International Journal on Software Tools for Technology Transfer (2024) 26:169–188
https://doi.org/10.1007/s10009-024-00740-7

GENERAL

Special Issue: FMICS 2022

Monitoring of spatio-temporal properties with nonlinear SAT solvers

André Matos Pedro[1] · Tomás Silva[1,2] · Tiago Sequeira[1] · João Lourenço[2] · João Costa Seco[2] · Carla Ferreira[2]

Accepted: 6 February 2024 / Published online: 22 February 2024

Abstract

The automotive industry is increasingly dependent on computing systems with different 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 first-order logic, where they are 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, which automatically generates a monitor from a formula, show that our approach is feasible for online monitoring in a real-world environment.

Keywords Formalization of traffic rules · Autonomous vehicles · Spatio-temporal logic · Runtime verification · Rutime monitoring · Non-linear SAT solvers

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 large impact a hazard can have [45]. 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 [40]. Challenges on the verification and validation methodologies for these systems are introduced by sub-symbolic AI methods where the decision algorithms are known to introduce non-determinism [2, 11, 14, 29].

Runtime verification (RV) is a lightweight verification method commonly used in safety-critical systems [30, 33, 48] performed during runtime and offers the possibility to act whenever a fault is observed. In RV, a formal requirement is used to automatically generate a monitor that checks whether 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 [30], to describe the evolution over time, and Modal Metric Spaces (MS), which allows us to formally reason about the surrounding space of the system [32]. 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 subject usually specify temporal and spatial features. The spatio-temporal languages (e.g., [24, 27, 34]) provide the adequate formalization and fulfillment of the ADS requirements [54], 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 focuses on monitoring 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 reference architecture, where the relations between the simulator, monitor, and vehicle can be seen. The simulator implements the scenario described using the ASAM standard [6] and the

✉ A. Matos Pedro

1 VORTEX-CoLab, Vila Nova de Gaia, Portugal

2 NOVA-LINCS, NOVA University Lisbon | FCT, Caparica, Portugal

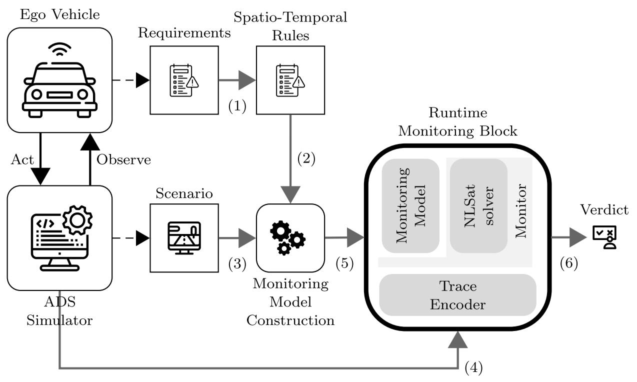


Fig. 1 Reference workflow for spatio-temporal monitoring of an autonomous (ego) vehicle

ego vehicle implements the set of requirements. Then the monitor block, which runs a solver, checks whether the requirement is met and draws a verdict. Step 1 starts with the formalization of the requirements/rules. From a micro perspective, the verification of a *LTL combined with a fragment of MS* (LTL×MS) [24] formula consists of constructing 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* (FOL$_\mathbb{R}$) constraints is performed. This engine creates a monitoring model in FOL$_\mathbb{R}$, which is interpreted by the non-linear satisfiability solver that is provided by the SMT solver Z3 [21] 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 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 behavior of an ADS, while driving at an urban road intersection, that must comply with road safety rules defined by the international Vienna convention [54]. 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 whether these legal requirements 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 FOL$_\mathbb{R}$ formula, nonlinear SAT solvers can verify its satisfiability.

Paper contributions First, we present a formalization of three traffic rules taken from the Vienna convention [54] using LTL×MS, and apply them to a traffic T-shaped junction

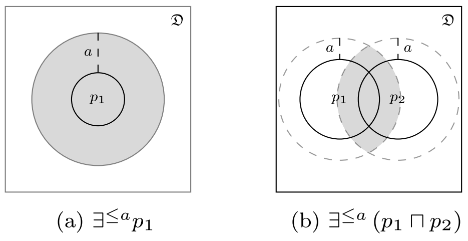

$\mathfrak{D}$
a
p_1
(a) $\exists^{\leq a} p_1$
$\mathfrak{D}$
a
a
p_1
p_2
(b) $\exists^{\leq a}(p_1 \sqcap p_2)$

Fig. 2 Distance term operators in metric space $\mathfrak{D}$

scenario. Second, we encode these rules and our scenario in FOL$_\mathbb{R}$, the language interpretable by the SMT solver Z3 [21]. Then, to encapsulate the encoding, our tool automatically generates runtime monitor blocks that can verify whether the requirements are verified in the simulated environment. Finally, we show evidence for the feasibility and scalability of online monitoring. This paper is an extended version of a previously published work [20]. It includes a detailed explanation of the language and its encoding with examples. Additionally, it presents correctness proofs for the encoding and includes an extended section on related work.

Paper structure Section 2 introduces some important concepts and definitions of the LTL×MS language. Section 3 explores the formalization of three road traffic rules using LTL×MS within the context of a T-shaped traffic junction where these rules are applicable. Additionally, the scenario is abstracted to FOL$_\mathbb{R}$. Section 4 presents the definition of the trace and its subsequent encoding to FOL$_\mathbb{R}$. Section 5 introduces the monitor generation approach, while Sect. 6 shows the feasibility of the overall monitor approach. Finally, Sects. 7 and 8 present the related work and draw conclusions and directions for future work, respectively.

2 Preliminaries

The combination of spatial logic with temporal logic has been exhaustively explored in previous works [1, 24, 25, 41]. LTL is a propositional discrete linear temporal logic suitable for model-checking reactive systems and RV [33]. In LTL, time flow is represented by a set of points that are strictly ordered by the precedence relation $<$ [23]. It primarily focuses on propositions and their sequencing. Additionally, LTL includes temporal operators such as "Until" (α **U** ω) — which asserts that α holds until ω becomes true — and "Since" (α **S** ω) — which states that α has been true since ω became true.

In the context of spatial logic, the language known as Modal Metric Spaces (MS) introduced by Kuts et al. [32] incorporates bounded distance operators, such as $\exists^{=a}$, $\exists^{<a}$, $\exists^{>a}$, and $\exists^{<b}_{>a}$. Figure 2 provides a visual depiction of $\exists^{\leq a} p_1$

and $\exists^{\leq a}(p_1 \sqcap p_2)$ in a metric space, where p_1 and p_2 represent spatial variables that are expanded by a distance of a units. Another restricted variant, $\mathrm{MS}^{\leq,<}$, focusing solely on the operators $\exists^{\leq,<}$, was proposed by Wolter and Zakharyaschev [57]. Moreover, the combination of LTL and $\mathrm{MS}^{\leq}$ was described by Aiello et al. [1, p. 545], who demonstrated the decidability of satisfiability and explored the computational complexity of this combination. Despite the richness of LTL×MS in terms of expressiveness, there is a lack of available decision procedures for spatio-temporal languages [27]. To the best of our knowledge, this work presents the first monitoring procedure for $\mathrm{LTL}\times\mathrm{MS}^{\leq}$.

Definition 1 (LTL×MS$^{\leq}$ – Syntax)
The terms and formulas are inductively defined by

$$\varrho ::= p \mid \overline{\varrho} \mid \varrho_1 \sqcap \varrho_2 \mid \varrho_1 \sqcup \varrho_2 \mid \exists^{\leq a}\varrho \mid \varrho_1\, \mathfrak{U}_{\!,}\, \varrho_2$$

$$\varphi ::= \varrho_1 \sqsubseteq \varrho_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1\, \mathcal{U}\, \varphi_2 \mid \varphi_1\, \mathcal{S}\, \varphi_2,$$

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 operator "Until" for terms and formulas, respectively. While $\mathcal{S}$ is the dual operator of $\mathcal{U}$ — the "Since" operator. Furthermore, ρ refers to a word of ϱ, and ϕ represents a word of φ.

Definition 2 (Terms Semantics)
A metric temporal model is a pair of the form $\mathfrak{M} = (\mathfrak{D}, \mathfrak{N})$ [1, p. 544], 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 $\Delta \times \Delta \mapsto \mathbb{R}_0^+$ describing the distance between every two points in Δ, satisfying the axioms identity of indiscernibles, symmetry and triangle inequality [31]. 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

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

The definition of the "Until" operator does not require that an LTL×MS term holds at n. We consider the open interval (n,m) based on how the semantics of terms are defined in [1, p. 528].

To comprehend the practicality of the remaining terms, Fig. 3 illustrates the behavior of $\mathfrak{U}_{\!,}$ while Fig. 4 illustrates the complement and the intersection operations. Now, let us take a look at an example.

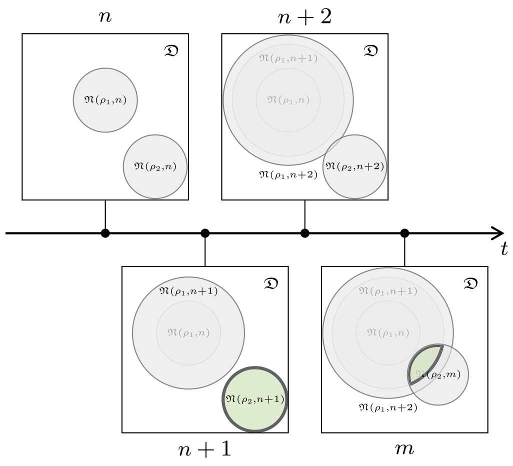


Fig. 3 Until term operator in metric space $\mathfrak{D}$

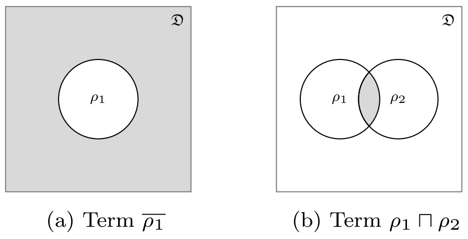


Fig. 4 Term operators in metric space $\mathfrak{D}$

Example 1
($\mathfrak{U}_{\!,}$ term) Suppose we have two terms ρ_1, ρ_2 and the "Until" set at n given by $\mathfrak{N}(\rho_1\, \mathfrak{U}_{\!,}\, \rho_2, n)$. Now, assume that ρ_2 holds for $n, n+1, n+2, n+3 = m$, ρ_2 does not grow or diminish, and ρ_2 does change its position heading towards the center. Also assume that ρ_1 holds for (n,m) (from the "Until" definition we know that ρ_1 at n does not influence the output) and that its radius is increasing between n and $n+2$ and stabilizes at m, $\mathfrak{N}(\rho_1, n+2) = \mathfrak{N}(\rho_1, m)$. According to the $\mathfrak{U}_{\!,}$ semantics, we can express $\mathfrak{N}(\rho_1 \mathfrak{U}_{\!,} \rho_2, n)$ as follows:

$$\begin{aligned}
&\mathfrak{N}(\rho_2, n+1)\\
&\cup (\mathfrak{N}(\rho_2, n+2) \cap \mathfrak{N}(\rho_1, n+1))\\
&\cup (\mathfrak{N}(\rho_2, m) \cap \mathfrak{N}(\rho_1, n+1) \cap \mathfrak{N}(\rho_1, n+2)).
\end{aligned}$$

Based on the given assumptions, it is evident that there is no intersection between $\mathfrak{N}(\rho_2, n+2)$ and $\mathfrak{N}(\rho_1, n+1)$. Additionally, both $\mathfrak{N}(\rho_2, n+1)$ and the intersection of $\mathfrak{N}(\rho_2, m)$ with $\mathfrak{N}(\rho_1, n+1)$ and $\mathfrak{N}(\rho_1, n+2)$ are not empty. When examining the motion of ρ_2 at m, it becomes apparent that it overlaps with $\mathfrak{N}(\rho_1, n+2)$. Consequently, $\mathfrak{N}(\rho_1 \mathfrak{U}_{\!,} \rho_2, n)$ is determined by the union of both regions (green) at $n+1$ and m.

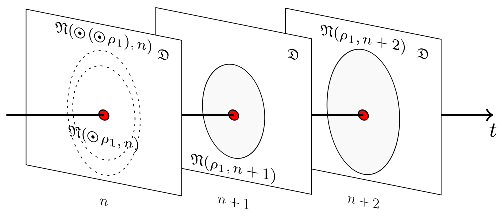

$\mathfrak{N}(\odot(\odot\rho_1),n)$
$\mathfrak{D}$
$\mathfrak{N}(\odot\rho_1,n)$
$\mathfrak{D}$
$\mathfrak{N}(\rho_1,n+1)$
$\mathfrak{N}(\rho_1,n+2)$
$\mathfrak{D}$
t
n
$n+1$
$n+2$

Fig. 5 Future projection of $\odot\rho_1$ and $\odot(\odot\rho_1)$ terms

Although the choice of using an open interval may seem like a minor detail, where coherence with the original "Until" semantics would be the only deciding factor, this subtle difference plays an important role when defining the "Next" operator. The inclusion of $\odot$ in the LTL and MS$^{\leq}$ language of terms makes the satisfiability problem undecidable, as mentioned in [1, p. 546]. Unlike, LTL×MS$^{\leq}$ that is decidable. As shown in Fig. 3, the 'Until' operator returns a union that includes the region occupied by ρ_2 at $n+1$, which is a direct consequence of utilizing an open interval (n,m).

By combining the "Until" operator with $\top$ (universe) and $\perp$ (empty set), we can define the shorthand notation called 'Next' as $\odot\rho \equiv \perp \mathfrak{U}, \rho$, and its corresponding semantics is given by

$$\begin{aligned}
&\mathfrak{N}(\odot\,\varrho,n) = \mathfrak{N}(\perp \mathfrak{U}, \varrho,n)\\
&= \mathfrak{N}(\varrho,n+1) \cup (\mathfrak{N}(\varrho,n+2)\cap\perp)\cup\cdots\cup(\mathfrak{N}(\varrho,m)\cap\perp)\\
&= \mathfrak{N}(\varrho,n+1)\cup\perp\cup\cdots\cup\perp\\
&= \mathfrak{N}(\varrho,n+1).
\end{aligned}$$

Figure 5 illustrates these future projections of a term ρ_1 and exemplifies the $\odot$ operator in the metric space $\mathfrak{D}$. The shorthand for "Eventually" and "Always" can also be defined as $\Diamond\mkern-10mu\cdot\;\varrho \equiv \top\,\mathfrak{U}, \varrho$ and $\boxdot\varrho \equiv \overline{\Diamond\mkern-10mu\cdot\;\overline{\varrho}}$, respectively. The meaning of these terms shorthand is as follows:

$$\mathfrak{N}(\Diamond\mkern-10mu\cdot\;\varrho,n) = \bigcup_{m>n}\mathfrak{N}(\varrho,m),$$

$$\mathfrak{N}(\boxdot\varrho,n) = \bigcap_{m>n}\mathfrak{N}(\varrho,m).$$

In simple terms, $\odot$ at time n represents the space occupied by ϱ at the next time step, $n+1$. For $\Diamond\mkern-10mu\cdot\;$, it means that at a given time n, term $\Diamond\mkern-10mu\cdot\;$ is considered as the union of all possible spatial extensions of ϱ after n. Similarly, for $\boxdot$, it means that at a given time n, term $\boxdot$ is seen as the common area shared by all possible spatial extensions of ϱ after n.

Definition 3 (Formulas Semantics [24])
The satisfaction relation of an LTL×MS formula φ in a model $\mathfrak{M}$ is defined as $(\mathfrak{M},n)\models\varphi$, where n is a time point in the set of natural numbers $\mathbb{N}$. The truth values of formulas in a model $\mathfrak{M}$ are defined as follows:

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

An LTL×MS formula φ is called satisfiable if there exists a model $\mathfrak{M}$ such that $(\mathfrak{M},n)\models\varphi$ holds for some time point $n\in\mathbb{N}$.

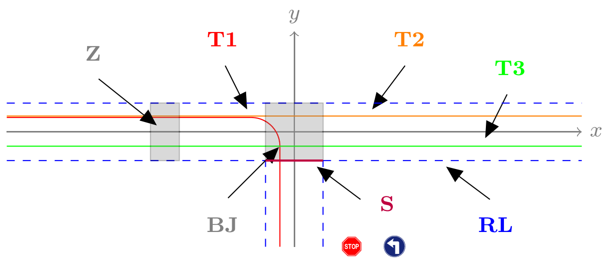
The definition of the "Until" operator in LTL×MS does not require an LTL×MS formula to hold at time point n. However, in LTL, the formula $\alpha\ \mathbf{U}\ \omega$ requires that α be either true or false at time point n. This distinction arises as the semantics of LTL×MS formulas do not provide any information about the truth value of α at n, due to the open interval (n,m). Consequently, this allows for shorthand notation using $\bigcirc$, which represents "Next" and is defined as $\bigcirc\varphi\equiv\perp\mathcal{U}\,\varphi$, similar to how it is used in the LTL×MS terms. A similar explanation can be provided for the "Since" operator.

Regarding temporal modalities, $\Diamond$ stands for "Eventually", and $\Box$ for "Always", which can be defined using $\mathcal{U}$: $\Diamond\varphi\equiv\top\,\mathcal{U}\,\varphi$, and $\Box\varphi\equiv\neg\Diamond\neg\varphi$. When talking about the past, the connectors are defined in an analogous way using $\mathcal{S}$. Thus, $\Diamond\mkern-11mu-\;\varphi\equiv\top\,\mathcal{S}\,\varphi$ for "Once", $\boxminus\varphi\equiv\neg\Diamond\mkern-11mu-\;\neg\varphi$ for "Historically" and $\ominus\varphi\equiv\perp\mathcal{S}\,\varphi$ 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 $\neg(\varrho\sqsubseteq\perp)$.

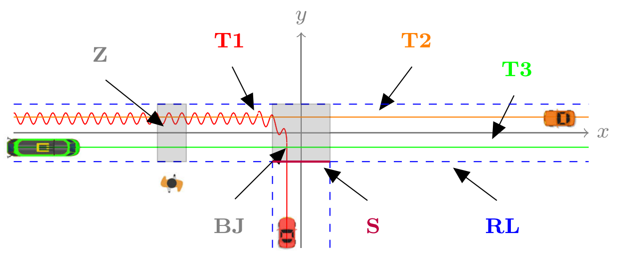
To construct complex formulas concisely, we introduce four spatial patterns that are similar to RCC8 [56]. These patterns are "Equal" – EQ,, "Disconnected" – DC, "Partially overlapped" – O, and "Strictly included" – I. We can show these patterns using := to denote "is defined", as follows:

$$\begin{aligned}
&\mathtt{EQ,}(\rho_1,\rho_2) := (\rho_1\sqsubseteq\rho_2)\wedge(\rho_2\sqsubseteq\rho_1),\\
&\mathtt{DC}(\rho_1,\rho_2) := \mathtt{EQ,}(\rho_1\sqcap\rho_2,\perp),\\
&\mathtt{O}(\rho_1,\rho_2) \quad := \neg(\mathtt{DC}(\rho_1,\rho_2))\wedge\neg(\rho_1\sqsubseteq\rho_2)\wedge\neg(\rho_2\sqsubseteq\rho_1),\\
&\mathtt{I}(\rho_1,\rho_2) \quad := (\rho_1\sqsubseteq\rho_2)\wedge\neg(\rho_2\sqsubseteq\rho_1).
\end{aligned}$$

Furthermore, the notation $\varrho_1=\varrho_2$ represents the conjunction of two formulas: $(\varrho_1\sqsubseteq\varrho_2)\wedge(\varrho_2\sqsubseteq\varrho_1)$. Similarly, $\varrho_1\neq\varrho_2$ stands for $\neg(\varrho_1\sqsubseteq\varrho_2)\vee\neg(\varrho_2\sqsubseteq\varrho_1)$.



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



(b) Sketch of an unsatisfiable model with actors and the oscillating Ego vehicle (C – red) trajectory (other vehicle C' – orange).

Fig. 6 Running example: An urban T-shaped junction scenario (Color figure online)

2.1 Encoding language FOL$_\mathbb{R}$

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 $\mathbb{P}$ denotes the set of real variables in FOL$_\mathbb{R}$. To utilize the existing decision procedure for FOL$_\mathbb{R}$, we convert LTL×MS into FOL$_\mathbb{R}$.

3 Running example

The depicted scenario in Fig. 6a captures a specific traffic situation studied in this work. It shows a T-shaped junction where vehicle **C** approaches from a one-way road and encounters a stop sign at the intersection. The objective is for vehicle **C** to enter a bi-directional road, which is occupied by a tram and another car referred to as **C′**. At the intersection, there is a box junction, where stopping is strictly prohibited according to the Vienna convention on road traffic. Additionally, a pedestrian zebra crossing **Z** is present on the bi-directional road. The scenario also includes three distinct solid lines denoted as **T1** (red), **T2** (orange), and **T3** (green), representing the reference trajectories that vehicles can follow in this particular running example. It is assumed that all actors are always present in the scenario during the trace execution. This assumption prevents the possibility of a car disappearing in the middle of its trajectory and suggests that the distance is infinite. While there is no problem with this, if any sequence in the middle is omitted, a formula stating that a car should be within a certain distance will always be false. However, there is no issue with allowing other actors to enter the scenario.

The objective of this running example is to validate the proposed approach for autonomous (ego) vehicles. We begin by formalizing the traffic rules using LTL×MS and preprocessing them accordingly. Next, we introduce the encoding of the traffic scenario itself, which primarily describes static objects in the environment. Subsequently, we describe the meaning of a trace, which captures the dynamic objects within the scenario. It is important to note that a scenario focuses on static entities such as trajectories, and traffic signs (e.g., crosswalks, stop signs), while a trace specifically captures the dynamic aspects of objects (e.g., position, size, shape) such as pedestrians, cyclists, and vehicles.

Now, let us analyze a set of rules that take into account safety measures. To ensure safety, an ego vehicle must follow a reference trajectory when approaching a crossing area and keep at most one meter away from that trajectory. This can be expressed in the LTL×MS language as follows:

$$\Box\left(\mathsf{O}\left(\mathbf{T1}, \exists^{\leq 1}\mathbf{C}\right)\right), \tag{1}$$

where **T1** corresponds to the reference trajectory, and **C** to the ego vehicle.

The model shown in Fig. 6b fails to satisfy (1) as the ego vehicle's oscillation along the reference trajectory **T1** exceeds the acceptable threshold of one meter.

3.1 Formalization of road traffic rules with LTL×MS

According to the Vienna convention [54], road traffic rules describe how pedestrians and vehicles should behave in a street environment. Without loss of generality, we identify three specific rules of interest to describe in the LTL×MS logic. 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 [54]) while following its trajectory, and (b) a vehicle should maintain a safety-margin from the vehicle in front of it. In LTL×MS, the (a) part of this rule can be described by

$$\neg\Diamond\left(\mathsf{O}\left(\mathbf{RL}, \exists^{\leq 1}\mathbf{C}\right)\right), \tag{2}$$

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 $\Diamond$, we arrive at the following expression:

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

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

$$\neg\Diamond\Big(\mathtt{O}\Big(\exists^{\leq 2}\mathbf{C}',\exists^{\leq 2}\mathbf{C}\Big)\Big),\tag{4}$$

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

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

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 [54]; (c) tram rails, based on article 23 al.3 [54].

Regarding part (a), 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 same position as it was in the previous moment. Writing in LTL×MS we have:

$$\Box\big(\mathtt{I}(\mathbf{C},\mathbf{BJ})\vee\mathtt{O}(\mathbf{C},\mathbf{BJ})\rightarrow\neg\mathtt{EQ}_,(\mathbf{C},\odot\mathbf{C})\big),\tag{6}$$

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 $\Box$ operators,

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

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

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

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 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

$$\begin{aligned}\Box\Big[\Big(&\mathtt{O}\Big(\mathbf{S},\exists^{\leq 1}\mathbf{C}\Big)\wedge\neg\diamondminus\,\mathtt{EQ}_,(\mathbf{C},\odot\mathbf{C})\Big)\\&\rightarrow\Diamond\Big(\mathtt{EQ}_,(\mathbf{C},\odot\mathbf{C})\wedge\mathtt{DC}(\mathbf{S},\diamonddot\,\mathbf{C})\Big)\Big],\end{aligned}\tag{8}$$

where **S** is the location of the stop sign. Starting from the expansion of $\odot$, $\diamonddot$, $\Diamond$, $\diamondminus$, $\mathtt{O}$, $\mathtt{EQ}_,$, $\mathtt{DC}$, and $\Box$ operators, we obtain

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

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 encoding of our running example.

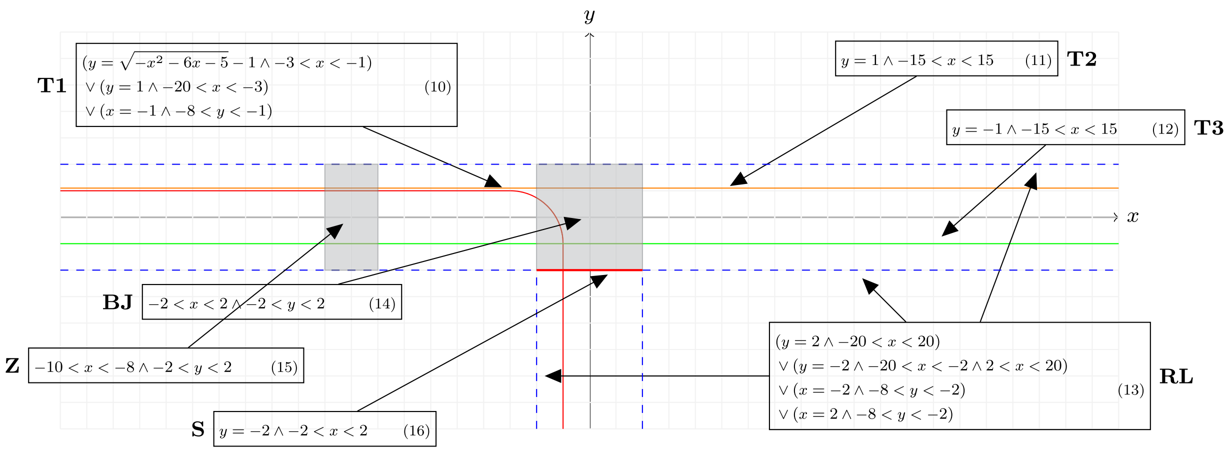
3.2 Scenario encoding with $\mathrm{FOL}_{\mathbb{R}}$

In our current example, the objects can be divided into two distinct categories: stationary objects like road limits, crosswalks, and box junctions, and dynamic objects capable of changing their position and shape over time, such as vehicles, trams, or pedestrians.

Figure 7 depicts the encoding for each static object found in the model illustrated in Fig. 6b, which is based on region constraints represented as inequalities. The trajectories and road limits are represented by line segments, expressed as sets of linear and non-linear polynomials. The box junction and crosswalk are defined by bounding boxes. The reference trajectory for vehicles and trams is described by three different line segments: (10), (11), and (12). The system of equations (13) represents the road limits of the scenario. The box junction, crosswalk, and stop sign areas are defined by (14), (15), and (16), respectively (see Fig. 7).

For dynamic objects, a continuous trace is necessary to track their positions at every time step. The trace is transmitted from the simulator as a tree data structure and translated into formulas written in $\mathrm{FOL}_{\mathbb{R}}$.

Now, let us turn our attention to the definition and encoding of finite and infinite traces.



y
x
T1
$(y = \sqrt{-x^2 - 6x - 5} - 1 \wedge -3 < x < -1)$
$\vee (y = 1 \wedge -20 < x < -3)$ (10)
$\vee (x = -1 \wedge -8 < y < -1)$
$y = 1 \wedge -15 < x < 15$ (11)
T2
$y = -1 \wedge -15 < x < 15$ (12)
T3
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. 7 Encoding of scenario as spatial variables: T1, T2, T3, RL, BJ, Z, and SL

4 Traces and trace encoder

In this section, we describe the trace encoding process and the construction of the evaluation function for observing spatial propositions based on traces. We begin by outlining the trace encoding definitions, which involve defining inductively and co-inductively the structure of traces. Subsequently, we introduce auxiliary functions that facilitate the efficient storage of constraint sets associated with the traces. Finally, we present the evaluation function.

For ease of encoding, an inductive list can be algebraically defined as a finite trace — a symbolic representation of sequences with predetermined lengths.

Definition 4 (Finite Trace)
A finite trace forms the set $A^{[0,n]} = \{\sigma : [0,n] \mapsto A\}$, where $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ defines a sequence of symbols with length n.

We can define an inductive list, denoted as $List(A)$, to represent a finite sequence of elements from the set A. The definition is as follows: $List(A) = Nil$ represents an empty list, denoted by Nil, indicating the absence of any elements (base case); and $List(A) = Cons(a, l)$ represents a non-empty list, where a is the first element of A and l is the remaining list (inductive case). Next, we can define the functions for retrieving the "head", "tail", and the "n-th" element, as follows:

$$hd : A^{[0,n]} \mapsto A \qquad tl : A^{[0,n]} \mapsto A^{[0,n]}$$

$$hd\ \big(Cons(a,l)\big) := a \qquad tl\ \big(Cons(a,l)\big) := l$$

$$nth : \mathbb{N}_0 \times A^{[0,n]} \mapsto A$$

$$nth\ (n,l) := \text{if } n > 1 \text{ then } nth\ \big(n-1, tl\ (l)\big) \text{ else } hd\,(l)$$

where nth returns the "n-th" element of a reversed list starting at position 1 (head of the reversed list).

Now, let us redirect our attention to the case of infinite traces. Unlike finite traces, which represent sequences with predetermined lengths, infinite traces are utilized when observing state machines that do not have a predefined duration and can exhibit infinite behavior. In practical terms, infinite traces are advantageous when symbols need to be incrementally evaluated during observation, as is the case with our monitoring approach. An infinite trace can be algebraically defined as a co-inductive stream allowing for the sequential observation of its elements through the iterative application of the head and tail operations. This lazy evaluation ensures that these elements are computed as needed.

Definition 5 (Infinite Trace)
An infinite trace forms the set $A^{\mathbb{N}_0} = \{\sigma : \mathbb{N}_0 \mapsto A\}$, where $(\sigma_0, \sigma_1, \sigma_2, \ldots)$ defines an unbounded sequence of symbols.

We can now define a co-inductive stream, denoted as $Stream(A)$, to represent an infinite sequence of elements from the set A, as follows: $Stream(A) = Nil$ represents an empty stream, where Nil means the absence of any elements (base case); $Stream(A) = Cons(a, s)$ represents a non-empty stream, where $a \in A$ is the current element and s is a function that computes the tail of the stream (co-inductive case).

To maintain a clear distinction between infinite and finite traces, we will substitute the terms "head" and "tail" with the terms "now" and "next" respectively. The definitions of the *now* and *next* functions are as follows:

$$now : A^{\mathbb{N}_0} \mapsto A \qquad next : A^{\mathbb{N}_0} \mapsto A^{\mathbb{N}_0}$$

$$now\ \big(Cons(a,s)\big) := a \qquad next\ \big(Cons(a,s)\big) := s\ ()$$

where *now* gets the current symbol in the sequence and *next* computes the next sequence of symbols.

To efficiently handle the concept of history, we introduce the co-inductive stream with history. This stream shares similarities with a regular co-inductive stream, but preserves the

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

Fig. 8 An example of a symbol $s \in A$ represented in JSON format, which includes a circular object (id = 1) with a center at $(0.5, -0.5)$ and a radius of 0.5 at $t = 1$

history of previous elements in addition to the current and future elements. This feature enables efficient access to past elements without the need for re-computation. In this context, we define a trace as the triple (k, l, s), where $k \in \mathbb{N}_0$ represents the head of the history list in the finite trace $l \in A^{[0,n]}$, and $s \in A^{\mathbb{N}_0}$ represents the infinite stream.

We also need to redefine *now* and *next* to incorporate the concept of history while also incorporating the dual of "next". The updated definitions are as follows:

$$
\begin{aligned}
&now_h : (\mathbb{N}_0 \times A^{[0,n]} \times A^{\mathbb{N}_0}) \mapsto A\\
&now_h\ (n,l,s) := \text{ if } n > 0 \text{ then } nth\ (n,l)\\
&\quad \text{else } now\ (s)\\
&next_h : (\mathbb{N}_0 \times A^{[0,n]} \times A^{\mathbb{N}_0}) \mapsto (\mathbb{N}_0 \times A^{[0,n]} \times A^{\mathbb{N}_0})\\
&next_h\ (n,l,s) := \text{ if } n > 0 \text{ then } (n-1,l,s)\\
&\quad \text{else } (n, l\ ::\ now_h(s), next(s))\\
&prev : (\mathbb{N}_0 \times A^{[0,n]} \times A^{\mathbb{N}_0}) \mapsto (\mathbb{N}_0 \times A^{[0,n]} \times A^{\mathbb{N}_0})\\
&prev\ \left(n,l,s\right) := \text{ if } n > 0 \text{ then } (n-1,l,s) \text{ else } (n,l,s)
\end{aligned}
$$

Now, let us assume that a trace is formed by a sequence of symbols, where each symbol of kind $\mathcal{R}$ corresponds to a list of objects $List(\mathcal{R})$. This correspondence is illustrated in the example JSON trace shown in Fig. 8. We opted to store the trace data from CARLA in JSON format for practical purposes. This allowed us to take advantage of the built-in libraries available in Python. To encode symbols, we extract the list of objects from the trace and generate inequality constraints that define each object, while the hash map stores the sets of inequalities for each spatial proposition (object).

The auxiliary definitions for adding constraints and retrieving them are as follows:

$$
\begin{aligned}
&add : (\mathfrak{P} \times \mathbb{L}) \times (\mathfrak{P} \mapsto \mathbb{L}) \mapsto (\mathfrak{P} \mapsto \mathbb{L})\\
&find : \mathfrak{P} \times (\mathfrak{P} \mapsto \mathbb{L}) \mapsto \mathbb{L}\\
&add_{\mathcal{H}} : (\mathfrak{P} \times \mathbb{L}) \mapsto \mathbb{B}
\end{aligned}
$$

$$add_{\mathcal{H}}\ (p,u) := \mathcal{H} \leftarrow add\ (p,u)\ \mathcal{H}; 1,$$

where *add* stores constraints in the hash map $\mathcal{H} : \mathfrak{P} \mapsto \mathbb{L}$, and *find* returns the constraints of a given spatial proposition. To reduce verbosity, we assume that the variable $\mathcal{H}$ is shared across multiple invocations of the function $add_{\mathcal{H}}$ and that these functions are not pure.

The *enc* function The definition of the *enc* function needs an auxiliary function *enco*. The *enc* function gets as input a symbol (list of objects), produces the constraints and adds them to the hash map $\mathcal{H}$. These functions are recursively defined by

$$
\begin{aligned}
&enco : \mathcal{R} \mapsto \mathbb{B}\\
&enco(id,p,r) := \ \texttt{if}\ r.type = \texttt{circle}\ \texttt{then}\ add_{\mathcal{H}}\ (id,\\
&\quad \texttt{let}\ (\texttt{x1}\ p.x)\ (\texttt{x2}\ p.y)(\texttt{x3}\ r.radius).\ obj\ (\texttt{circle})\\
&\ )\ \texttt{else}\ (\ \texttt{if}\ r.type = \texttt{bbox}\ \texttt{then}\ add_{\mathcal{H}}\ (id,\\
&\quad \texttt{let}\ (\texttt{x1}\ p.x)\ (\texttt{x2}\ p.y)(\texttt{x3}\ r.w)\ (\texttt{x4}\ r.h).\ obj\ (\texttt{bbox})\\
&\quad)\ \texttt{else}\ 0\ )\\
&enc : List(\mathcal{R}) \mapsto \mathbb{B}\\
&enc(l) := \ \texttt{if}\ l = Nil\ \texttt{then}\ 1\ \texttt{else}\ o \leftarrow hd(l);\\
&\ enco\ (o.id, o.position, o.region)\ \texttt{and}\ enc\ (tl(l)).
\end{aligned}
$$

We can see how the resultant expressions can use the variables binder `let`. The evaluation of the expression

$$\texttt{let}\ ((\texttt{x1}\ 1)\ (\texttt{x2}\ 2)\ (\texttt{x3}\ 3)).(\texttt{x4} - \texttt{x1})^2 + (\texttt{x5} - \texttt{x2})^2 < \texttt{x3}^2$$

results in $(\texttt{x1} - 1)^2 + (\texttt{x2} - 2)^2 < 3^2$, where `x1` and `x2` are the (renamed) remaining free variables. Later, we can bind these variables with a quantifier, for instance, such as

$$\forall (x1\ x), (x2\ y).\ (\texttt{x1} - 1)^2 + (\texttt{x2} - 2)^2 < 3^2,$$

where $(1, 2)$ is the center point of the circle and 3 the radius. This is the way we replace free variables.

The *obj* function Without loss of generality, let us consider a circle to be equivalent to a ball and a bounding box to be a rectangle or square in two-dimensional Euclidean space. It is important to note that while other geometric shapes can be translated, they fall outside the scope of our current running example. As the function *obj* is currently undefined, the function $obj : id \mapsto \mathbb{L}$ is responsible for generating object constraints that define circles and bounding boxes using free variables, where $id \in \texttt{circle}, \texttt{bbox}$.

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

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

Fig. 9 Conversion functions $\mathbf{conv}_{\varrho}$ and $\mathbf{conv}_{\varphi}$

For example, the function can be defined as follows:

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

It is worth mentioning that the *enco* and *obj* functions must have different versions that support different shapes and dimensions, such as the three-dimensional Euclidean space. Although these functions are dependent on the specific type of trace being used, they can be readily replaced to accommodate different cases.

The evaluation function Since our objective is to construct an evaluation function for observations of spatial propositions based on traces, trace encoding involves identifying the set of constraints stored in the hash map $\mathcal{H}$ by function *enco*. As these constraints are derived from the input trace, we can now define the evaluation function. In general terms, the trace encoding consists in the construction of the function

$$eval : \mathfrak{P} \mapsto \mathbb{L}$$
$$eval\ (p) := find\ p\ \mathcal{H},$$

which evaluates a spatial variable p to an expression in $\mathrm{FOL}_{\mathbb{R}}$ in the $\mathcal{H}$ hash map.

In practice, the trace is translated into an intermediate representation that can be interpreted and solved by a satisfiability solver. Next, we will discuss the transformation of formulas using two different approaches.

5 Monitoring model construction

Our algorithm takes as input an LTL×MS property, which represents a requirement under analysis, and generates a model in $\mathrm{FOL}_{\mathbb{R}}$. It is important to keep in mind that the languages of terms and formulas in LTL×MS are denoted by the sets $\mathfrak{T}$ and $\mathfrak{F}$, respectively.

Each term $\rho \in \mathfrak{T}$ (i.e., the set of all words in ϱ) is translated into $\mathrm{FOL}_{\mathbb{R}}$ using the recursive function $\mathbf{conv}_{\varrho} : \mathfrak{T} \mapsto \mathbb{L}$. The details of this function can be found in Fig. 9. Additionally, we utilize the function $dist : \mathbb{R} \times \mathfrak{T} \mapsto \mathbb{L}$ to apply Property 1. It states that any formula containing distance operators can be transformed into an equivalent formula where the distance operators are applied exclusively to the spatial propositions. For the sake of simplicity, we omit the explicit definition of the recursive function *dist* in this context.

Property 1 (Distance Operator Distribution)
Let ρ be a term, V be the set of free variables in ρ, and e be a rational number. For any ρ and e, the distance operator $\exists^{\leq e}\rho$ can be expressed equivalently as $\exists^{\leq e}a$ for every free variable $a \in V$.

The function $next_{\varrho} : \mathfrak{T} \mapsto \mathbb{L}$ also exhibits a similar distributive property as the distance operators, but instead of distance, it assigns to each proposition the successor (a nested structure of next operators applied exclusively to propositions).

To conclude the conversion function over terms, $\mathbf{conv}_{\varrho}$, we introduce a function called *unfold* : $\mathfrak{T} \times \mathfrak{T} \mapsto \mathbb{L}$. This function generates a limited instance (since our trace has a finite length of n) of the sequence

$$\mathbf{conv}_{\varrho}(\odot\rho_2) \vee \bigvee_{j=2}^{m-1} \left[\mathbf{conv}_{\varrho}(\underbrace{\odot \ldots \odot}_{j\times} \rho_2) \wedge \bigwedge_{i=1}^{j-1} \mathbf{conv}_{\varrho}(\underbrace{\odot \ldots \odot}_{i\times} \rho_1) \right],$$

where ρ_1, ρ_2 are the terms of the 'Until' and $m = n$.

Lemma 1 (Correctness of the monitoring procedure for LTL×MS$^{\leq}$ terms)
For any term ρ and any time point k, there exists an $l_k \in A^{[0,n]}$ such that $\mathfrak{R}(\rho, k) = l_k$ iff $\mathbf{conv}_{\varrho}(\rho)$.

Lemma 1 shows the derivation of the $unfold$ expression. The full proof sketch can be found in the Appendix.

Let us now turn our attention to formulas. Each formula $\phi \in \mathfrak{F}$ (representing the set of all words in φ) is translated using the function $\mathbf{conv}_{\varphi} : \mathfrak{F} \mapsto \mathbb{L}$, as defined in Fig. 9. To bind all the remaining free variables in the resulting $\text{FOL}_{\mathbb{R}}$ expression, we use the expression

$$\forall \mathbf{x} \in \mathbb{R}^k.(\mathbf{conv}_{\varrho}(\rho_1) \to \mathbf{conv}_{\varrho}(\rho_2)),$$

where $k \in \mathbb{N}$. As an example, consider the formula $\forall(x,y) \in \mathbb{R}^2.\ x^2 + y^2 < 1^2 \to x^2 + y^2 < 3^2$. This formula states that for all points (x,y), if the point is located inside a circle of radius 1 centered at $(0,0)$, then it is also located inside a larger circle of radius 3 with the same center $(0,0)$.

The function $next_{\varphi} : \mathfrak{F} \mapsto \mathbb{L}$ generates a formula ϕ from the next instance, while $previous_{\varphi} : \mathfrak{F} \mapsto \mathbb{L}$ generates a formula ϕ from the previous instance.

Next, the function $unfold_X : \mathfrak{F} \times \mathfrak{F} \mapsto \mathbb{L}$ generates a bounded instance (given that our trace has a finite length of n) of the sequence

$$\mathbf{conv}_{\varphi}(X\phi_2) \vee \bigvee_{j=2}^{m-1} \left[\mathbf{conv}_{\varphi}(\underbrace{X \ldots X}_{j\times}\phi_2) \wedge \bigwedge_{i=1}^{j-1} \mathbf{conv}_{\varphi}(\underbrace{X \ldots X}_{i\times}\phi_1) \right],$$

where ϕ_1, ϕ_2 are the formulas of the "Until" and $m = n$.

Lemma 2 (Correctness of the monitoring procedure for LTL×MS$^{\leq}$ formulas)
For any formula ϕ and any time point k, there exists an $l_k \in A^{[0,n]}$ such that $\mathfrak{M}(\phi, k)$ iff $\mathbf{conv}_{\varphi}(\phi)$.

Lemma 2 describes the full derivation of the $unfold_X$ expression, which can be found in the Appendix. Additionally, the function $unfold_{\mathcal{U}} : \mathfrak{F} \times \mathfrak{F} \to \mathbb{L}$ is defined as $unfold_X$ when $X = \bigcirc$. Similarly, the function $unfold_{\mathcal{S}} : \mathfrak{F} \times \mathfrak{F} \to \mathbb{L}$ is defined as $unfold_X$ when $X = \ominus$.

To complete the encoding process, we need to address the missing component: the encoding of the entire trace, both for finite and infinite cases. The encoding of the trace primarily involves constructing the function $eval$ as previously defined. This function replaces spatial variables with expressions in $\text{FOL}_{\mathbb{R}}$, but these expressions must be obtained from the trace and stored in the map function. In addition to the evaluation function, we have also defined the enc function, which encodes objects of a symbol and stores them.

5.1 Unroll method for finite traces

To address the finite case, we introduce the function $encode$, which recursively constructs sets of constraints for a given finite trace, as follows:

$$encode : A^{[0,n]} \times \mathbb{N}_{>0} \mapsto \mathbb{L}$$

$$encode(t,n) := \text{if } n > 0 \text{ then } enc(c(hd(t))) \texttt{ and }$$
$$encode(tl(t), n-1) \text{ else } 1,$$

where $c : A \mapsto List(\mathcal{R})$ is defined by $c\ (s) := s.objects$. It is important to note that the definition of the function $encode$ relies on the enc function. The purpose of the enc function is to encode the valuations of spatial variables using sets of constraints in $\text{FOL}_{\mathbb{R}}$. During the encoding process, the conversion of formulas and terms into incomplete $\text{FOL}_{\mathbb{R}}$ expressions (as shown in Fig. 9) acts as an intermediate step.

Consider a trace, denoted as trc, with a length of n, and ϕ representing a formula in LTL×MS. To begin the encoding, we execute the following operation:

$$encode\ (trc, n)$$

This encodes all the objects contained within the symbols of the trace. Subsequently, we incorporate all the sets of constraints (as illustrated in Fig. 7) into the hash map $\mathcal{H}$. Finally, we combine the encoding of the trace and the formula using the $inline$ function. This function performs the necessary replacement of the trace objects into the structure of the encoded formula. The $inline$ function is defined as $inline : \mathbb{L} \times (\mathfrak{P} \mapsto \mathbb{L}) \mapsto \mathbb{L}$. It is worth noting that this inlining also binds every free variable of $\mathbf{conv}_{\varphi}(\phi)$ in $\mathcal{H}$ that results as an effect from computing $encode(trc, n)$. We proceed with the inlining process by computing

$$inline\ (\mathbf{conv}_{\varphi}(\phi), \mathcal{H})$$

Here, $\mathcal{H}$ comprises the sets of inequalities that represent both the scenario and the trace. By performing the inlining, we merge the formula $\mathbf{conv}_{\varphi}(\phi)$ with the information contained in $\mathcal{H}$.

While this encoding requires solving the result of the $inline$ function only once with the Z3 [21] Non-Linear Satisfiability solver, we can see the trace acting as the state machine model — a single finite execution of a state machine — and the formalization of rules and the scenario, the specification. In broad terms, it is worth noting that a monitor implements the specification and enables the observation of a state machine's execution.

To enable incremental observation, we have devised an alternative method that generates multiple monitoring model constraints to be solved incrementally instead of resolving them all at once.

5.2 Incremental method for finite and infinite traces

In this incremental method, we no longer need to unfold temporal operators like $unfold_{\mathcal{U}}$ and $unfold_{\mathcal{S}}$. Instead, we

Fig. 10 Encoding of **incr** function

$$\mathbf{incr}(\phi,\Sigma,\mathsf{s}) := \begin{cases} \mathit{solve}\Big(\mathbf{conv}_{\varphi}(\widehat{\phi}_1), \mathit{enc}(c\ (\mathit{now}\ (\Sigma)))\Big) & \text{if } \phi = \widehat{\phi}_1 \\ \mathit{implies}\Big[\mathbf{incr}\Big(\phi_1,\Sigma,\mathsf{s}\Big), \mathbf{incr}\Big(\phi_2,\Sigma,\mathsf{s}\Big)\Big] & \text{if } \phi = \phi_1 \rightarrow \phi_2 \\ \mathit{ite}\Big[\mathsf{s} = \mathfrak{u}, \mathbf{incr}\Big(\Box\phi_1, \mathit{next}(\Sigma), c_{\top}\big(\mathsf{s}, \mathbf{incr}(\phi_1, \mathit{next}(\Sigma), \mathfrak{u})\big)\Big), \mathfrak{f}\Big] & \text{if } \phi = \Box\phi_1 \\ \mathit{ite}\Big[\mathsf{s} = \mathfrak{u}, \mathbf{incr}\Big(\boxminus\phi_1, \mathit{prev}(\Sigma), c_{\top}\big(\mathsf{s}, \mathbf{incr}(\phi_1, \mathit{next}(\Sigma), \mathfrak{u})\big)\Big), \mathfrak{f}\Big] & \text{if } \phi = \boxminus\phi_1 \end{cases}$$

assume that temporal terms are bounded while temporal formulas are unbounded. This eliminates the requirement for both bounded terms and formulas, as in the unroll method. To process the temporal part of the algorithm, we employ incremental evaluation techniques, utilizing push and pop operators within a nonlinear satisfiability solver.

During evaluation, we take into account the known temporal structure that exists within the property being analyzed. Several known patterns can be considered. Some of these patterns include:

$$\Box\widehat{\phi},\quad \Box(\widehat{\phi}_1 \rightarrow \Diamond\widehat{\phi}_2),\qquad \Diamond(\widehat{\phi}_1 \wedge \Box\widehat{\phi}_2),$$
$$\Diamond\widehat{\phi},\quad \Box(\widehat{\phi}_1 \rightarrow \neg\Diamond\widehat{\phi}_2),\quad \Diamond(\widehat{\phi}_1 \wedge \Diamond\neg\widehat{\phi}_2).$$

Given a formula $\phi \in \varphi$, a formula without temporal operators $\widehat{\phi}$, an infinite trace Σ with *next*, *prev* operators, and a symbol $\mathsf{s} \in \mathfrak{S}$, we can define **incr** as shown in Fig. 10, where $c : A \mapsto \mathit{List}(\mathcal{R})$ returns the current objects ($c\ (s) := s.\mathit{objects}$). The s acts as a state indicator, representing true ($\mathfrak{t}$), false ($\mathfrak{f}$), or unknown ($\mathfrak{u}$). $\mathfrak{S}$ denotes the set $\{\mathfrak{t},\mathfrak{f},\mathfrak{u}\}$, $\mathit{ite} : \{\mathfrak{t},\mathfrak{f}\} \times \{\mathfrak{t},\mathfrak{f}\} \times \{\mathfrak{t},\mathfrak{f}\} \mapsto \{\mathfrak{t},\mathfrak{f}\}$ defines the if-then-else function, $\mathit{implies} : \{\mathfrak{t},\mathfrak{f}\} \times \{\mathfrak{t},\mathfrak{f}\} \mapsto \{\mathfrak{t},\mathfrak{f}\}$ implements the implication, and $c_{\top} : \mathfrak{S} \times \{\mathfrak{t},\mathfrak{f}\} \mapsto \mathfrak{S}$ converts the pair $(\mathfrak{u},\mathfrak{f})$ to $\mathfrak{f}$ and $\mathfrak{u}$ otherwise.

Property 2 (Spatial Isolation on ϱ terms)
For any assignment ρ that includes a spatial variable $a \in \mathfrak{P}$, ρ cannot alter the valuation of a.

From Property 2, we conclude that terms have no effect on spatial variables and this simplifies the evaluation of the *implies* function ("and" and "not" can be constructed using *implies*) in the incremental evaluation function **incr**. Furthermore, a spatial variable maintains its form regardless of where it is evaluated at a given time instant.

It is important to note that the computation of **incr**($\Box\,\phi$) has two truth values: *false* or *unknown*. Similarly, **incr**($\Diamond\,\phi$) has *true* or *unknown*, while pattern **incr**($\Box\,(\phi_1 \rightarrow \Diamond\,\phi_2)$) has the same values of **incr**($\Box\,\phi$).

Function $\mathit{solve} : \mathbb{L} \times \mathbb{L} \mapsto \{\mathfrak{t},\mathfrak{f}\}$ solves an expression in $\mathrm{FOL}_{\mathbb{R}}$ assuming another expression in $\mathrm{FOL}_{\mathbb{R}}$. It is important to note that the $\bigcirc$ temporal operator vanishes and is not incrementally evaluated and that the explicit definition of the "until" operator is missing. It has come to our attention that the version we are currently describing here is partially complete, but sufficient to encode our rules as well as those that adhere to similar patterns in practical scenarios. Roughly speaking, we can perform the incremental evaluation on any formula that is constructed inductively by

$$\varphi ::= \widehat{\phi}_1 \mid \varphi \rightarrow \varphi \mid \Box\varphi \mid \boxminus\varphi,$$

where "and" and "not" can be constructed using *implies*, and $\Diamond\,\varphi$ and $\Diamond\!\!\!\!\!-\,\varphi$ can be constructed using $\Box\,\varphi$ and $\boxminus\,\varphi$.

The incremental method isolates the temporal formulas from the solver and handles them using a separate algorithm (see Fig. 10). In specific cases, this approach enhances monitoring speed by reducing the problem space sent to the solver. Additionally, it allows for estimation techniques based on vehicle velocity or other parameters from the CARLA simulator, such as acquisition rate or other synchronization mechanisms. For instance, when monitoring slow cars with bounded temporal terms in the property of interest, as is the case with our three rules, we can dynamically modify the acquisition rate using runtime information. Consequently, there is no need for the solver to consider the whole temporal aspect of the formulas, which deals well with systems that run forever.

6 Empirical evaluation

The monitoring process is designed to run in parallel with the ADS under test, without directly interfering with the system itself, as depicted in Fig. 1. The workflow revolves around simulating a specific scenario that provides observations to the ADS. The ADS reacts to these observations and produces actions for the agents running on the simulator in a closed loop. The monitor receives the observations from the simulator in the form of a trace, which is then checked against a property to determine whether it is satisfied or not.

The evaluation of traces and scenarios was conducted on an i5-8365U CPU running Linux 5.10.11. The traces were generated using a simulated T-shaped junction scenario in the CARLA 0.9.13 autonomous driving simulator [22]. Scalability is an important aspect to consider, as the size of the trace can significantly impact monitoring performance. Therefore, we tested each property with different trace sizes to assess the performance of different methods.

Table 1 Table displaying the evaluation results. The first two columns indicate the considered rules and traces, where $|\varrho|$ denotes the number of temporal terms, $|\varphi|$ denotes the number of formulas, and $|\Sigma|$ denotes the length of the trace. The last two columns, unroll and incremental methods, show the time (in seconds) and the memory (in MB) used by the solver, the overall runtime the monitor takes to execute (RT) and frames per second (FPS)

	Rule($\|\varrho\|, \|\varphi\|$)	Trace Id($\|\Sigma\|$)	Unroll				Incremental			
			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	65.00	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	56.00
	2.a (1,3)	e5(13)	0.18	3.53	0.23	31.71	0.17	2.83	0.33	26.00
	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	**45.90**	0.12	3.57	0.27	**50.10**
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	116.80
	1.b (1,1)	s1(243)	0.15	5.14	0.45	405.00	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	9.00
	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	5.10
	Average		226.84	37.91	239.96	**104.40**	3.85	4.32	8.42	**71.10**

During the empirical evaluation, comparing hand-built sample traces (e1–e10), the Incremental method demonstrated better performance on average compared to the Unroll method. However, there were some exceptions, such as in rule 1.b, where the Unroll method performed slightly better but with higher memory consumption (as shown in Table 1). It is worth highlighting that the Incremental method demonstrated superior performance compared to the Unroll method, particularly in rules 1.a and 3. This resulted in a shorter average execution time of 0.12 seconds and lower memory usage of $3.57\,MB$ by the solver. It demonstrated faster run time (RT) of $0.27\,s$ and achieved higher frames per second (FPS) values of 50.1 on average.

Similar observations were made during the simulator evaluation (traces obtained from the simulation environment, denoted as s1–s6), but the differences were more pronounced. In the case of rule 3, the solver in the Unroll method took approximately 85 times longer than the solver in the Incremental method, with execution times of $1044.16\,s$ and $12.25\,s$, respectively. Moreover, the Unroll method showed considerably higher memory usage (average of $37.91\,MB$) compared to the Incremental method (average of $4.32\,MB$).

Considering the average value of the Incremental method (greater than 60), our approach is capable of comfortably working with modern cameras operating at a frame rate of $60\,Hz$, even though ADSs cameras typically have lower frame rates. An interesting aspect to note is that our performance measurements are independent of different resolutions. This is because our approach does not rely on the size of the image matrix, allowing it to maintain consistent performance regardless of the resolution used.

In summary, the data presented in Table 1 demonstrates the advantage of the Incremental method over the Unroll method. The tool and documentation for artifact evaluation can be found at the following link: https://github.com/anmaped/stem-binaries.git.

6.1 Integration with our simulation-based test bed

The simulation test bed we have developed allows us to integrate our stem tool into real-world testing of ADS functionalities. An overview of how the simulation appears in the CARLA simulator is shown in Fig. 11, although pedestrians crossing the street are barely visible in the image. For test deployment, our test bed utilizes a service-based, containerized approach. The configurations for the services to be deployed are stored as Kubernetes deployment files, which are combined with test scenario files to form complete test batches.

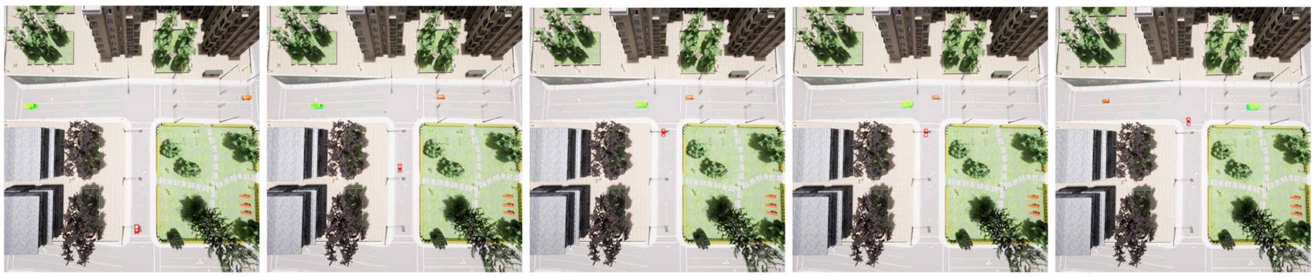


Fig. 11 Frame sequence of one CARLA run with two vehicles and one Ego vehicle using our running example

```
(property (always (not (overlap (prop "/vehicle/orange") (prop "/vehicle/red") ) ) ) )
(property (not (eventually (overlap (expand (prop "/vehicle/orange") 1. ) (expand (prop "/vehicle
    /red") 1. ) ) ) ) ) )
(property (always (implies (or (include (prop "/vehicle/red") (prop "/object/bj") )  (overlap (
    prop "/vehicle/red") (prop "/object/bj") ) ) (not (equal (prop "/vehicle/red") (nextt (prop "
    /vehicle/red") ) ) ) ) ) ) )
```

Fig. 12 Example of three properties in stem's specification language

Each batch serves as an independent unit, containing all the necessary data for a single test deployment. The test bundle is a versioned repository that includes all the artifacts and code required to test and validate the ADS. This repository contains not only the code, but also the unit tests essential for the testing process. This approach guarantees that the testing and validation process remains repeatable and consistent, allowing for the early detection of potential faults in the development of ADS functionalities.

Additionally, there is a container dedicated to monitoring the behavior of vehicles. Its purpose is to assess whether the system being tested has controlled the ego vehicle following predefined rules, ground truth, and traffic laws. This assessment is performed online through the following steps:

1. The traffic rules are converted into properties and provided to our stem tool as symbolic expressions (see Fig. 12). For example, one expression specifies that neither the `red` nor `orange` vehicles should collide (rule 1). Another expression adds a safety margin of 1.0 unit to this constraint using the `expand` operator (rule 2). A third expression states that the ego vehicle (`red`) should not stop in the box junction (rule 3);
2. Once monitors are generated based on these expressions, they are packaged within the container and launched by the test deployment orchestration;
3. Finally, verdicts indicating compliance or non-compliance with rules are displayed on a dashboard, while logs are stored in a volume for potential future analysis if necessary.

7 Related work

Spatial logic has been studied quite extensively over the decades. Initially addressed from a theoretical point of view, questions like expressiveness, decidability, provability, and complexity, were among the main efforts of researchers. For instance, the work by Kurucz et al. [31] focused on answering those questions in the context of modal logic in metric spaces. We refer the reader to [1] for a comprehensive review of these topics.

More recently, spatial logic has been applied to matters of a more practical nature. Not only decision procedures have been devised for fragments of spatial languages [12], but also several application domains have enjoyed the benefits of being able to reason in space, and have also motivated the development of new domain-specific spatial languages [26].

One noteworthy example of a spatial language that has emerged from similar circumstances is the spatial logic for closure spaces (SLCS) proposed by Ciancia et al. [15]. This spatial language operates on closure spaces, specifically quasi-discrete closure spaces. With the availability of a decision procedure [17], it becomes possible to employ lightweight formal methods, such as model checking [16], in the field of medical imaging [13]. It establishes a connection between metric spaces and SLCS through an operator $\mathcal{D}^{\leq}$ whose semantics coincide with semantics of the "Extension" operator $\exists^{\leq}$ introduced in Sect. 2, if and only if a finite model is considered, meaning that more general space definitions, such as Euclidean spaces, are excluded.

An important SLCS spatial formula is the 'Surround', which is presented by the authors as a spatial version of the "Temporal Until". Adapted from [1, Chap. 5], it states that a spatial until returns an area whose points satisfy a certain property and the points belonging to its boundary closure must satisfy another given formula.

SLCS was expanded with two temporal operators of computer tree logic (CTL) to create the spatial-temporal logic for closure spaces (STLCS). This language allows for reasoning about the evolution of spatially distributed CPSs, such as smart cities [19, 53] or bike sharing systems [18]. However, STLCS and CTL do not have quantitative semantics defined, meaning that satisfaction or violation of a property can be determined, but not the degree of satisfaction or violation. It is important to note that there is a conceptual difference between the "Until" term in this paper and the "Surround" formula in STLCS, which both use the same "Surround" operator as in SLCS.

In the context of STLCS, the SLCS's "Surround" operator computes an area that satisfies certain conditions at a given time point. On the other hand, the "Spatial Until" introduced in Sect. 2 constructs a term by combining unions and intersections of future projections of the same term. This provides a direct way to relate the term at different time points and is extensively used in this work for formalizing traffic rules into LTL×MS formulas using the "Next" operator. Therefore, there is a fundamental difference in interpretation between these two "Spatial Until" operators.

The definition of SLCS's spatial until also served as inspiration to several spatial extensions of signal temporal logic. For instance, the spatial signal temporal logic (SSTL) presented by Nenzi et al. [42], extended signal temporal logic (STL) with the "Surrounded" and "Somewhere" spatial operators, the latter borrowed from the well-established multiprocess network logic [44]. SSTL was designed to reason about spatio-temporal models whose locations are static, this downside was surpassed by the introduction of spatio-temporal reach and escape logic (STREL) by Bartocci et al. [7] to deal with dynamical CPSs. The authors have extended STL with spatial operators "Reach" and "Escape", which in turn can be used to derive the SLCS's "Surround" operator. Unlike SLCS and STLCS, SSTL and STREL have quantitative semantics, enabling the use of techniques such as falsification or parameter synthesis.

Although the study of spatio-temporal languages is a relatively new field compared to temporal and spatial logics, several authors have explored it from both theoretical and practical points of view [1, Chap. 9]. For example, Bennett et al. [9] introduced propositional spatio-temporal logic (PSTL), which combines propositional temporal logic and modal logic $\mathcal{S}4_u$ to reason in multi-dimensional spaces and time. More recently, PSTL has been applied to formalize safety requirements in train control systems [51]. Similarly, Sun et al. [52] used Metric Temporal Logic to enhance expressiveness when reasoning about time in CPS safety requirements.

Also, Li et al. proposed a highly expressive spatio-temporal specification language (STSL) that utilizes signal temporal logic and $\mathcal{S}4_u$ to reason about time and space [34]. STSL can be divided into two categories: STSL *PC*, where spatial propositions change over time, and STSL *OC*, which represents how spatial terms change over time. The completeness and decidability of STSL are discussed in a subsequent work [35], based on the principles of *PC* and *OC* as introduced by Gabelaia et al. [24].

Recently, Li et al. [36] expanded the capabilities of STSL *PC* by introducing boolean and quantitative semantics. These new semantics were applied in two different scenarios: the falsification of an adaptive cruise control system to identify counter-examples that violate a safety property expressed in STSL *PC*, and the parameter synthesis of a path planning quadrotors algorithm to compute a trajectory that reliably satisfies an STSL *PC* formula.

In addition to networked systems and CPS, spatio-temporal languages find application in pattern recognition. The work by Haghighi et al. [27] introduced spatial-temporal logic (SpaTeL), while Bartocci et al. [8] combined the spatial language tree spatial superposition logic (TSSL) with STL for pattern recognition in reaction-diffusion systems governed by partial differential equations.

In this discussion on spatio-temporal logic, it becomes evident that there are various possible combinations of temporal and spatial logic. The choice of the logic itself and the way they are combined play a significant role [1, Chap. 9]. For example, STSL *OC* and LTL×MS both adhere to the principles of local object change (*LOC*) and asymptotic object change (*AOC*). These principles allow for the representation of motion in terms of space-time Cartesian product over fixed periods (*LOC*) or non-fixed periods (*AOC*) [1, p. 533]. Another example is STSL *PC*, which is less expressive than its *OC* counterpart, but still decidable according to [35]. The presence of the *PC* principle can also be observed in the previously mentioned STL spatial extensions. Indeed, this principle is mandatory for any formal language aiming to describe spatio-temporal behavior [1, p. 531].

The Cartesian product of temporal and spatial logics results in languages that are significantly distinct from the STL spatial extensions, SSTL [42] or STREL [7]. One notable difference is that in LTL×MS it is not possible to nest spatial and temporal formulas, whereas it is allowed in STLCS and the spatial extensions of STL. Additionally, the former type of spatio-temporal logics has separate syntax and semantics for terms and formulas. This separation allows for greater expressiveness when describing spatial entities, whether they are point-based or set-based terms.

The language LTL×MS is not limited to a specific application and can be used in various domains, which is advantageous. In contrast, SpaTeL, SSTL, or STREL were specifically designed for certain use cases like discrete reaction-diffusion patterns or spatially and networked distributed dynamical systems. These cases typically involve discrete spaces as the preferred choice for representing spatial structures to be analyzed. One example is the spatial aggregation signal temporal logic (SaSTL) [38]. Developed as an extension of STL, this framework focuses on aggregating and counting signals from various locations within a smart city. As a result, it has shown great potential in monitoring the safety requirements of smart cities in real-time. However, it should be noted that SaSTL is a highly specialized language with limited applicability beyond this specific domain. Another example that is more relevant to our use case is the Multi-Lane Spatial Logic (MLSL), a spatial language introduced by Hilscher et al. [28]. MLSL is a multi-dimensional spatial language that utilizes one dimension, a continuous space, to describe the position of a vehicle in a specific lane, and another dimension, a discrete space, to specify the lane number. We will explore this topic in more detail below.

Conversely, LTL×MS operates under the assumption of metric spaces, which can either be discrete or continuous. In this article, we specifically focus on continuous metric spaces and utilize snapshots of metric models $\mathfrak{M}$ that are obtained from the Cartesian product between LTL and a fragment of modal logic for metric spaces $MS^{\leq}$. We have chosen this combination of discrete time and continuous space as it aligns well with our use case of monitoring traffic rules.

Furthermore, LTL×MS not only allows for the inclusion of other spatial structures and representations, such as discrete spaces, as long as a metric is provided, but also provides flexibility in choosing frame time flows. Unlike LTL, which requires a strict relation order $<$ between time points to be satisfied without the need for defining a metric between them, LTL×MS allows for the choice of uneven spacing between time stamps or even no spacing at all. In this work, we build upon CARLA's simulator acquisition rate, which remains constant throughout the execution. However, if needed and described in Sect. 5.2, one can opt to drop some frames between events based on vehicle position and velocity to derive a more efficient monitoring algorithm.

To sum up, we would like to highlight three key features of LTL×MS that we found valuable in our use case, with regards to its relation with the various spatio-temporal languages discussed earlier:

1. Equipped with the principles of *LOC* and *AOC*, we can engage in reasoning over specific, finite time intervals as well as over the entire duration of time;
2. The until of the terms, despite not having a physical meaning, allows a direct and simple way to argue about the same spatial objects at different points in time;
3. The formal language can be tailored, by choosing appropriate metric spaces and time flows, to different needs. Whether this be the optimization of the monitor or the use of LTL×MS in different contexts, other than formalizing traffic rules.

7.1 Formal verification of AVs

Up to this point, the primary topic of discussion has been the distinctions between various spatio-temporal languages and their respective pros and cons. Now, our attention turns towards the state-of-the-art related to AVs. This field of study is abundant with works that employ formal verification, both offline and at runtime, contracts, and formalization of traffic rules using formal languages.

There are numerous applications where formal approaches can assist an AV in correctly accomplishing a specific task. Vasile et al. [55] focus on formalizing a minimum-violation plan that provides a means to balance the satisfaction of client demands, which must be met within specified deadlines, with the violation of specific traffic rules. To achieve this, they integrate a fragment of LTL into the graph algorithm RRT*. This work demonstrates that formal verification can go beyond traditional validation and verification methodologies in addressing traffic-related issues.

An AV system should ideally have the ability to self-check if its autonomous driving component adheres to traffic rules. Aréchiga [5] proposed a significant advancement in this area using STL. He introduced the concept of automatically synthesizing runtime monitors, similar to what we have presented in our work, but without considering spatial aspects as a primary concern. A key finding of this work was the establishment of a set of contracts that, when followed by all traffic participants, guarantees that the overall system will not experience collisions. Cardoso et al. [14] also suggest verification by contract as a useful tool to handle complex systems like AVs.

The process of formalizing traffic rules involves converting ambiguous natural language traffic regulations into precise logical formulas that can be understood and interpreted by computers. However, this task presents its challenges, which are discussed in Prakken's study on Dutch traffic law [43]. This work explores potential approaches for designing AVs that comply with traffic rules that often contain conflicts, exceptions, and common-sense knowledge. It offers a unique perspective compared to the typical CPS. From a practical standpoint, Bhuiyan et al. work [10] addresses exceptions and conflicts within the context of Australian traffic rules using defeasible deontic logic (DDL).

Rizaldi et al. [46] utilize the tool Isabelle to generate code and monitor the satisfaction of overtaking rules in LTL. Another study [39] explores similar rules, but in an interstate situation, formalizing the overtaking rules using MTL properties. Some works focus on formalizing rules in road junction scenarios, such as those discussed in [3, 4].

So far, we have only mentioned the formalization of traffic rules using temporal languages. However, there is also literature that addresses this problem from a spatial and spatio-temporal perspective.

One notable example is MLSL [28, 37], which was initially designed for scenarios like highways. Xu and Li [58] extended MLSL by introducing vertical lanes intersecting with horizontal lanes, adding a second dimension to account for road junctions and other geometries. Schwammberger proposed Urban Multi-Lane Spatial Logic (UMLSL) [49], which focuses on urban settings like roundabouts. Schwammberger also developed a temporal extension of UMLSL called MLSTL [50], which captures both spatial and temporal aspects required by UK road junction rules. He used LTL to reason about time.

To our understanding, there has been limited research in the formalization of traffic rules using spatio-temporal languages. This influenced our decision to employ LTL×MS. In terms of technical aspects, Sahin et al. work [47] closely relates to ours, as it converts STL into a mixed-integer programming solver, allowing for real-time monitoring of autonomous vehicle failures in an urban setting. In contrast, we convert LTL×MS expressions into $\mathrm{FOL}_{\mathbb{R}}$ and establish a monitoring procedure using Z3 [21].

8 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 for 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.

Currently, our monitoring procedure is capable of verifying the satisfaction of an LTL×MS formula. However, it cannot pinpoint which actor and which part of the scenario caused a violation of a specific formula. To address this issue, we propose utilizing Z3's minimum unsatisfiability core.

Additionally, we can enhance LTL×MS by providing it with robust semantics that go beyond simple Boolean values. This would enable us to determine not only whether a formula is satisfied or violated, but also quantify the degree of satisfaction or violation. With this enhanced semantics, we can explore techniques like falsification or parameter synthesis that rely on robust semantics for more accurate analysis and evaluation with real-world scenarios.

Appendix A: Conversion with the unroll method: monitoring model construction

Proof (Sketch – Lemma 1)
The proof sketch proceeds by structural induction over ϱ.

Base case $\varrho = p$ By definition, the valuation $\mathfrak{R}$ is a map associating each spatial variable p and time point t to a set $\mathfrak{R}(p,t) \subseteq \Delta$. Assume that the valuations in Δ are given by sets of inequalities at time point k with N-dimensions as the construction of $List(A)$ is given by the function obj that just constructs inequalities and maps them to spatial variables. Again, by definition $\mathbf{conv}_\varrho(p)$ is given by the function $eval$ that gets an expression in $\mathrm{FOL}_{\mathbb{R}}$ from sets of inequalities constructed by the obj function.

We know that there is an $\mathbf{x} \in \mathbb{R}^N$ such that $\mathfrak{R}(p,0) = \mathbf{x}$ *iff* $eval(p)$ holds. So there is an $\mathbf{x} \in \mathbb{R}^N$ such that $\mathfrak{R}(p,0) = \mathbf{x}$ *iff* $\mathbf{conv}_\varrho(p)$ also holds. Now, we have to show that

$$\text{for all } k \in [0,n], \text{there is an } \mathbf{x} \in \mathbb{R}^N \text{ such that}$$
$$next_\varrho(next_\varrho(\cdots next_\varrho(eval(p))\cdots)) \textit{ iff } \mathfrak{R}(p,k) = \mathbf{x}.$$

We skip this intermediate proof sketch that needs to proceed by induction on the size of the 'nesting next' over p. Therefore, for all $k \in [0,n]$, there is an $\mathbf{x} \in \mathbb{R}^N$ such that $\mathfrak{R}(p,k) = \mathbf{x}$ *iff* $\mathbf{conv}_\varrho(p)$ holds.

Inductive step $\varrho = \varrho_1 \mathfrak{U}_s \varrho_2$ By the semantics definition of ϱ, we have that

$$\bigcup_{m>n}\left(\mathfrak{R}(\varrho_2,m)\cap\bigcap_{k\in(n,m)}\mathfrak{R}(\varrho_1,k)\right)$$
$$=\bigcup_{j=n+1}^{m}\left[\mathfrak{R}(\varrho_2,j)\cap\bigcap_{k=n+1}^{j-1}\mathfrak{R}(\varrho_1,k)\right]$$
$$=\mathfrak{R}(\varrho_2,n+1)$$
$$\cup\left[\mathfrak{R}(\varrho_2,n+2)\cap\mathfrak{R}(\varrho_1,n+1)\right]$$
$$\cup\left[\mathfrak{R}(\varrho_2,n+3)\cap\mathfrak{R}(\varrho_1,n+1)\cap\mathfrak{R}(\varrho_1,n+2)\right]$$

$$\cup \dots \cup [\mathfrak{N}(\varrho_2,m) \cap \mathfrak{N}(\varrho_1,n+1) \cap \dots \cap \mathfrak{N}(\varrho_1,m-1)] .$$

Given that $\mathfrak{N}(\odot \varrho,n) = \mathfrak{N}(\varrho,n+1)$, we can write the latter expression as

$$\begin{aligned}
&\mathfrak{N}(\odot \varrho_2,n)\\
&\cup [\mathfrak{N}(\odot\odot \varrho_2,n) \cap \mathfrak{N}(\odot \varrho_1,n)] \cup \dots\\
&\cup \Big[\mathfrak{N}\big(\underbrace{\odot \dots \odot}_{(m-1)\times} \varrho_2,n\big) \cap \mathfrak{N}(\odot \varrho_1,n) \cap \cdots\\
&\cap \mathfrak{N}\big(\underbrace{\odot \dots \odot}_{(m-2)\times} \varrho_1,n\big)\Big]\\
&= \mathfrak{N}(\odot \varrho_2,n) \cup \bigcup_{j=2}^{m-1} \Big[\mathfrak{N}\big(\underbrace{\odot \dots \odot}_{j\times} \varrho_2,n\big)\\
&\cap \bigcap_{i=1}^{j-1} \mathfrak{N}\big(\underbrace{\odot \dots \odot}_{i\times} \varrho_1,n\big)\Big].
\end{aligned}$$

Let us now assume that ρ_1 and ρ_2 are any terms, and there exists an m such that $m > n$, where intuitively m is the time point when ρ_2 occurs. By applying the inductive hypothesis, we know that all the sets given by $\mathfrak{N}$ are given by sets of inequalities in $\text{FOL}_{\mathbb{R}}$. Therefore, for any term ρ_1, ρ_2 we have that

$$\begin{aligned}
&\mathbf{conv}_{\varrho}(\odot \rho_2)\\
&\vee \bigvee_{j=2}^{m-1} \Big[\mathbf{conv}_{\varrho}\big((\underbrace{\odot \dots \odot}_{j\times} \rho_2)\big) \wedge \bigwedge_{i=1}^{j-1} \mathbf{conv}_{\varrho}\big((\underbrace{\odot \dots \odot}_{i\times} \rho_1)\big)\Big],
\end{aligned}$$

where the resulting expression is a formula in $\text{FOL}_{\mathbb{R}}$. We will not provide the proof sketch for the remaining terms as they follow a similar pattern.

Proof (Sketch – Lemma 2)
The proof sketch proceeds by structural induction over φ.

Base case $\varphi = \varrho_1 \sqsubseteq \varrho_2$ By the semantics definition of φ, we have that $\mathfrak{N}(\varrho_1,n) \subseteq \mathfrak{N}(\varrho_2,n)$.

Let us now assume that ρ_1 and ρ_2 are any terms and there is a symbol l_k (the snapshot at a given time k). By definition, the valuations of ρ_1 and ρ_2 can be written as $\mathfrak{N}(\varrho_1,n) \subseteq \mathfrak{N}(\varrho_2,n) \subseteq l_k$. Let us look at ρ_1 first. By Lemma 1, we can write that for all $k \in [0,n]$, there is an $\mathbf{x} \in \mathbb{R}^N$, $\mathfrak{N}(\rho_1,k) = \mathbf{x}$ *iff* $\mathbf{conv}_{\varrho}(\rho_1)$. The same repeats for ρ_2.

Thus, for all $k \in [0,n]$, there is an $\mathbf{x} \in \mathbb{R}^N$ such that $\mathfrak{N}(\rho_1,k) \subseteq \mathfrak{N}(\rho_2,k) \subseteq l_k$ iff $\mathbf{conv}_{\varrho}(\rho_1) \to \mathbf{conv}_{\varrho}(\rho_2)$ holds.

Therefore, for all $k \in [0,n]$, there is an $\mathbf{x} \in \mathbb{R}^N$, $\mathfrak{M}(\rho_1 \sqsubseteq \rho_2,k)$ iff $\mathbf{conv}_{\varphi}(\varrho_1 \sqsubseteq \varrho_2)$ holds.

Inductive step $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$ An analogous proof sketch of the 'Until' term in Lemma 1 is obtained for the formulas, as follows:

$$\begin{aligned}
&\mathbf{conv}_{\varphi}(X\phi_2)\\
&\vee \bigvee_{j=2}^{m-1} \Big[\mathbf{conv}_{\varphi}(\underbrace{X \dots X}_{j\times} \phi_2) \wedge \bigwedge_{i=1}^{j-1} \mathbf{conv}_{\varphi}(\underbrace{X \dots X}_{i\times} \phi_1)\Big].
\end{aligned}$$

Again, we will not provide the proof sketch for the remaining terms as they follow a similar pattern.

A.1 Conversion examples

Example 1 (region contradiction — $\overline{a} \sqsubseteq a$)
The expression $\mathbf{conv}_{\phi}(\overline{a} \sqsubseteq a)$ is translated to the input language of Satisfiability Modulo Theories (SMT) solvers such as:

$$\begin{aligned}
&\forall a'.\ \big(\mathbf{conv}_{\varrho}(\overline{a}) \to \mathbf{conv}_{\varrho}(a)\big)\\
&\Leftrightarrow \forall a'.\ \big(\neg(\mathbf{conv}_{\varrho}(a)) \to eval(a)\big)\\
&\Leftrightarrow \forall a'.\ \big(\neg(eval(a)) \to eval(a)\big)\\
&\Leftrightarrow \forall a'.\ \big(eval(a) \vee eval(a)\big)\\
&\Leftrightarrow \forall a'.\ eval(a)
\end{aligned}$$

where a' means the free variables from which the 'a' spatial variable converts in $\text{FOL}_{\mathbb{R}}$. Note that a' refines to (x, y) when the spatial variable is defined in a 2D Euclidean space.

Example 2 (region expansion — $a \sqsubseteq \exists^{\leq 3} a$.)
The expression $\mathbf{conv}_{\phi}(a \sqsubseteq \exists^{\leq 3} a)$ is translated to the input language of SMT solvers such as:

$$\begin{aligned}
&\forall a'.\big(\mathbf{conv}_{\varrho}(a) \to \mathbf{conv}_{\varrho}(\exists^{\leq 3} a)\big)\\
&\Leftrightarrow \forall a'.\big(eval(a) \to dist(3, \mathbf{conv}_{\varrho}(a))\big)\\
&\Leftrightarrow \forall a'.\big(eval(a) \to dist(3, eval(a))\big)\\
&\Leftrightarrow \forall a'.\big(\neg(eval(a)) \vee dist(3, eval(a))\big)
\end{aligned}$$

where a' means the free variables from which the 'a' spatial variable converts in $\text{FOL}_{\mathbb{R}}$.

Example 3 (region with temporal behavior — $a \sqsubseteq (\perp \,\mathfrak{U}\, d) \wedge \neg(a \sqsubseteq d)$.)
The expression $\mathbf{conv}_{\phi}(a \sqsubseteq (\perp \,\mathfrak{U}\, d) \wedge \neg(a \sqsubseteq d))$ is translated

to the input language of SMT solvers such as:

$$
\begin{aligned}
&\mathbf{conv}_{\varphi}(a \sqsubseteq (\bot \ \mathfrak{U} \ d)) \wedge \mathbf{conv}_{\varphi}(\neg(a \sqsubseteq d)) \Leftrightarrow \\
&\forall a' \ d'.\big(\mathbf{conv}_{\varrho}(a) \rightarrow \mathbf{conv}_{\varrho}(\bot \, \mathfrak{U}, d)\big) \wedge \neg \mathbf{conv}_{\varphi}(a \sqsubseteq d) \Leftrightarrow \\
&\forall a' \ d'.\big(eval(a) \rightarrow next_{\varrho}(\mathbf{conv}_{\varrho}(d))\big) \\
&\quad \wedge \neg \forall a' \ d'.\big(\mathbf{conv}_{\varrho}(a) \rightarrow \mathbf{conv}_{\varrho}(d)\big) \Leftrightarrow \\
&\forall a' \ d'.\big(eval(a) \rightarrow next_{\varrho}(eval(d))\big) \\
&\quad \wedge \neg \forall a' \ d'.\big(eval(a) \rightarrow eval(d)\big) \Leftrightarrow \\
&\forall a' \ d'.\big(\neg(eval(a)) \vee next_{\varrho}(eval(d))\big) \\
&\quad \wedge \neg \forall a' \ d'.\big(\neg(eval(a)) \vee eval(d)\big)
\end{aligned}
$$

where a', d' are free variables from which 'a','d' spatial variable converts in $\mathrm{FOL}_{\mathbb{R}}$. Again, note that a' and d' refines to $(x1, y1)$ and $(x2, y2)$ when the spatial variables are defined in a 2D euclidean space.

Acknowledgements We would like to express our sincere gratitude to the anonymous reviewers for their invaluable feedback and constructive criticism of this article. Their insightful comments and recommendations have greatly improved the quality and clarity of this paper.

Funding Open access funding provided by FCT|FCCN (b-on). This work is supported by the EU/Next Generation EU, through Programa de Recuperação e Resiliência (PRR) [Project Route 25 No. C645463824-00000063]. This work was also partially supported by: i) the European Regional Development Fund (ERDF) through the Competitiveness and Internationalization Operational Program (COMPETE 2020) of Portugal 2020 [Project STEROID with No. 069989 (POCI-01-0247-FEDER-069989)]; ii) and FCT grant UIDB/04516/2020.

References

1. Aiello, M., Pratt-Hartmann, I., van Benthem, J.: Handbook of Spatial Logics. Springer, Berlin (2007)
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), 6 (2022)
3. Alves, G.V., Dennis, L.A., Fisher, M.: Formalisation and implementation of road junction rules on an autonomous vehicle modelled as an agent. In: FM Workshops 2019. Proceedings, Porto, Portugal, October, 2019. LNCS, vol. 12232, pp. 217–232. Springer, Berlin (2019)
4. 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)
5. Aréchiga, N.: Specifying safety of autonomous vehicles in signal temporal logic. In: 2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019, pp. 58–63. IEEE, Los Alamitos (2019)
6. Association for Standardisation of Automation and Measuring Systems. https://www.asam.net/standards/. Retrieved 2022-04-11
7. Bartocci, E., Bortolussi, L., Loreti, M., Nenzi, L.: Monitoring mobile and spatially distributed cyber-physical systems. In: Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE '17, pp. 146–155. Assoc. Comput. Mach., New York (2017)
8. Bartocci, E., Gol, E.A., Haghighi, I., Belta, C.: A formal methods approach to pattern recognition and synthesis in reaction diffusion networks. IEEE Trans. Control Netw. Syst. **5**(1), 308–320 (2018)
9. Bennett, B., Cohn, A.G., Wolter, F., Zakharyaschev, M.: Multi-dimensional modal logic as a framework for spatio-temporal reasoning. Appl. Intell. **17**(3), 239–251 (2002)
10. Bhuiyan, H., Governatori, G., Bond, A., Demmel, S., Islam, M.B., Rakotonirainy, A.: Traffic rules encoding using defeasible deontic logic. In: JURIX 2020, Brno, Czech Republic, December, 2020. Frontiers in Artificial Intellig and Applications, vol. 334, pp. 3–12. IOS Press, Amsterdam (2020)
11. Borg, M., Englund, C., Wnuk, K., Duran, B., Levandowski, C., Gao, S., Tan, Y., Kaijser, H., Lönn, H., Törnqvist, J.: 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)
12. Bresolin, D., Sala, P., Monica, D.D., Montanari, A., Sciavicco, G.: A decidable spatial generalization of metric interval temporal logic. In: Markey, N., Wijsen, J. (eds.) TIME 2010–17th International Symposium on Temporal Representation and Reasoning, Paris, France, 6–8 September 2010, pp. 95–102. IEEE Comput. Soc., Los Alamitos (2010)
13. Buonamici, F.B., Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Spatial logics and model checking for medical imaging. Int. J. Softw. Tools Technol. Transf. **22**(2), 195–217 (2020)
14. Cardoso, R., Kourtis, G., Dennis, L., Dixon, C., Farrell, M., Fisher, M., Webster, M.: A review of verification and validation for space autonomous systems. Curr. Robot. Rep. **2**, 09 (2021)
15. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space. In: Díaz, J., Lanese, I., Sangiorgi, D. (eds.) Proceedings, Theoretical Computer Science – 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Lecture Notes in Computer Science, vol. 8705, pp. 222–235. Springer, Berlin (2014)
16. Ciancia, V., Grilletti, G., Latella, D., Loreti, M., Massink, M.: An experimental spatio-temporal model checker. In: Bianculli, D., Calinescu, R., Rumpe, B. (eds.) Software Engineering and Formal Methods – SEFM 2015 Collocated Workshops: ATSE, HOFM, MoKMaSD, and VERY*SCART, York, UK, September 7-8, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9509, pp. 297–311. Springer, Berlin (2015)
17. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Spatial logic and spatial model checking for closure spaces. In: Bernardo, M., De Nicola, R., Hillston, J. (eds.) Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems – 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Advanced Lectures, Bertinoro, Italy, June 20-24, 2016. Lecture Notes in Computer Science, vol. 9700, pp. 156–201. Springer, Berlin (2016)
18. Ciancia, V., Latella, D., Massink, M., Paskauskas, R., Vandin, A.: A tool-chain for statistical spatio-temporal model checking of

bike sharing systems. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques – 7th International Symposium, ISoLA 2016, Imperial, Proceedings, Part I, Corfu, Greece, October 10-14, 2016. Lecture Notes in Computer Science, vol. 9952, pp. 657–673 (2016)

19. Ciancia, V., Gilmore, S., Grilletti, G., Latella, D., Loreti, M., Massink, M.: Spatio-temporal model checking of vehicular movement in public transport systems. Int. J. Softw. Tools Technol. Transf. **20**(3), 289–311 (2018)
20. de Matos Pedro, A., Silva, T., Sequeira, T.F., Lourenço, J., Seco, J.C., Ferreira, C.: Monitoring of spatio-temporal properties with nonlinear SAT solvers. In: Groote, J.F., Huisman, M. (eds.) Formal Methods for Industrial Critical Systems – 27th International Conference, FMICS 2022, Warsaw, Poland, September 14-15, 2022. Lecture Notes in Computer Science, vol. 13487, pp. 155–171. Springer, Berlin (2022)
21. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008, Proceedings, Budapest, Hungary, March, 2008. LNCS, vol. 4963, pp. 337–340. Springer, Berlin (2008)
22. Dosovitskiy, A., Ros, G., Codevilla, F., López, A.M., Koltun, V.: CARLA: an open urban driving simulator. In: CoRL 2017. Proceedings, Mountain View, California, USA, November, 2017. Machine Learning Research, vol. 78, pp. 1–16. PMLR (2017)
23. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, pp. 995–1072. MIT Press, Cambridge (1990)
24. Gabelaia, D., Kontchakov, R., Kurucz, Á., Wolter, F., Zakharyaschev, M.: Combining spatial and temporal logics: expressiveness vs. complexity. J. Artif. Intell. Res. **23**, 167–243 (2005)
25. 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, Amsterdam (2002)
26. Grosu, R., Smolka, S.A., Corradini, F., Wasilewska, A., Entcheva, E., Bartocci, E.: Learning and detecting emergent behavior in networks of cardiac myocytes. Commun. ACM **52**(3), 97–105 (2009)
27. Haghighi, I., Jones, A., Kong, Z., Bartocci, E., Grosu, R., Belta, C.: Spatel: a novel spatial-temporal logic and its applications to networked systems. In: HSCC'15, Seattle, WA, USA, April, 2015. Proceedings, pp. 189–198. ACM, New York (2015)
28. Hilscher, M., Linker, S., Olderog, E.-R., Ravn, A.P.: An abstract model for proving safety of multi-lane traffic manoeuvres. In: Qin, S., Qiu, Z. (eds.) Formal Methods and Software Engineering – 13th International Conference on Formal Engineering Methods, ICFEM 2011. Proceedings, Durham, UK, October 26-28, 2011. Lecture Notes in Computer Science, vol. 6991, pp. 404–419. Springer, Berlin (2011)
29. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev. **37**, 100270 (2020)
30. Kane, A.: Runtime Monitoring for Safety-Critical Embedded Systems. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA (2015)
31. Kurucz, A., Wolter, F., Zakharyaschev, M.: Modal logics for metric spaces: open problems. In: We Will Show Them! Essays in Honour of Dov Gabbay, Volume Two, pp. 193–108. College Pub., London (2005)
32. Kutz, O., Wolter, F., Sturm, H., Suzuki, N.-Y., Zakharyaschev, M.: Logics of metric spaces. ACM Trans. Comput. Log. **4**(2), 260–294 (2003)
33. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Program. **78**(5), 293–303 (2009)
34. Li, T., Liu, J., Kang, J., Yin, H., Yin, W., Chen, X., Stsl, H.W.: A novel spatio-temporal specification language for cyber-physical systems. In: QRS 2020, pp. 309–319. IEEE, Los Alamitos (2020)
35. Li, T., Liu, J., Sun, H., Chen, X., Zhang, L., Sun, J.: A spatio-temporal specification language and its completeness & decidability. J. Cloud Comput. **9**, 65 (2020)
36. Li, T., Liu, J., Sun, H., Chen, X., Yin, L., Mao, X., Sun, J.: Runtime verification of spatio-temporal specification language. Mob. Netw. Appl. **26**(6), 2392–2406 (2021)
37. Linker, S., Hilscher, M.: Proof theory of a multi-lane spatial logic. Log. Methods Comput. Sci. **11**(3) (2015)
38. Ma, M., Bartocci, E., Lifland, E., Stankovic, J.A., Sastl, L.F.: Spatial aggregation signal temporal logic for runtime monitoring in smart cities. In: 11th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2020, Sydney, Australia, April 21–25, 2020, pp. 51–62. IEEE, Los Alamitos (2020)
39. Maierhofer, S., Rettinger, A.-K., Mayer, E.C., Althoff, M.: Formalization of interstate traffic rules in temporal logic. In: IEEE Intelligent Vehicles Symposium, IV 2020, Las Vegas, NV, USA, October 19 – November 13, 2020, pp. 752–759. IEEE, Los Alamitos (2020)
40. Mehmed, A.: Runtime monitoring for safe automated driving systems. PhD thesis, Mälardalen University (2020)
41. Muller, P.: A qualitative theory of motion based on spatio-temporal primitives. In: KR'98, Trento, June, 1998, pp. 131–143. Kaufmann, Los Altos (1998)
42. Nenzi, L., Bortolussi, L., Ciancia, V., Loreti, M., Massink, M.: Qualitative and quantitative monitoring of spatio-temporal properties with SSTL. Log. Methods Comput. Sci. **14**(4) (2018)
43. Prakken, H.: On the problem of making autonomous vehicles conform to traffic law. Artif. Intell. Law **25**(3), 341–363 (2017)
44. Reif, J.H., Sistla, A.P.: A multiprocess network logic with temporal and spatial modalities. J. Comput. Syst. Sci. **30**(1), 41–53 (1985)
45. 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)
46. Rizaldi, A., Keinholz, J., Huber, M., Feldle, J., Immler, F., Althoff, M., Hilgendorf, E., Nipkow, T.: Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/hol. In: IFM 2017, Turin, Italy, September, 2017. LNCS, vol. 10510, pp. 50–66. Springer, Berlin (2017)
47. Sahin, Y.E., 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)
48. Sánchez, C., Schneider, G., Ahrendt, W., Bartocci, E., Bianculli, D., Colombo, C., Falcone, Y., Francalanza, A., Krstic, S., Lourenço, J.M., Nickovic, D., Pace, G.J., Rufino, J., Signoles, J., Traytel, D., Weiss, A.: A survey of challenges for runtime verification from advanced application domains (beyond software). Form. Methods Syst. Des. **54**(3), 279–335 (2019)
49. Schwammberger, M.: An abstract model for proving safety of autonomous urban traffic. Theor. Comput. Sci. **744**, 143–169 (2018)
50. Schwammberger, M., Vaz Alves, G.: Extending urban multi-lane spatial logic to formalise road junction rules. In: Farrell, M., Luckcuck, M. (eds.) Proceedings Third Workshop on Formal Methods for Autonomous Systems, FMAS 2021, Virtual, October 21-22, 2021. EPTCS, vol. 348, pp. 1–19 (2021)
51. Shao, Z., Liu, J., Ding, Z., Chen, M., Jiang, N.: Spatio-temporal properties analysis for cyber-physical systems. In: 2013 18th International Conference on Engineering of Complex Computer Systems, Singapore, July 17–19, 2013, pp. 101–110. IEEE Comput. Soc., Los Alamitos (2013)
52. Sun, H., Liu, J., Chen, X., Du, D.: Specifying cyber physical system safety properties with metric temporal spatial logic. In: Sun, J., Reddy, Y.R., Bahulkar, A., Pasala, A. (eds.) 2015 Asia-Pacific

Software Engineering Conference, APSEC 2015, New Delhi, India, December 1-4, 2015, pp. 254–260. IEEE Comput. Soc., Los Alamitos (2015)

53. Tsigkanos, C., Kehrer, T., Ghezzi, C.: Modeling and verification of evolving cyber-physical spaces. In: Tichy, M., Bodden, E., Kuhrmann, M., Wagner, S., Steghöfer, J.-P. (eds.) Software Engineering und Software Management 2018, Fachtagung des GI-Fachbereichs Softwaretechnik, SE 2018, 5.-9. März 2018, Ulm, Germany. LNI, vol. P–279, pp. 113–114. Gesellschaft für Informatik (2018)
54. United Nations: Vienna convention on road traffic (1968). https://unece.org/DAM/trans/conventn/Conv_road_traffic_eN.pdf. Retrieved 2022-04-11
55. Vasile, C.I., Tumova, J., Karaman, S., Belta, C., Rus, D.: Minimum-violation scltl motion planning for mobility-on-demand. In: 2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 – June 3, 2017, pp. 1481–1488. IEEE, Los Alamitos (2017)
56. Wolter, F., Zakharyaschev, M.: Spatial reasoning in rcc-8 with Boolean region terms. In: Proceedings of the 14th European Conference on Artificial Intelligence, ECAI'00, pp. 244–248, NLD. IOS Press, Amsterdam (2000)
57. Wolter, F., Zakharyaschev, M.: Reasoning about distances. In: Gottlob, G., Walsh, T. (eds.) IJCAI'03, Acapulco, Mexico, August 9-15, 2003. Proceedings, pp. 1275–1282. Kaufmann, San Mateo (2003)
58. 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 Comput. Soc., Los Alamitos (2016)