Progress-Aware Runtime Monitoring of Abstract Scenario Specifications

Nelson Brüchmann[1,*], Ishan Saxena[1] and Bernd Westphal[1]

[1]German Aerospace Center (DLR) e.V., Institute of Systems Engineering for Future Mobility, Oldenburg, Germany

Abstract

Runtime Monitoring (RM) enables the automated and objective evaluation of a system w.r.t its system specification. In the automotive domain, RM allows for the evaluation of whether a vehicle's actual behaviour corresponds to an abstract specification of its expected behaviour while interacting inside a traffic scenario. Such runtime monitors, in the automotive domain, must not only consider spatial aspects such as distances, positions, velocities, etc., but also temporal aspects such as the order in which particular traffic situations (or scenarios) occur. Existing runtime monitors provide verdicts on whether the actual vehicle behaviour corresponds to the abstract scenario specification, with possible outcomes being satisfied, violated, or inconclusive. Undesired behaviour is detected whenever there is a discrepancy between the monitor's verdict and the expected outcome defined by the scenario specification. Tracing the cause for the undesired behaviour requires knowledge of prior system behaviour. This information can be derived from exact knowledge of the intermediate verdicts of the runtime monitor, the so called monitor progression. This is not trivial to determine because the current satisfied constraints may simultaneously correspond to multiple (traffic) situations. Additionally, the satisfaction of temporal properties also depends on prior spatial and temporal properties.

To improve the analysis of unexpected system behaviours, we propose progress-aware RM of abstract scenario specifications. Specifically, we introduce hypercuts to describe all non-violating past and future system behaviours, simplifying the analysis of unexpected verdicts. Each monitor verdict is annotated with a hypercut, which allows the samples to be related back to the abstract scenario specification structure. In this work, we provide a well-defined hypercut using Duration Calculus and describe the construction of a progress-aware runtime monitor, based on Traffic Sequence Chart (TSC) monitoring. Finally, we demonstrate how hypercuts improve the analysis of unexpected verdicts through an application example and present a visualization method.

Keywords

Formal Specification, Runtime Monitoring, Automata Theory

CEUR-WS.org/Vol-4324/paper9.pdf

1. Introduction

Runtime monitoring (RM) enables the automatic and objective assessment of behaviours. In the automotive context, a behaviour may be the execution of a manoeuvre, such as passing an obstacle. These manoeuvres are differentiated by specific attributes defined in a specification. A manoeuvre is considered an instance of an abstract scenario specification if it satisfies both the spatial properties associated with traffic situations and the temporal properties associated with traffic scenarios. Runtime monitors perform this assessment and return a verdict indicating whether the specification is satisfied, violated, or currently inconclusive. While satisfied and violated verdicts are definitive (every continuation of the behaviour satisfies/violates the specification), the inconclusive verdict may change as the behaviour prefix continues. RM can be applied in a variety of contexts, since the specifications are not bound to only one specific application. For example, one can define a specification to validate autonomous driving functions that employ artificial intelligence. In this instance but also in general, the specification defines both the car's behaviour and the expected verdict. A monitor synthesized from this specification then evaluates whether the behaviour generated by artificial intelligence satisfies the requirements, violates them, or results in an inconclusive verdict. Whenever the monitor's verdict differs from the expected verdict, an undesired manoeuvre is detected. Identifying the specific violated properties

8th International Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis. OVERLAY 2026 @ FLoC, July 18–19, Lisbon, Portugal

*Corresponding author.

nelson.bruechmann@dlr.de (N. Brüchmann); ishan.saxena@dlr.de (I. Saxena); bernd.westphal@dlr.de (B. Westphal)

0000-0003-0575-4402 (I. Saxena); 0000-0002-6824-0567 (B. Westphal)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-09

is necessary for determining the changes required to achieve the expected verdict, as such details are not captured by the monitor's verdict. The verdicts of current runtime monitors (such as [1, 2]) only indicate whether the behaviour has satisfied or violated a specification, but they fail to specify which spatial or temporal properties were violated. This challenge can be addressed by analysing the spatial violations of the specific behaviour sample that triggered the unexpected verdict. However, this requires knowledge of the monitors progress of the specification, as it is unclear which temporal properties are satisfied at the moment of unexpected verdict occurrence. Therefore, it is necessary to analyse all previous monitor samples leading up to the unexpected verdict. This ensures that the temporal properties are fully understood before drawing a conclusion based on the spatial violations. Moreover, tracking progressions becomes significantly more complex whenever a single behaviour satisfies multiple temporal properties simultaneously.

In this work, we will describe the 'hypercut' property, which reflects the monitor's current progress with respect to a specification. A novel progress-aware monitor provides hypercuts alongside a monitor's verdict, to improve the analysis of traces with unexpected behaviour. We apply progress-aware monitoring to an automotive domain application example. For this, we use Traffic Sequence Charts (TSC) as a specification language and extend the monitoring concept with hypercuts. TSC have been demonstrated to be suitable for specifying and monitoring automotive traffic situations [3, 4, 1, 5] but also maritime [6] and railway [7, 8] domain situations.

A closely related work [9] employs LTL-based specifications for runtime monitoring. The authors provide an analysis of potential monitoring failures and offer guidance on how to identify and address them. While their approach focuses on providing general guidance for failure analysis, our approach utilizes hypercuts to directly represent the monitor's progress, thereby performing a portion of the analysis by helping to identify specific temporal property violations. Furthermore, our method provides the means to trace temporal property satisfactions and violations for every monitored behaviour.

2. Representing progress of monitors

This section formally introduces hypercuts as a representation of progress for Monitors. First, we provide some basics on TSC-Monitoring, with a particular focus on the temporal aspects of the Monitoring. For details on the underlying object model and the recognition of spatial views, see [1], [10] and [11]. Finally, we present hypercuts as a progress representation for runtime monitors.

Formally, a TSC is defined as a tuple $SV = (sv_1, \ldots, sv_n)$ of spatial views, where each sv_i defines the spatial properties of a specific traffic situation. Temporal components, such as sequence, concurrency, and choice operators, specify the expected ordering of these situations to construct an abstract traffic scenario specification. This work focuses on temporal sequences, which represent an ordered progression of traffic situations. In a TSC, such a sequence is denoted by arrows that connect consecutive spatial views, where sv_i refers to the i^{th} element of the TSC tuple. Informally, a concrete behaviour (or driving manoeuvre) satisfies a TSC specification if the final spatial view is reached, thereby satisfying all previous spatial views in sequence in succession. In the Spatial View Recognition (SVR) component, the TSC Monitor determines the satisfaction of all spatial views for the set of sampled input data. A spatial view is satisfied if all spatial properties hold for the given time point. In other words, the SVR output over time is an interpretation function $\mathcal{I} : \mathbb{R}_{\geq 0} \to \{\text{true}, \text{false}\}^n$, where $\mathcal{I}(t)(sv) \in \{\text{true}, \text{false}\}$ represents the truth value of variable sv at time t. The Temporal Evolution Recognition (TER), evaluates whether each spatial view has been satisfied in succession and generates the monitor's verdict. Specifically, for a time interval $[b, e]$ with $b, e \in \mathbb{R}_{\geq 0}$ and an interpretation $\mathcal{I}$ that assigns to every timepoint $t \in [b, e]$ which spatial views are satisfied. The TER determines at runtime whether an interpretation satisfies the TSC specification on the interval $(\mathcal{I}, [b, e])$ and returns a verdict.

To describe sequences of spatial view satisfactions over time intervals, we use Duration Calculus (DC) [12]. A phase $\mathcal{P}_i$ is a time period of an interpretation during which a specific spatial view is satisfied without interruption. This is expressed in Duration Calculus as $\mathcal{P}_i = \lceil sv_i \rceil$. A phase sequence of length n describes an uninterrupted succession of phases, where spatial views are satisfied one after

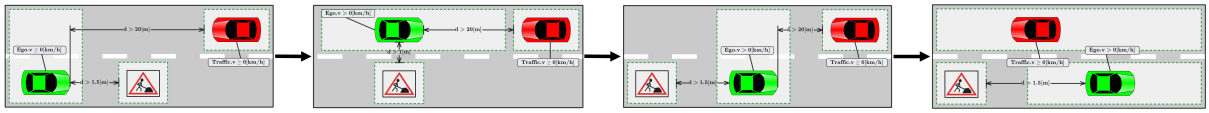


Figure 1: A TSC formalising the manoeuvre: passing by a construction site with oncoming traffic while maintaining distance from both the construction site and the oncoming vehicle.

another until the n^{th} element is satisfied. This is expressed by the DC formula $\mathcal{F}_i = \lceil sv_1 \rceil; \ldots; \lceil sv_i \rceil$.

The ';' chop operator is used to separate two neighbouring phases. It is satisfied if the time interval is split such that the left operand is satisfied on the first sub-interval, followed by the right operand on the second. Specifically, the satisfaction of these Duration Calculus primitives is defined as $(\mathcal{I}, [b, e]) \models \lceil sv_i \rceil$ iff for any $t \in [b, e]$ and $b < e$, $\mathcal{I}(t) = \mathit{true}$, and $(\mathcal{I}, [b, e]) \models sv_i; sv_j$ if there is an $m \in [b, e]$ s.t. $(\mathcal{I}, [b, m]) \models sv_i$ and $(\mathcal{I}, [m, e]) \models sv_j$.

Intuitively, a *hypercut* encodes the current progress of the monitor. For a TSC with n spatial views, $\mathcal{F}_n$ must be satisfied by the behaviour to satisfy the full chart. Hypercuts represent every partial phase sequence $\mathcal{F}_m$ (where $m \leq n$) to track progress through the satisfied substructures of the TSC. Specifically, these phase sequences indicate which previous spatial views in the TSC sequence have been satisfied and identify the monitor's current spatial view phase. A hypercut thus distinguishes past and current satisfied spatial views from those that have not yet been satisfied. For example, if a behaviour satisfies $\mathcal{F}_2$, then $\mathcal{P}_1$ must be satisfied before $\mathcal{P}_2$. Furthermore, hypercuts account for behaviours that satisfy multiple phase sequences simultaneously, as spatial view phases may overlap. This case is demonstrated in the provided example in section 3. Multiple overlapping spatial views require a hypercut to have non-contiguous phase sequences. For instance, if three spatial view phases overlap and the second view ceases to be satisfied while the others remain active, the monitor must track multiple distinct progressions for the same behaviour. None of the phase sequences can be omitted. If one phase sequence cannot remain or progress to the next phase, another may still provide a future phase sequence that satisfies the TSC. Formally, a hypercut is a Duration Calculus formula consisting of a concatenation of phase sequences. The function $f : \mathcal{I} \times Intv \to \mathcal{DC}$ generates the hypercut as Duration Calculus formula from an interpretation and interval:

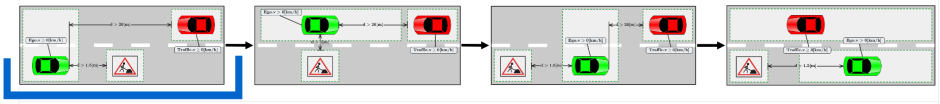
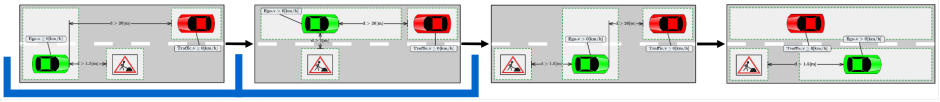
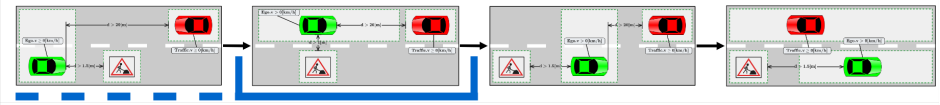
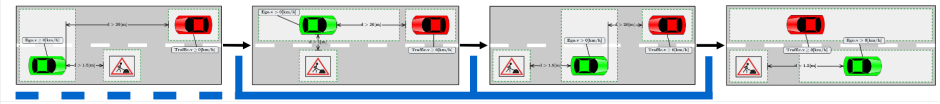
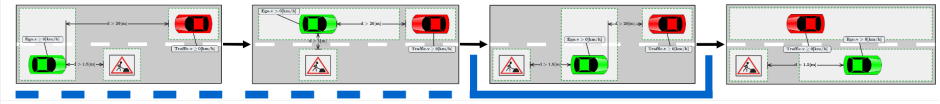
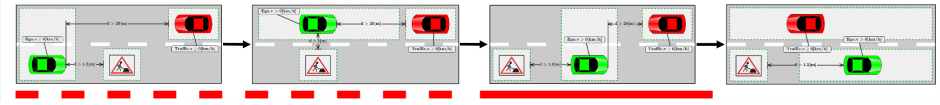
$$f(\mathcal{I}, [b, e]) := \bigwedge \{\mathcal{F}_i \mid \mathcal{I}, [b, e] \models \mathcal{F}_i, \text{ for } 1 \leq i \leq n\} \tag{1}$$

A hypercut is a conjunction of the satisfied phase sequences of an interval and interpretation. Such a hypercut can also be represented by a power set in a bit pattern, where each bit represents a phase sequence and the i^{th} position of the bit corresponds to the $\mathcal{F}_i$ phase sequence and the conjunction of the bits set to one represents the hypercut. For example, the pattern 101 corresponds to $\mathcal{F}_1 \wedge \mathcal{F}_3$. With these bit patterns we can extend the TER automata construction described in [1]. In that work, a TER is generated using a powerset construction, where each element of the set represents a location. To extend the automaton's verdict with a hypercut annotation, we include the current powerset element in the automaton's location during construction. The automaton then outputs this annotation alongside the verdict. This results in a bit pattern that represents a correct hypercut.

3. Example application

For the example application we use a traffic scenario, where a car has to safely and conveniently pass by a construction site with oncoming traffic. Figure 1 shows a possible formalisation of this traffic scenario. The ego vehicle (green) approaches a construction site, changes lane, passes the construction site, and then changes lane again. Throughout the entire manoeuvre it must maintain a safe distance from both the construction site and the oncoming vehicle. Note that the car's behaviour for a given time point may satisfy two spatial views for a period of time. The green boxes in the spatial views express that the car has to be somewhere within the box. When the car changes its lane, the behaviour satisfies two spatial views because for a time period, because one spatial view allows the car to be in both lanes,

Table 1
Visualisation of hypercuts for a car's concrete manoeuvre with a row for each change of the phase sequences.

Hypercut	Hypercut Visualization
$\lceil sv_1 \rceil$	
$\lceil sv_1 \rceil \wedge \lceil sv_1 \rceil ; \lceil sv_2 \rceil$	
$\lceil sv_1 \rceil ; \lceil sv_2 \rceil$	
$\lceil sv_1 \rceil ; \lceil sv_2 \rceil \wedge$ $\lceil sv_1 \rceil ; \lceil sv_2 \rceil ; \lceil sv_3 \rceil$	
$\lceil sv_1 \rceil ; \lceil sv_2 \rceil ; \lceil sv_3 \rceil$	
$\{\}$	

while another spatial view requires the car to be in a specific lane. This way of formalising lane changes ensures that the following spatial view is reachable from the previous spatial view. Table 1 visualises the hypercuts of a concrete manoeuvre. For conciseness, only the first hypercut after a change of hypercut is shown in a row. The TSC acts as reference anchor for the lines below. A solid horizontal line under a spatial view represents that the manoeuvre is in the phase of the spatial view above. The solid line can be under multiple spatial views, as a behaviour can satisfy multiple phase sequences at once. A dashed line indicates passed phases. The behaviour has been in the corresponding phase, but is currently not in the given phase. Finally, vertical lines on the left side of a spatial view indicate possible continuations for the next behaviour sample. The set of possible next phases is determined by the union of the current active phases and the possible next phases of active phases. Once the chart becomes satisfied, the lines turn green and if the chart is violated, the lines become red. In this example behaviour the line becomes red in the last row, indicating a violation and in our case an unexpected behaviour. Because the previous hypercut contained only $\lceil sv_3 \rceil$ before the violation, we can conclude that the car violated the spatial constraints of the third and fourth spatial views, indicating that the behaviour can neither remain in the current phase nor progress to the next one. The violated properties of the third and fourth spatial view now indicate the reason for the violation. In this example the distance to the red car is not large enough, indicating an earlier lane change would enable the car to drive a satisfying manoeuvre.

4. Conclusion and future work

In this work, we introduced hypercuts as a novel representation of progress for runtime monitors. Specifically, a hypercut encodes the temporal substructures of a specification that have been satisfied at each monitoring step. This representation enhances the analysis of unexpected behaviours by improving the tracing of violations without the need to analyse the entire execution trace. We presented how hypercuts can be applied to monitoring Traffic Sequence Charts, and demonstrated the improved analysis procedure for automotive traffic situations.
While the current hypercut focuses on progress tracking, future work will explore the integration of hypercuts with other logical constructs, such as disjunction and concurrency.

Declaration on Generative AI

During the preparation of this work, the author(s) used DeepL and LanguageTool in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] R. Stemmer, I. Saxena, L. Panneke, D. Grundt, A. Austel, E. Möhlmann, B. Westphal, Runtime monitoring of complex scenario-based requirements for autonomous driving functions, Sci. of Comp. Prog. 244 (2025) 103301. URL: https://elib.dlr.de/214014/. doi:10.1016/j.scico.2025.103301.

[2] M. Leucker, C. Schallhart, A brief account of runtime verification, The Journal of Logic and Algebraic Programming 78 (2009) 293–303. doi:10.1016/j.jlap.2008.08.004, the 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07).

[3] D. Grundt, A. Köhne, I. Saxena, R. Stemmer, B. Westphal, E. Möhlmann, Towards runtime monitoring of complex system requirements for autonomous driving functions, in: M. Luckcuck, M. Farrell (Eds.), Proceedings Fourth International Workshop on Formal Methods for Autonomous Systems (FMAS) and Fourth International Workshop on Automated and verifiable Software sYstem DEvelopment (ASYDE), Berlin, Germany, volume 371 of *Electronic Proceedings in Theoretical Computer Science*, Open Publishing Association, 2022, pp. 53–61. doi:10.4204/EPTCS.371.4.

[4] J. S. Becker, Safe linear encoding of vehicle dynamics for the instantiation of abstract scenarios, in: A. E. Haxthausen, W. Serwe (Eds.), 29th International Conference on Formal Methods for Industrial Critical Systems, FMICS 2024, volume 14952 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 3–20. URL: https://elib.dlr.de/205077/. doi:10.1007/978-3-031-68150-9_1.

[5] D. Grundt, A. Rakow, P. Borchers, E. Möhlmann, What does AI need to know to drive: Testing relevance of knowledge, Sc. of Comp. Prog. 244 (2025) 103297. URL: https://elib.dlr.de/213085/. doi:10.1016/j.scico.2025.103297.

[6] A. Austel, M. Steidel, B. Westphal, Formal specification of situations in scenario-based testing of maritime assistance systems, in: European Workshop on Maritime Systems Resilience and Security (MARESEC 2024), 2024. URL: https://elib.dlr.de/204947/.

[7] M. Wild, J. S. Becker, C. Schneiders, E. Möhlmann, A scenario-based simulation framework for testing of highly automated railway systems, in: VEHITS 2025, Proceedings of the 11th Internationa Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS, SciTePress, 2025, pp. 88–99. URL: https://elib.dlr.de/212746/. doi:10.5220/0013286600003941.

[8] J. S. Becker, M. Wild, A. Rakow, E. Möhlmann, How far can we go? Capturing traffic rules via Traffic Sequence Charts, in: IEEE International Automated Vehicle Validation Conference (IAVVC), 2025. URL: https://elib.dlr.de/221593/. doi:10.1109/IAVVC61942.2025.11219537.

[9] S. Goyal, A. Griggio, S. Tonetta, Leveraging contracts for failure monitoring and identification in automated driving systems, in: A. Madeira, A. Knapp (Eds.), Software Engineering and Formal Methods, Springer Nature Switzerland, Cham, 2025, pp. 441–460.

[10] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, A. Rakow, Traffic Sequence Charts - From Visualization to Semantics, Reports of SFB/TR 14 AVACS 117, SFB/TR 14 AVACS, 2017.

[11] I. Saxena, D. Grundt, E. Möhlmann, B. Westphal, Towards runtime detection of novel traffic situations, in: OVERLAY'25, CEUR Workshop Proceedings, 2025. URL: https://zenodo.org/doi/10.5281/zenodo.18298437. doi:10.5281/ZENODO.18298437.

[12] E.-R. Olderog, H. Dierks, Real-Time Systems: Formal Specification and Automatic Verification, Cambridge University Press, 2008.