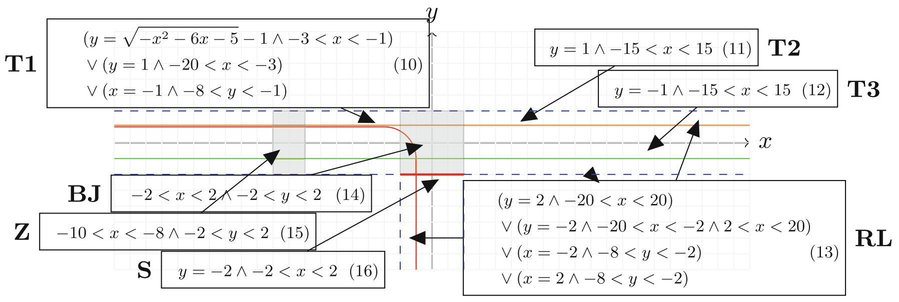

T1
$(y=\sqrt{-x^2-6x-5}-1\wedge -3<x<-1)$
$\vee(y=1\wedge -20<x<-3)$ (10)
$\vee(x=-1\wedge -8<y<-1)$
y
$y=1\wedge -15<x<15$ (11)
T2
$y=-1\wedge -15<x<15$ (12)
T3
x
BJ
$-2<x<2\wedge -2<y<2$ (14)
Z
$-10<x<-8\wedge -2<y<2$ (15)
S
$y=-2\wedge -2<x<2$ (16)
$(y=2\wedge -20<x<20)$
$\vee(y=-2\wedge -20<x<-2\wedge 2<x<20)$
$\vee(x=-2\wedge -8<y<-2)$ (13)
$\vee(x=2\wedge -8<y<-2)$
RL

Fig. 4. Scenario encoding as spatial variables **T1**, **T2**, **T3**, **RL**, **BJ**, **Z** and **SL**.

where **S** *is the location of the stop sign. Starting by the expansion of* $\odot$, $\Diamonddot$, $\Diamond$, $\diamondminus$, `O`, `EQ`, `DC`, *and* $\Box$ *operators, we have*

$$\top\,\mathcal{U}\Big[\big(\mathbf{S}\sqcap\exists^{\leq 1}\mathbf{C}=\bot\big)\vee\neg\big(\mathbf{S}\sqsubseteq\exists^{\leq 1}\mathbf{C}\big)\vee\neg\big(\exists^{\leq 1}\mathbf{C}\sqsubseteq\mathbf{S}\big)$$
$$\vee\Big(\top\,\mathcal{S}\big(\mathbf{C}\sqsubseteq(\,\bot\mathfrak{U}\,\mathbf{C})\wedge(\,\bot\mathfrak{U}\,\mathbf{C})\sqsubseteq\mathbf{C}\big)\Big)$$
$$\vee\Big(\top\,\mathcal{U}\big((\mathbf{C}\sqsubseteq(\,\bot\mathfrak{U}\,\mathbf{C})\wedge(\,\bot\mathfrak{U}\,\mathbf{C})\sqsubseteq\mathbf{C})\wedge(\mathbf{S}\sqcap(\,\top\mathfrak{U}\,\mathbf{C})=\bot)\big)\Big)\Big]. \qquad (9)$$

The derived expressions (3), (5), (7) and (9) of the three considered rules are ready to be used as input to the monitor generation algorithm presented later. Let us proceed with the scenario and trace encoding of our running example.

3.2 Scenario and Trace Encoding (Static and Dynamic Objects)

Figure 4 shows the encoding for each static object present in the model of Fig. 3b, which is based on region restrictions represented as inequalities. The trajectories and road limits are described by line segments which can be expressed as sets of linear and non-linear polynomials. The box junction and the crosswalk are defined by bounding boxes.

The objects present in our running example are divided into two categories: static objects, as road limits, crosswalk, box junction, etc.; and objects that can behave dynamically over time, such as vehicles, trams or pedestrians. For these latter elements, there is a need of a continuous trace to keep track of their position at every time step. At all instants, the trace is sent from the simulator in the form of a tree data structure, and it is translated into formulas written in $\text{FOL}_{\mathbb{R}}$ (see Fig. 4). In practice, the scenario and trace are transformed in such way that a satisfiability solver can interpret them. Let us turn our attention to trace definition and encoding.

Definition 4 (Infinite Trace). *An infinite trace forms the set* $A^{\mathbb{N}_0}=\{\sigma:\mathbb{N}_0\mapsto A\}$, *where* $(\sigma_0,\sigma_1,\sigma_2,\dots)$ *defines a sequence of symbols.*

For the sake of simplicity, we will use the function $add : \mathbb{S} \times \mathbb{L} \mapsto \mathbb{B}$ to add constraints to the set h (hash map), and $find : \mathbb{S} \mapsto \mathbb{L}$ to return the constraint with a given index (string). These functions are not effect-free. Also, the $next : A^{\mathbb{N}_0} \mapsto A^{\mathbb{N}_0}$ function over the sequence of symbols with type $A^{\mathbb{N}_0}$ is defined by $next\ ((Cons(h, t)) := t()$, and $now : A^{\mathbb{N}_0} \mapsto A$ by $now(Cons(h, t)) := h$. These functions get the next sequence of symbols and the current symbol in the sequence, respectively. An A symbol has a list of objects, and set $A_{[O]}$ is a list of A_O objects (see the JSON trace in Fig. 5). For the sake of simplicity, we also define the dual of "next" as $prev : A^{\mathbb{N}_0} \mapsto A^{\mathbb{N}_0}$.

In general terms, the trace encoding consists on the construction of the function $eval : \mathfrak{P} \mapsto \mathbb{L}$ that is defined by $eval(p) := find\ p$, which evaluates a spatial variable to an expression in $\text{FOL}_{\mathbb{R}}$. To encode a symbol from a trace, we have to pick the symbol from the trace and produce the set of inequality constraints that defines their objects. Figure 5 presents the definition of the *enc* function and other auxiliary functions, where the *enc* function gets as input a symbol and produces the constraints with an index to the set h. Also, the function *encode* constructs the set of constraints for a given finite trace. To encode infinite traces, we have to infinitely iterate over trace symbols and produce the inequalities in an incremental way. Instead of defining a new encoding function, we make use the *next* and *prev* functions in the next section.

Without loss of generality, let us see a circle as a ball, and a bounding box as a rectangle or square in the two-dimensional Euclidean space. Note that other geometric shapes can be translated but are out of the scope of our running example. The $obj : id \mapsto \mathbb{L}$ function generates the objects as constraints defining circles and bounding boxes with free variables, and $id \in \{\texttt{circle}, \texttt{bbox}\}$. It defines, as follows:

$$obj(s) := \begin{cases} (\texttt{x4} - \texttt{x1})^2 + (\texttt{x5} - \texttt{x2})^2 < \texttt{x3}^2, & \text{if } s = \texttt{circle} \\ (\texttt{x1} - \texttt{x3}/2) \leq \texttt{x5} \wedge \texttt{x5} \leq (\texttt{x1} + \texttt{x3}/2) \wedge & \\ (\texttt{x2} - \texttt{x4}/2) \leq \texttt{x6} \wedge \texttt{x6} \leq (\texttt{x2} + \texttt{x4}/2), & \text{if } s = \texttt{bbox} \end{cases}.$$

Let us now see how the resultant expressions can use the variables binder `let`. The evaluation of the expression `let` ((`x1` 1) (`x2` 2) (`x3` 3)). *obj* `circle` results in $(\texttt{x1} - 1)^2 + (\texttt{x2} - 2)^2 < 3^2$, where `x1` and `x2` are the remaining free variables. Then, we can bind these variables with a quantifier such as $\forall x, y.$ `let` (`x1` 1) (`x2` 2) (`x3` 3). $obj(\texttt{circle})$, where $(1, 2)$ is the center point of the circle, and 3 the radius. This will be the way we replace free variables.

4 Monitoring Model Construction

As input our algorithm receives an LTL $\times$ MS property that represents a requirement under analysis and produces a model in $\text{FOL}_{\mathbb{R}}$. Every term $\rho \in \mathfrak{T}$ (the set of all words of ϱ) is translated by the recursive function $\mathbf{conv}_{\varrho} : \mathfrak{T} \mapsto \mathbb{L}$ into $\text{FOL}_{\mathbb{R}}$ (see Fig. 6). The $dist : \mathbb{R} \times \mathfrak{T} \mapsto \mathbb{L}$ function applies the Property 1 that says that any formula containing distance operators has an equivalent formula where the distance operators are just applied to the propositions.

```
$c : A \mapsto A_{[O]}$
$c(s) := s.objects$
$enc : A_{[O]} \mapsto \mathbb{B}$
$enc(l) := if\ l = empty\ then\ true$
  $else\ enco(hd(l))$ and $enc(tl(l))$
$enco : A_O \mapsto \mathbb{B}$
$enco(o) := if\ s.type =$ circle $then$
  $add(o.id,$ let (x1 $o.x$) (x2 $o.y$)
    (x3 $o.radius$). $obj$(circle))
  $else\ (if\ s.type =$ bbox $then$
    $add(o.id,$ let (x1 $o.x$) (x2 $o.y$)
      (x3 $o.width$) (x4 $o.height$).
      $obj$(bbox)) $else\ false$ )
$encode : A^{\mathbb{N}_0} \times \mathbb{N}_{>0} \mapsto \mathbb{L}$
$encode(t, n) :=$ if $n > 0$ then
   $enc(c(now(t)))$ and
   $encode(next(t), n-1)$ else $true$
```

(a) Definition of the function *encode* and its auxiliar functions.

```
{
 "trace": [
   {
     "symbolid": "1",
     "ts": "00:00:01",
     "objects": [
       {
         "id": 1,
         "position": {
           "x": 0.5,
           "y": -0.5
         },
         "region": {
          "type": "circle",
          "radius": 0.5
         }
       },
     ... ]
   },
 ... ]
}
```

(b) Trace with one circular object with center $(0.5, -0, 5)$ and radius 0.5.

Fig. 5. Functional definition and example of a trace in JSON format.

Property 1 (Distance Operator). Let ρ be a term, V the set of free variables in ρ, and e a rational number. For any ρ and e, the distance operator $\exists^{\leq e}\rho$ has an equivalent expression with every free variable $a \in V$ such that $\exists^{\leq e}a$.

The $next_\varrho : \mathfrak{T} \mapsto \mathbb{L}$ function also has a similar property to distance operators but instead of distance it assigns to each proposition the successor (a nested of next operators just on propositions). To conclude $\mathbf{conv}_\varrho$ conversion function over terms, the *unfold* : $\mathfrak{T} \times \mathfrak{T} \mapsto \mathbb{L}$ function generates a bounded instance of the infinite sequence

$$\bigvee_{i=1}^{n} \left[\bigwedge_{j=1}^{i} \left(\underbrace{\odot \ldots \odot}_{\text{j times}} \rho_1 \right) \wedge \underbrace{\odot \ldots \odot}_{\text{j times}} \rho_2 \right],$$

where $\rho_1, \rho_2 \in \varrho$ are the input terms. Let us now move our attention to formulas.

Every formula ϕ in $\mathfrak{F}$ (the set of all words of φ) is translated by the function $\mathbf{conv}_\varphi : \mathfrak{F} \mapsto \mathbb{L}$ (again in Fig. 6). The expression $\forall(x, y, \cdot).(\mathbf{conv}_\varrho(\rho_1) \rightarrow \mathbf{conv}_\varrho(\rho_2))$ binds all the remaining free variables of the resulting expression in $\mathrm{FOL}_\mathbb{R}$. For instance, $\forall(x, y).x < y$. The function $next_\varphi : \mathfrak{F} \mapsto \mathbb{L}$ generates

$$\mathbf{conv}_{\varrho}(\rho) := \begin{cases} eval(p), & \text{if } \rho = p \\ \neg\mathbf{conv}_{\varrho}(\rho), & \text{if } \rho = \overline{\rho} \\ \mathbf{conv}_{\varrho}(\rho_1) \wedge \mathbf{conv}_{\varrho}(\rho_2), & \text{if } \rho = \rho_1 \sqcap \rho_2 \\ \mathbf{conv}_{\varrho}(\rho_1) \vee \mathbf{conv}_{\varrho}(\rho_2), & \text{if } \rho = \rho_1 \sqcup \rho_2 \\ dist(e, \mathbf{conv}_{\varrho}(\rho)), & \text{if } \rho = \exists^{\leq e} \rho \\ next_{\varrho}(\mathbf{conv}_{\varrho}(\rho)), & \text{if } \rho = \ \bot \mathfrak{U}\, \rho \\ \mathbf{conv}_{\varrho}(unfold(\rho_1, \rho_2)), & \text{if } \rho = \ \rho_1 \mathfrak{U}\, \rho_2, \end{cases}$$

$$\mathbf{conv}_{\varphi}(\phi) := \begin{cases} \forall(x, y, \cdot).(\mathbf{conv}_{\varrho}(\rho_1) \to \mathbf{conv}_{\varrho}(\rho_2)), & \text{if } \phi = \rho_1 \sqsubseteq \rho_2 \\ \neg(\mathbf{conv}_{\varphi}(\phi)), & \text{if } \phi = \neg\rho \\ \mathbf{conv}_{\varphi}(\phi_1) \wedge \mathbf{conv}_{\varphi}(\phi_2), & \text{if } \phi = \phi_1 \wedge \phi_2 \\ \mathbf{conv}_{\varphi}(\phi_1) \vee \mathbf{conv}_{\varphi}(\phi_2), & \text{if } \phi = \phi_1 \vee \phi_2 \\ next_{\varphi}(\mathbf{conv}_{\varphi}(\phi)), & \text{if } \phi = false\,\mathcal{U}\,\phi \\ \mathbf{conv}_{\varphi}(unfold_{\mathcal{U}}(\phi_1, \phi_2)), & \text{if } \phi = \phi_1\,\mathcal{U}\,\phi_2 \\ previous_{\varphi}(\mathbf{conv}_{\varphi}(\phi)), & \text{if } \phi = false\,\mathcal{S}\,\phi \\ \mathbf{conv}_{\varphi}(unfold_{\mathcal{S}}(\phi_1, \phi_2)), & \text{if } \phi = \phi_1\,\mathcal{S}\,\phi_2 \end{cases}$$

Fig. 6. Conversion functions $\mathbf{conv}_{\varrho}(\rho)$ and $\mathbf{conv}_{\varphi}(\phi)$.

formula ϕ from the next instance, while $previous_{\varphi} : \mathfrak{F} \mapsto \mathbb{L}$ generates formula ϕ from the previous instance. Function $unfold_X : \mathfrak{F} \times \mathfrak{F} \mapsto \mathbb{L}$ generates a bounded instance of the infinite sequence

$$\bigvee_{i=1}^{n} \left[\bigwedge_{j=1}^{i} \left(\underbrace{X \ldots X}_{\text{j times}} \rho_1 \right) \wedge \underbrace{X \ldots X}_{\text{j times}} \rho_2 \right]$$

where $\phi_1, \phi_2 \in \varphi$ are the input formulas. Funtion $unfold_{\mathcal{U}} : \mathfrak{F} \times \mathfrak{F} \mapsto \mathbb{L}$ is defined by $unfold_X$ when $X = \bigcirc$, while $unfold_{\mathcal{S}} : \mathfrak{F} \times \mathfrak{F} \mapsto \mathbb{L}$ by $unfold_X$ when $X = \ominus$.

Trace Inlining. Since formulas and terms converts into incomplete $\mathrm{FOL}_{\mathbb{R}}$ expressions, the formalization of the trace completes the encoding. The trace encoding consists essentially on the construction of the function *eval* that has ben already defined. This function replaces spatial variables with expressions in $\mathrm{FOL}_{\mathbb{R}}$. Note that the trace is a valuation and assigns constraints to the expressions in $\mathrm{FOL}_{\mathbb{R}}$. *encode* has been already defined while $inline : \mathbb{L} \times \mathbb{L} \mapsto \mathbb{L}$ includes the trace in the monitoring model. Note that this inlining is a binding of every free variable of $\mathbf{conv}_{\varphi}(\phi)$ in $encode(trc, n)$. The first argument receives the monitoring model, and the second argument receives the mapping of the spatial variables to the constraints in $\mathrm{FOL}_{\mathbb{R}}$ (given by the hash map). Let *trc* be a trace of length n, and ϕ a formula in LTL $\times$ MS.. The inlining is defined by

$$inline(\mathbf{conv}_{\varphi}(\phi), encode(trc, n)).$$

The process concludes by inlining the finite trace in the monitoring model.

Partial Incremental Evaluation – Without Unfolding Temporal Operators. To improve algorithm efficiency, scalability, and support infinite traces we decided to construct a modified version of the previous algorithm without using the unfolding of temporal operators (functions $\mathcal{U}$ and $\mathcal{S}$). We perform this on the assumption that temporal terms are bounded. The temporal part is then processed incrementally using incremental evaluation (push and pop operators) on the non-linear satisfiability solver. The $\mathtt{s}$ acts as a state such as true $\mathfrak{t}$, false $\mathfrak{f}$ or unknown $\mathfrak{u}$. For this evaluation, we consider the known temporal patterns

$$\Box\hat{\phi}, \Diamond\hat{\phi}, \Box(\hat{\phi}_1 \rightarrow \Diamond\hat{\phi}_2), \Box(\hat{\phi}_1 \rightarrow \neg\Diamond\hat{\phi}_2), \Diamond(\hat{\phi}_1 \wedge \Box\hat{\phi}_2), \text{ and } \Diamond(\hat{\phi}_1 \wedge \Diamond\neg\hat{\phi}_2).$$

Past temporal operators are unrolled for infinite traces and incrementally evaluated for infinite traces (unknown last element). The $eval_{\mathrm{i}}(\Box\,\phi)$ has the truth value *false* or *unknown*, while $eval_{\mathrm{i}}(\Diamond\,\phi)$ has the truth value *true* or *unknown*, and $eval_{\mathrm{i}}(\Box\,(\phi_1 \rightarrow \Diamond\,\phi_2))$ has the same truth value of $eval_{\mathrm{i}}(\Box\,\phi)$.

One could expect to construct the function $eval_{\mathrm{i}}(\phi, \Sigma, \mathtt{s})$ defined by

$$\begin{cases} solve\big(conv^i_\varphi(\phi), enc(\Sigma)\big) & \text{if } \phi = \hat{\phi} \\ and\Big[eval_{\mathrm{i}}\Big(\hat{\phi}_1, \Sigma, \mathtt{s}\Big), eval_{\mathrm{i}}\Big(\phi_1, \Sigma, \mathtt{s}\Big)\Big] & \text{if } \phi = \hat{\phi}_1 \wedge \phi_1 \\ implies\Big[eval_{\mathrm{i}}\Big(\phi_1, \Sigma, \mathtt{s}\Big), eval_{\mathrm{i}}\Big(\phi_2, \Sigma, \mathtt{s}\Big)\Big] & \text{if } \phi = \phi_1 \rightarrow \phi_2 \\ ite\Big[\mathtt{s} = \mathfrak{u}, eval_{\mathrm{i}}\Big(\phi, next(\Sigma), c_\top\big(\mathtt{s}, eval_{\mathrm{i}}(\phi_1, next(\Sigma), \mathfrak{u})\big)\Big), \mathfrak{f}\Big] & \text{if } \phi = \Box\,\phi_1 \\ ite\Big[\mathtt{s} = \mathfrak{u}, eval_{\mathrm{i}}\Big(\phi, next(\Sigma), c_\bot\big(\mathtt{s}, eval_{\mathrm{i}}(\phi_1, next(\Sigma), \mathfrak{u})\big)\Big), \mathfrak{t}\Big] & \text{if } \phi = \Diamond\,\phi_1 \\ ite\Big[\mathtt{s} = \mathfrak{u}, eval_{\mathrm{i}}\Big(\phi, prev(\Sigma), c_\bot\big(\mathtt{s}, eval_{\mathrm{i}}(\phi_1, prev(\Sigma), \mathfrak{u})\big)\Big), \mathfrak{t}\Big] & \text{if } \phi = \diamondminus\,\phi_1 \end{cases}$$

where $\phi \in \varphi$ a formula, $\hat{\phi}$ a formula without temporal operators, Σ is a infinite trace with $next, prev$ operators, and $\mathtt{s} \in \mathfrak{S}$ a symbol. $\mathfrak{S}$ denotes the set $\{\mathfrak{t}, \mathfrak{f}, \mathfrak{u}\}$, $ite : \{\mathfrak{t}, \mathfrak{f}\} \times \{\mathfrak{t}, \mathfrak{f}\} \times \{\mathfrak{t}, \mathfrak{f}\} \mapsto \{\mathfrak{t}, \mathfrak{f}\}$ defines the if-then-else function, $and : \{\mathfrak{t}, \mathfrak{f}\} \times \{\mathfrak{t}, \mathfrak{f}\} \mapsto \{\mathfrak{t}, \mathfrak{f}\}$ implements the conjunction, $implies : \{\mathfrak{t}, \mathfrak{f}\} \times \{\mathfrak{t}, \mathfrak{f}\} \mapsto \{\mathfrak{t}, \mathfrak{f}\}$ implements the implication, $c_\top : \mathfrak{S} \times \{\mathfrak{t}, \mathfrak{f}\} \mapsto \mathfrak{S}$ converts the pair $(\mathfrak{u}, \mathfrak{f})$ to $\mathfrak{f}$ and $\mathfrak{u}$ otherwise, and $c_\bot : \mathfrak{S} \times \{\mathfrak{t}, \mathfrak{f}\} \mapsto \mathfrak{S}$ converts the pair $(\mathfrak{u}, \mathfrak{t})$ to $\mathfrak{t}$ and $\mathfrak{u}$ otherwise.

Property 2 (Spatial Isolation on ϱ terms). A spatial variable $a \in \mathfrak{P}$ is free of modifiers for any term.

From Property 2, terms have no free variables and no assumptions have to be given for the incremental evaluation, the reason why it simplifies *implies* and *and* functions in the incremental evaluation function $eval_{\mathrm{i}}$. A spatial variable maintains its form regardless of where it is evaluated. Function $solve : \mathbb{L} \times \mathbb{L} \mapsto \{\mathfrak{t}, \mathfrak{f}\}$ solves an expression in $\mathrm{FOL}_{\mathbb{R}}$ assuming another expression in $\mathrm{FOL}_{\mathbb{R}}$. Note that $\bigcirc$ temporal operator vanish and is not incrementally evaluated.

Table 1. Table displaying the evaluation results. The first column indicates the considered rules. The last two columns, unroll and incremental methods, show the time (in seconds) and the memory (in Megabytes) used by the solver, the overall runtime the monitor takes to execute (RT) and frames per second (FPS).

			Unroll				Incremental			
	Rule($\|\varrho\|, \|\varphi\|$)	$\Sigma(\|\Sigma\|)$	Solver		RT	FPS	Solver		RT	FPS
			Time	Mem			Time	Mem		
Empirical	1.a (2,2)	e1(13)	1.14	4.49	1.19	5.58	0.26	2.87	0.42	19.12
	1.a (2,2)	e2(13)	0.07	4.05	0.13	65	0.03	2.86	0.04	185.71
	1.b (1,1)	e3(13)	0.02	3.03	0.05	185.71	0.12	2.75	0.26	34.21
	1.b (1,1)	e4(13)	0.05	3.06	0.09	92.86	0.08	2.45	0.17	56
	2.a (1,3)	e5(13)	0.18	3.53	0.23	31.71	0.17	2.83	0.33	26
	2.a (1,3)	e6(13)	0.16	3.51	0.21	35.14	0.09	2.85	0.17	53.85
	2.b (1,3)	e7(13)	0.25	3.53	0.29	24.07	0.17	2.80	0.34	27.45
	3. (3,6)	e8(14)	0.50	6.98	1.05	9.03	0.07	5.40	0.26	39.39
	3. (3,6)	e9(13)	1.29	7.06	1.71	4.33	0.10	5.45	0.28	36.84
	3. (3,6)	e10(15)	1.11	7.45	1.64	5.45	0.11	5.44	0.46	22.81
	Average		0.48	4.67	0.66	**45.9**	0.12	3.57	0.27	**50.1**
Simulator	1.a (2,2)	s1(243)	73.82	18.48	74.29	1.64	2.38	2.91	3.18	43.71
	1.a (2,2)	s2(157)	0.34	5.78	0.46	196.25	0.79	2.89	1.08	83.96
	1.a (2,2)	s3(146)	0.28	6.37	0.44	202.78	0.51	2.89	0.74	116.8
	1.b (1,1)	s1(243)	0.15	5.14	0.45	405	0.70	2.79	1.52	109.46
	1.b (1,1)	s4(311)	0.40	5.19	0.64	299.04	0.97	2.85	1.75	114.34
	2.a (1,3)	s1(243)	6.73	7.95	7.09	17.58	1.19	2.86	2.07	74.54
	2.a (1,3)	s5(369)	12.99	7.66	13.72	13.82	2.37	2.90	3.96	58.29
	2.a (1,3)	s6(198)	8.11	7.83	8.69	11.79	0.46	2.85	0.80	157.14
	3. (3,6)	s4(311)	396.40	110.23	413.11	0.38	10.83	7.80	23.71	9
	3. (3,6)	s5(369)	951.87	117.48	1029.76	0.19	9.90	8.31	27.27	9.93
	3. (3,6)	s6(198)	1044.16	124.92	1090.95	0.09	12.25	8.52	26.57	5.1
	Average		226.84	37.91	239.96	**104.4**	3.85	4.32	8.42	**71.1**

5 Empirical Evaluation

The monitor runs in parallel to the ADS under test having no direct impact on the system itself, as seen in Fig. 1. The system evolves around the simulation of a specific scenario, that feeds ADS with its observations. The system reacts to observation and produces actions for the agents running on the simulator in an endless loop. The monitor receives the observations from the simulator as a trace to check a property and generates a verdict indicating if its satisfied.

The traces and scenario were evaluated on a i5-8365U CPU running Linux 5.10.11. Traces are provided by a simulated T-shaped junction scenario in the CARLA 0.9.13 autonomous driving simulator [9]. Scalability is an aspect to keep in mind since the size of a trace matters for monitoring performance, therefore, we test each property with different trace sizes to understand how different methods perform. When performing the empirical evaluation (hand-built sample traces to validate the tool – e1–e10), the Unroll method is slower than the Incremental method in average, with exception of rule 1.b, where the Unroll is slightly better, with a higher memory consumption (see Table 1). However, the biggest difference are the rules 1.a and 3, where the Incremental method is clearly better than the Unroll. These rules impact the highest average in the time (0.48 s) and memory spent (4.67 MB) by the solver, as well as the RT (0.66 s) and low FPS values (45.9) of the Unroll, in comparison with the Incremental method.

The behavior described previously also applies when it comes to the Simulator evaluation (traces got from simulation environment – s1–s6). Yet, the differences are more pronounced. In the worst case scenario (rule 3), the time spent by the solver in the Unroll method is approximately 85 times slower than the solver in Incremental method, 1044.16 s and 12.25 s, respectively. Moreover, the memory usage is considerably higher in Unroll (in average 37.91 MB) than in Incremental (in average 4.32 MB) method.

When observing the average value of the incremental method (higher than 60), this value means that our approach is able to comfortably work with modern cameras with an acquisition rate of 60 Hz. ADSs cameras have lower framerates. Our performance measurements are also prone to different resolutions as our approach does not depend of the size of the image matrix. To summarize, the data displayed in Table 1 shows a clear advantage of Incremental over Unroll method. The tool and documentation for artifact evaluation can be found in https://github.com/anmaped/stem-binaries.git.

6 Related Work

When talking about autonomous vehicles, these systems are subjected to the local traffic laws and is a crucial problem to solve, as pointed out by Henry Prakken [26]. He studied if the Dutch traffic law, with its exceptions, conflicts, and commonsense knowledge, can be implemented in fully autonomous vehicles and present three approaches to design AVs in compliance with traffic rules. Cristian-Ioan Vasile *et al.* [32] formalized a minimum-violation plan of an AV by using a fragment of LTL. Moreover, they used the logic fragment to specify the behavior and incorporate it in the motion planner algorithm.

Alternatively, the AV as a system ideally has to self-check whether the autonomous part obeys the traffic rules. Based on signal temporal logic (STL), Nikos Aréchiga [4] proposed a step forward in this direction. He enabled the automatic synthesis of runtime monitors, similar to what we presented in this work, but without considering space as a first-class citizen. Also, he defined a set of contracts to ensure that the overall system will not have collisions if followed by all traffic participants. Cardoso *et al.* [7] suggests verification by contracts as a powerful tool to handle complex systems such as AVs.

Similar to our work, Xu and Li [34] introduced a spatial logic to check collision avoidance properties. They do not consider the evolution in time of the traffic junction with its actors and do not produce any verdict. Another work that resembles ours is [29], where they encode STL to a mixed-integer programming solver, allowing the monitoring of AV failures in an urban scenario in real-time. In our work, we encode our LTL $\times$ MS expression to $\text{FOL}_{\mathbb{R}}$ as well, but traffic rules are not formalized in STL as we do with our temporal language.

Several works focus on the formalization of traffic rules. For example, [5] uses Defeasible Deontic Logic to handle exceptions and resolve conflicts in overtaking Australian traffic rules. In terms of temporal logic, the research presented in [3,21,28], addressed several traffic scenarios, such as highways and junctions. Schwammberger and Alves [31] proposed a spatio-temporal language similar to the one in our work to formalize three road crossing rules in the UK and emphasizes the need for a Digital Highway Code for AVs, but the decision procedure is missing. Pek *et al.* [25] writes overtaking rules as non-linear arithmetic expressions and uses real-world data and simulations to validate their method.

7 Conclusion and Future Work

Even with smarter techniques, unfolding the $\mathcal{U}$ and $\mathcal{S}$ operators is computationally expensive and proves infeasible in practical terms. Incremental evaluation of infinite traces at run-time reduces the burden of checking spatial constraints, since unbounded time is a bottleneck when solving time constraints with a satisfiability solver. In our approach, the temporal sequences are checked partially at runtime and the spatial part using exclusively the satisfiability solver.

Our empirical evaluation shows good evidence of the scalability of our incremental evaluation method by running symbols of arbitrary sequences with more 70 symbols or 'frames' per second. To emphasize it, a conventional CPU (one core) could monitor a trace from a camera with a total acquisition rate greater than 60 Hz which we tested by setting up our running example on the CARLA autonomous driving simulator. Our approach also takes advantage of multiple cores as we could split the objects in the environment into different instances, the Ego vehicle and the surrounding objects.

One way to optimize our tool, is to configure the solver to use the most suitable tactic, tailoring it even more for the models we intend to verify. Another way, is to increase the number of surrounding objects and use predictive distance-based techniques based on geometric projections to allow the monitor to skip symbols of a sequence and decrease CPU utilization.

Acknowledgments. This work was partially supported by the European Regional Development Fund (ERDF) through the Competitiveness and Internationalization Operational Program (COMPETE 2020) of Portugal 2020 [Project STEROID with number 069989 (POCI-01-0247-FEDER-069989)]. This work was also partially supported by FCT/MCTES grant UIDB/04516/2020.

References

1. Aiello, M., Pratt-Hartmann, I., van Benthem, J.: Handbook of Spatial Logics. Springer, Dordrecht (2007). https://doi.org/10.1007/978-1-4020-5587-4

2. Akintunde, M.E., Botoeva, E., Kouvaros, P., Lomuscio, A.: Formal verification of neural agents in non-deterministic environments. Auton. Agents Multi-Agent Syst. **36**(1), 1–36 (2021). https://doi.org/10.1007/s10458-021-09529-3
3. Alves, G.V., Dennis, L.A., Fisher, M.: A double-level model checking approach for an agent-based autonomous vehicle and road junction regulations. J. Sens. Actuator Netw. **10**(3), 41 (2021)
4. Aréchiga, N.: Specifying safety of autonomous vehicles in signal temporal logic. In: 2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, 9–12 June 2019, pp. 58–63. IEEE (2019)
5. Bhuiyan, H., Governatori, G., Bond, A., Demmel, S., Badiul Islam, M., Rakotonirainy, A.: Traffic rules encoding using defeasible deontic logic. In: JURIX 2020, Brno, Czech Republic, December 2020, volume 334 of Frontiers in Artificial Intelligence and Applications, pp. 3–12. IOS Press (2020)
6. Borg, M., et al.: Safely entering the deep: a review of verification and validation for machine learning and a challenge elicitation in the automotive industry. J. Autom. Softw. Eng **1**, 12 (2018)
7. Cardoso, R., et al.: A review of verification and validation for space autonomous systems. Curr. Robot. Rep. **2**, 09 (2021)
8. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
9. Dosovitskiy, A., Ros, G., Codevilla, F., López, A.M., Koltun, V.: CARLA: an open urban driving simulator. In: CoRL 2017, Mountain View, California, USA, November 2017, Proceedings, volume 78 of Machine Learning Research, pp. 1–16. PMLR (2017)
10. Allen Emerson, E.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, pp. 995–1072. Elsevier and MIT Press, London (1990)
11. Association for Standardisation of Automation and Measuring Systems. https://www.asam.net/standards/. Accessed 11 Apr 2022
12. Gabelaia, D., Kontchakov, R., Kurucz, A., Wolter, F., Zakharyaschev, M.: Combining spatial and temporal logics: expressiveness vs. complexity. J. Artif. Intell. Res. **23**, 167–243 (2005)
13. Gerevini, A., Nebel, B.: Qualitative spatio-temporal reasoning with RCC-8 and Allen's interval calculus: computational complexity. In: ECAI'2002, Lyon, France, July 2002. Proceedings, pp. 312–316. IOS Press (2002)
14. Haghighi, I., Jones, A., Kong, Z., Bartocci, E., Grosu, R., Belta, C.: SpaTeLl: a novel spatial-temporal logic and its applications to networked systems: a novel spatial-temporal logic and its applications to networked systems. In: HSCC 2015, Seattle, WA, USA, April 2015. Proceedings, pp. 189–198. ACM (2015)
15. Huang, X., et al.: A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev. **37**, 100270 (2020)
16. Kane, A.: Runtime monitoring for safety-critical embedded systems. Ph.D. thesis, Carnegie Mellon University, Pittsburgh (2015)
17. Kurucz, A., Wolter, F., Zakharyaschev, M.: Modal logics for metric spaces: open problems. In: We Will Show Them! Essays in Honour of Dov Gabbay, Vol. 2, pp. 193–108. College Publications (2005)
18. Kutz, O., Wolter, F., Sturm, H., Suzuki, N.-Y., Zakharyaschev, M.: Logics of metric spaces. ACM Trans. Com. Log. **4**(2), 260–294 (2003)

19. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Logic Algebraic Program. **78**(5), 293–303 (2009)
20. Li, T., STSL: a novel spatio-temporal specification language for cyber-physical systems. In: QRS 2020, pp. 309–319. IEEE (2020)
21. Maierhofer, S., Rettinger, A., Charlotte Mayer, E., Althoff, M.: Formalization of interstate traffic rules in temporal logic. In: 2020 IEEE Intelligent Vehicles Symposium (IV), pp. 752–759. IEEE (2020)
22. Mehmed, A.: Runtime monitoring for safe automated driving systems. Ph.D. thesis, Mälardalen University (2020)
23. Muller, P.: A qualitative theory of motion based on spatio-temporal primitives. In: KR1998, Trento, June 1998, pp. 131–143. Morgan Kaufmann (1998)
24. United Nations. Vienna convention on road traffic (1968). https://unece.org/DAM/trans/conventn/Conv_road_traffic_EN.pdf. Accessed 11 Apr 2022
25. Pek, C., Zahn, P., Althoff, M.: Verifying the safety of lane change maneuvers of self-driving vehicles based on formalized traffic rules. In: 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 1477–1483 (2017)
26. Prakken, H.: On the problem of making autonomous vehicles conform to traffic law. Artif. Intell. Law **25**(3), 341–363 (2017). https://doi.org/10.1007/s10506-017-9210-0
27. Riedmaier, S., Ponn, T., Ludwig, D., Schick, B., Diermeyer, F.: Survey on scenario-based safety assessment of automated vehicles. IEEE Access **8**, 87456–87477 (2020)
28. Rizald, A., et al.: Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/HOL. In: Polikarpova, N., Schneider, S. (eds.) IFM 2017. LNCS, vol. 10510, pp. 50–66. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66845-1_4
29. Sahin, Y.M., Quirynen, R., Di Cairano, S.: Autonomous vehicle decision-making and monitoring based on signal temporal logic and mixed-integer programming. In: 2020 American Control Conference (ACC), pp. 454–459 (2020)
30. Sánchez, C., et al.: A survey of challenges for runtime verification from advanced application domains (beyond software). Formal Methods Syst. Des. **54**, 279–335 (2019). https://doi.org/10.1007/s10703-019-00337-w
31. Schwammberger, M., Alves, G.V.: Extending urban multi-lane spatial logic to formalise road junction rules. In: FMAS 2021, Virtual, October 2021. Proceedings, volume 348 of EPTCS, pp. 1–19 (2021)
32. Vasile, C.-I., Tumova, J., Karaman, S., Belta, C., Rus, D.: Minimum-violation scLTL motion planning for mobility-on-demand. In: ICRA 2017, pp. 1481–1488 (2017)
33. Wolter, F., Zakharyaschev, M.: Reasoning about distances. In: Gottlob, G., Walsh, T. (eds.) IJCAI 2003, Acapulco, Mexico, 9–15 August 2003. Proceedings, pp. 1275–1282. Morgan Kaufmann (2003)
34. Xu, B., Li, Q.: A spatial logic for modeling and verification of collision-free control of vehicles. In: ICECCS 2016, Dubai, United Arab Emirates, November 2016. Proceedings, pp. 33–42. IEEE Computer Society (2016)