A Preliminary Approach to Verify Platoon Behaviour Using Execution Traces and Model Checking

Simona Correra(✉), Valeria Sorgente, Giulia Varriano, Vittoria Nardone, Francesco Mercaldo, and Antonella Santone

Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise, Campobasso, Italy
s.correra@studenti.unimol.it,
{valeria.sorgente,giulia.varriano,vittoria.nardone, francesco.mercaldo,antonella.santone}@unimol.it

Abstract. The rapid advancement of autonomous vehicle technology has significantly changed Intelligent Transportation Systems. Cooperative Cruise Control increases the efficiency of connected and autonomous vehicles by improving road safety, reducing fuel emissions and increasing traffic flow. By allowing vehicles to work together, Cooperative Cruise Control addresses key problems that cause road accidents, such as inadvertent braking, rear-end collisions, and driver error. This technology will enable vehicles to respond almost instantly to changes in speed and distance. Improve communication and coordination between vehicles and promote safe human distancing - reducing the chance of errors - new verification methods are needed to ensure the reliable performance of these platoon systems. This paper presents a preliminary approach to analyzing platoon system behaviors through execution traces. The aim is to develop an abstract model that reflects operations performed by platoon systems. Then, verifying the model using rigorous mathematical verification techniques, *i.e.,* model checking. The study ensures that the system consistently exhibits the desired behavior.

Keywords: Platoon · Formal Methods · Autonomus Veihcle

1 Introduction

With the rapid growth of driverless vehicle technology in recent years, Intelligent transportation systems (ITS) face significant changes. The Cooperative Adaptive Cruise Control (CACC) [1] is an innovative example of connected and autonomous vehicle technology. It aims to improve road safety, reduce fuel emissions and increase traffic flow. The CACC-supported vehicle category presents a important option to address the high rate of road accidents worldwide. By allowing connected vehicles to cross in a coordinated manner, CACC solutions can reduce some of the most common causes of accidents, such as inadvertent

A. Kocian et al. (Eds.): INTSYS 2024, LNICST 608, pp. 411–423, 2025.
https://doi.org/10.1007/978-3-031-86370-7_25

braking, rear-end collisions and driver error. A platoon is a group of vehicles that travel together, coordinating their speed and maintaining a safe distance from one another. CACC-based platoons advance vehicle coordination and communication by allowing vehicles to respond almost instantly to changes in speed and distance. Furthermore, CACC supports a safer space between cars and reduce the chance of human error, which often leads to accidents. The use of such advanced automation can reduce accidents on the road network. As a result, CACC could define an essential step in creating safer and more efficient roads around the world.

Several works have been proposed in the literature to develop and verify various aspects of platoon systems. Some considerable examples include research on platooning protocols [13,18], which focuses on the rules and procedures governing vehicle coordination; existing testbeds that evaluate platooning protocols and messages [4,5,22], providing practical insights into real-world applications; and studies on cybersecurity in platooning [11,12,23], addressing the potential vulnerabilities and threats that these systems may face. These studies provide valuable insights into the comprehensive understanding of platoon systems and their operational dynamics.

Considering the critical role these emerging technologies play in increasing road safety, it is necessary to develop new methods to verify the reliable performance of the platoon system [2]. Rigorous verification processes are required to ensure these systems perform reliably and as intended in various real-world conditions [6]. Thus, to ensure better safety for all road users using comprehensive testing and inspection methods is needed. A comprehensive assessment confirms that CACC-based class control meets accuracy standards, reliability, and safety standards at a high level.

For the above reasons, we propose a preliminary approach to verify platoon systems' behaviour. Specifically, we aim to analyze the execution traces of a platoon system to develop an abstract model depicting its behaviour. By applying model checking– a rigorous mathematical verification technique–, we want to assess the system's behaviour to ensure that it consistently exhibits the desired behaviours while avoiding any unwanted ones.

The remainder of this paper is organized as follows: Sect. 2 briefly describes the model checking technique and outlines CACC and platooning, Sect. 3 overviews the current literature and contextualizes our work in the existing literature. Section 4 presents the workflow of the proposed approach, and Sect. 5 shows preliminary results. Section 6 discusses challenges and limitation of the proposed approach. Finally, Sect. 7 concludes the work.

2 Background

This section provides an overview of Cooperative Adaptive Cruise Control (CACC) and Platooning to contextualize the context of this work better. Moreover, in the following, we briefly describe the Model Checking technique workflow to provide the reader with preliminary notions about the technique used in this work.

2.1 Platoon and CACC

Cooperative Adaptive Cruise Control (CACC) [1] is an extension of Adaptive cruise control (ACC) [24]. CACC is an on-board system designed to maintain a minimum safe distance from vehicles directly ahead in the same lane. This distance can be either fixed or dynamic, depending on the vehicles speeds, and is typically aligned with a specified time interval that reflects a minimum reaction time. To guarantee a safe and efficient drive, CACC incorporates several components. It collects real-time data regarding the environment using various onboard sensors, such as radar, LIDAR, and cameras. Communication V2X, where X can be either another vehicle (V) or local roadside units (I), allows vehicles to share their data with others. Through V2X communication, every vehicle will receive and send information concerning traffic conditions and potential risks. The onboard unit processes sensor data and V2X communications and computes safe inter-vehicle distance. This analysis will allow the vehicle to adapt its driving strategy appropriately using acceleration and brake commands.

A group of vehicles that move together by synchronizing their speed and maintaining a safe distance between each other is called a platoon. The communication within the vehicle improves their cooperation toward any agreements over the manoeuvres and driving speeds, improving traffic safety through the prevention of collisions. Furthermore, the reduced gap between the vehicles shortens the body-dynamic drag against the following cars, lowering their energy consumption and, therefore, their CO2 emissions. Platooning indicates the coordinated driving of connected vehicles in a line, based on wireless communication, to manage functions such as platoon creation, joining, and leaving. Platooning holds two primary architectures: centralized, wherein a leader or a server makes operational decisions, and distributed, with each vehicle autonomously deciding on its position by using V2X and sensor data for maintaining safety distances and performing manoeuvres collaboratively.

2.2 Model Checking

Model Checking [7] is a technique within Formal Methods, which are rigorous mathematical approaches commonly employed to specify and verify complex systems. Owing to their precision and reliability, such techniques are particularly well-suited for the development and verification of safety-critical systems. Model Checking involves exhaustively exploring all possible states of a system and verifying whether specific properties hold in each state using a Model Checker tool. This tool requires two inputs: a formal model of the system and temporal logic formulas that specify the desired behaviors. The Model Checker evaluates these properties against the system model and provides a binary outcome: it returns true if the system satisfies the property and false otherwise. If the system fails to satisfy a property, the Model Checker can generate a counterexample—a sequence of events demonstrating how the property is violated. This feature is especially valuable for diagnosing and addressing the root cause of the violation.

The Formal Model of the System. The behavior of the system, as defined during the specification process, can be described using a formal model. This behavior is typically represented as a *Labeled Transition System (LTS)*. An LTS is composed of a set of states, transitions between these states, and labels that annotate the transitions. Additionally, one state is designated as the initial state. Formally, an LTS is defined as a quadruple $T = (S, \mathcal{A}, \longrightarrow, s)$, where S represents the set of states, $\mathcal{A}$ denotes the set of transition labels (or actions), $s \in S$ is the initial state, and $\longrightarrow \subseteq S \times \mathcal{A} \times S$ specifies the transition relation.

Processes are used to algebraically represent the elements of an LTS. One of the most notable process algebras employed for modeling complex systems is Milner's Calculus of Communicating Systems (CCS) [14]. This formalism provides a set of fundamental operators for defining finite processes, operators for communication and concurrency, and a mechanism for recursion to represent infinite behaviors. The syntax of *processes* behaves as follows:

$$p ::= \text{nil} \mid \alpha.p \mid p + p \mid \ p|p \ \mid p\backslash L \mid p[f] \mid x$$

where α ranges over a finite set of actions $\mathcal{A} = \{\tau, a, \bar{a}, b, \bar{b}, ...\}$. Processes can perform input actions labeled as "unblocked", denoted by a, while output actions are considered "blocked", represented by $\bar{a}$. When a process is ready to execute an action a, it can synchronize with another process performing the corresponding blocked action $\bar{a}$. This is why these are called *complementary actions*.

An action $\tau \in \mathcal{A}$ is a generic *internal action*. This allows processes to be described abstractly, hiding complex sequences of operations, and keeping details private. The sets of *visible actions* $\mathcal{A} - \{\tau\}$ are included in the set L, which can be used to create communication between processes through internal actions. Visible actions are those used to define the system properties.

The relabeling function f, in processes of the form $p[f]$, is a function such that $f : \mathcal{A} \to \mathcal{A}$, such that it satisfies the constraint $f(\tau) = \tau$. In addition, each constant x is defined by a definition of constant $x \stackrel{\text{def}}{=} p$.

The semantics of CCS by induction on the structure of processes is very rich:

- The nil process allows no action to be performed;
- The process $\alpha.p$ can execute α and later become the process p;
- The $p + q$ process can act as either process (p or q);
- The $|$ operator expresses parallel composition, while the $\backslash$ operator expresses action restriction. If visible actions are in the set L, then $p\backslash L$ is a process that acts like p but cannot execute any of the actions (even blocked actions) found in L. This is true even though any pair of complementary actions can be executed for communication;
- The $[f]$ operator guarantees relabeling of actions: if p executes α and becomes the new process p', then $p[f]$ can execute $f(\alpha)$ and become $p'[f]$.

The behavior of the process x where $(x \stackrel{\text{def}}{=} p)$ is that of its definition p.

Temporal Logic Formulas for Properties. Temporal logic formulas provide a formal way to specify that certain properties will hold in all states at every step, or that a specific event will happen at some point in the future. Two simple instances are the *safety properties* and the *liveness properties*. The first one expresses that an undesirable situation will never happen, while the second one declares that some reactions will always follow some actions.

In our study, we adopt *mu-calculus* logic [20] as the temporal logic framework. The syntax of mu-calculus is presented below, where Z represents variables, and K and R denote subsets of actions from $\mathcal{A}$. $\phi ::=$

$$\texttt{tt} \mid \texttt{ff} \mid Z \mid \phi \vee \phi \mid \phi \wedge \phi \mid [K]\,\phi \mid \langle K \rangle\,\phi \mid \nu Z.\phi \mid \mu Z.\phi$$

The satisfaction by a state s in a transition system of the formula ϕ, written as $s \models \phi$, is outlined as follows:

- **Basic cases:** Every state satisfies `tt`, while no state satisfies `ff`;
- **Logical Connectives:** A state satisfies $\phi_1 \vee \phi_2$ ($\phi_1 \wedge \phi_2$) if it satisfies ϕ_1 or (and) ϕ_2.
- **Modal operators $[K]\,\phi$ and $\langle K \rangle\,\phi$:**
  i. A state satisfies $[K]\,\phi$ if, for every action in K, it transitions to a state that satisfies ϕ.
  ii. A state satisfies $\langle K \rangle\,\phi$ if it can transition to a state that satisfies ϕ by performing an action in K.

Formal Verification Environment. Model Checking techniques [7] require a formal verification environment to validate the properties defined on the system model. Concurrency Workbench of New Century (CWB-NC) [8] is one of the most widely used environments for verifying software systems, supporting various specification languages such as CCS. In this study, we use CWB-NC as the formal verification environment.

3 Related Work

This section reports on the current literature on verifying platoon behaviour. In [21], it has been proposed to control vehicle platoon with collision-safety and efficiency measures based on the platoon behaviour. It uses a local model predictive control (MPC) formulation to separate safety from tracking control. Furthermore, a braking hold-back strategy has been defined to improve the efficiency of the platoon with smaller inter-vehicle distances. A corridor-based reference shaping and V2V communication strategy have been introduced to have a behaviour based on the situation, providing excellent performance with traffic disturbances and low communication effort.

Various projects have been designed and implemented in the past years to improve safety, emissions reductions, and driving comfort. For example, COMPANION (Cooperative Dynamic Formation of Platoons for Safe and Energy-optimized Goods Transportation) was developed to define a cooperative system for truck platooning management to reduce fuel consumption and improve

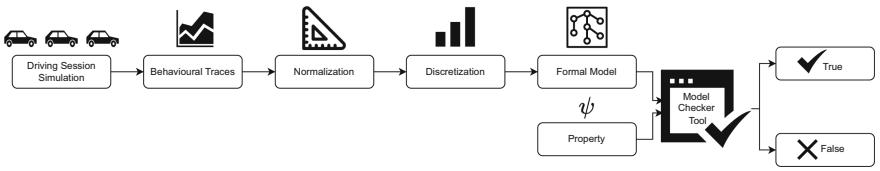

Fig. 1. Our preliminary approach workflow.

safety. Communication among vehicles is performed using V2X communication and ACC. Due to the different parties involved in the platoon (for example, sensors and inter-vehicle communication), it can be vulnerable to attacks. In [17], a strategy is proposed to improve the protection of the platoon. The authors introduce a protocol validated through theoretical analysis and experimental simulation using the PLEXE simulator. This allows the enhancement of platooning resilience by incorporating mechanisms such as real-time voting, which enables vehicles to detect and mitigate malicious attacks.

4 Preliminary Approach

The Fig. 1 depicts the workflow of our preliminary approach to verify a platoon system. The process begins with collecting traces from a simulated platooning driving session. These traces capture time-series data on vehicle parameters such as velocity and acceleration.

Initially, the collected data are subject to normalization using the min-max normalization technique [3]. Once normalized, the data is then discretized through two different methods:

- The first method identifies maximum and minimum values and discretizes the data into three intervals.
- The second method applies a sliding window to the samples, assessing their stationarity within the interval using the Augmented Dickey-Fuller (ADF) test [9]. For non-stationary samples, the slope of the interpolated data curve is computed to determine whether the trend is increasing or decreasing.

The discretized values are then used to construct formal models of the system. More in detail, we define two transformation functions, *i.e.,* f_1 and f_2, used on data produced by both discretization methods. Table 1 shows a simple example of execution traces for three vehicles, simplified to include only three moments in time. Note that for both Acceleration and Velocity, we use the following notation: $FeatureName_{XY}$, where $FeatureName$ corresponds to a string indicating a discretized value for the feature, X indicates the vehicle ID and Y is the instant of time when the discretized value occurs.

As stated in Sect. 2.2, to build the formal model, it is necessary to transform traces into an LTS using processes algebraically. To do so, we define two transformation functions producing two distinct formal models. The first transformation

Table 1. Example of Discretized Traces.

Vehicle ID	Time	Acceleration	Velocity
V_1	1	Acc_{11}	Vel_{11}
V_1	2	Acc_{12}	Vel_{12}
V_1	3	Acc_{13}	Vel_{13}
V_2	1	Acc_{21}	Vel_{21}
V_2	2	Acc_{22}	Vel_{22}
V_2	3	Acc_{23}	Vel_{23}
V_3	1	Acc_{31}	Vel_{31}
V_3	2	Acc_{32}	Vel_{32}
V_3	3	Acc_{33}	Vel_{33}

function defined is f_1. It takes as input all discretized traces of one vehicle and produces as output sequential processes able to mimic in each state the parallel composition of features. For this objective, we need the following definitions (as shown in Milner [14]).

Definition 1 (Well-terminating process). A CCS process is considered well-terminating if it performs the action $\overline{\delta}$ exclusively when it terminates immediately afterward.

The δ can be found in the definition of the operators $\|$ and ";" (see Table 2), and it is introduced to synchronise processes on termination. The two operators represent the sequentialization and parallel execution of two well-terminating processes, and the resulting processes from their application remain well-terminating.

Table 2. Operators for well-terminating processes.

$p\|q = (p[\delta_1/\delta] \mid q[\delta_2/\delta] \mid (\delta_1.\delta_2.DONE + \delta_2.\delta_1.DONE))\backslash\{\delta_1, \delta_2\}$
$p; q = (p[\delta_3/\delta] \mid \delta_3.q)\backslash\{\delta_3\}$
$DONE \stackrel{\text{def}}{=} \overline{\delta}.nil$

From the semantic CCS algebra:

- The constant $DONE$ represents a process that terminates immediately without performing any further actions;
- The ";" operator is used to enforce the sequential execution of two processes, p and q. In this way, p must terminate before q can begin its execution;
- The process "$p\|q$" models the parallel execution of p and q, terminating only when both processes have completed.

From this point forward, only CCS programs in which the operator | has been consistently replaced by | are considered.

More precisely, p_{ij} process has the following definition:

$$p_{ij} \stackrel{\text{def}}{=} (Acceleration \parallel Velocity)\ ; p_{ij+1}$$

where p_{ij} process represents the parallel composition of the vehicle i features at each time instant j, spanning from 1 to n. Therefore, for vehicle i, there will be a series of processes at each time instant j, mirroring the parallel composition of all trace features at each timestamp. Roughly speaking, the f_1 transformation function generates a process able to reproduce the sequential alternancy of features for a specific vehicle. Thus, it reproduces how the vehicle dynamics change during the trip.

The second transformation function is f_2, and it produces a formal model that mimics the parallel composition of the vehicles during their trip by synchronizing their evolution at each instant of time. More in detail, we define a synchronization process *sink* coordinating all vehicles at each timestamp, *i.e.,* all vehicles are scheduled by the orchestrator process *sink* having the following definition:

$$sink \stackrel{\text{def}}{=} sync_1 \ldots . sync_n . \overline{go}_1 \ldots \overline{go}_n . sink$$

Thus, this formal model comprises the parallel composition of each vehicle, each modelled as a process of features, with an embedded synchronizing process that, at each moment, can coordinate the vehicle evolution during the trip. More in detail, we show in the following a simple example of the complete formal model.

$$\begin{aligned}
p_{00} &\stackrel{\text{def}}{=} (increasingAcc0.DONE \| increasingVel0.DONE); (\overline{synk_0}.go_0.p_{01}) \\
&\vdots \\
p_{n0} &\stackrel{\text{def}}{=} (increasingAccN.DONE \| increasingVelN.DONE); (\overline{synk_0}.go_0.p_{n1}) \\
sink &\stackrel{\text{def}}{=} sync_1 \ldots sync_n \overline{go}_1 \ldots \overline{go}_n . sink \\
All &\stackrel{\text{def}}{=} (p_{00} \| \ldots p_{n0} \| sink) \backslash \{syc_0, \ldots sync_n, go_0, \ldots go_n\};
\end{aligned}$$

The above definition involves n processes, one for each vehicle, and a *sink* process synchronizes vehicles at each timestamp. The process *ALL* allows the parallel composition of all vehicles synchronized through the sink and all restricted synchronization actions.

Finally, these models jointly with specific behavioural properties of interest are inputs to the Model Checker tool. This tool verifies if the formal model meets the specified properties. If the property holds, the tool returns a true value; otherwise, it returns false.

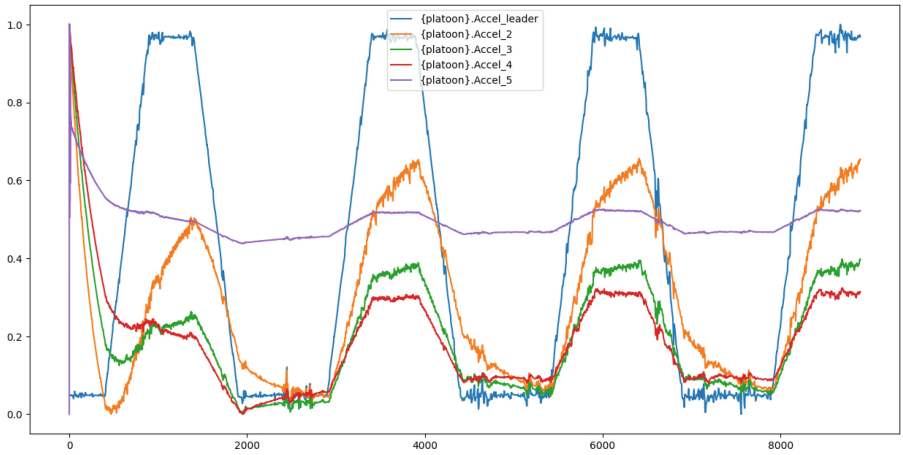

Fig. 2. Normalized acceleration trends.

4.1 Our Case Study

We used traces generated using the approach proposed by Palmieri *et al.* [15]. Traces comprise 8,899 velocity and acceleration samples collected every 0.01 s from 4 cars in a platoon to evaluate our methodology. First, we normalized the samples with the `scikit-learn` [16] method MinMaxScaler: this allowed us to obtain data comprising 0 and 1. Once data had been normalized, and since formal methods require non-numeric samples, we performed the discretization. Observing the nature of the samples, we noticed that trends were repeated cyclically. For velocity, we had a trend formed by constant and increasing velocity that repeated itself. Instead, the acceleration trend comprises four repeating parts: low-level stationary, increasing, high-level stationary, and decreasing. Figure 2 and Fig. 3 show normalized trends of acceleration and velocity, respectively.

We performed two types of discretization. One type was made using the `cut` method from `NumPy` library [10], and a bin equal to three was applied on each sample. The second discretization was performed in two phases: firstly, we used the Augmented Dickey-Fuller test to evaluate if the sample in a window were constant, and in the case of non-constant results, we assessed the slope to define if there were an increasing or decreasing. To apply the statistic test, the `adfuller` method from statsmodel library [19] while the slope was defined using `polyfit` method from `NumPy` library [10]. To determine the best size of the window, we tried several widths (50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550)

5 Preliminary Result

Table 3 shows two possible formulas that can be verified on the formal models created with our approach. Specifically, the first formula checks that when the

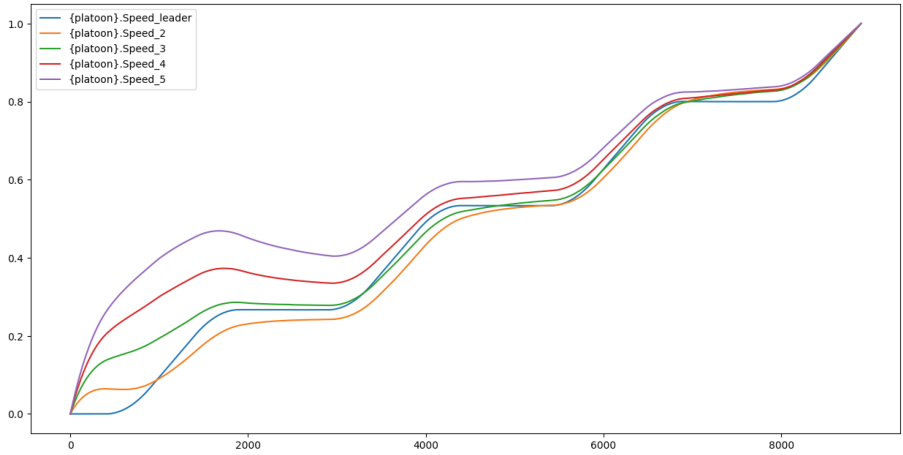

Fig. 3. Normalized velocity trends.

Table 3. Two formulas for checking normal behaviours.

φ_1	$= \mu X.\, \langle increasingAcc0 \rangle\, \varphi_{1_1} \vee \langle -increasingAcc0 \rangle\, X$
φ_{1_1}	$= \mu X.\, \langle increasingAcc2 \rangle\, \varphi_{1_2} \vee \langle -increasingAcc2 \rangle\, X$
φ_{1_2}	$= \mu X.\, \langle increasingAcc3 \rangle\, \varphi_{1_3} \vee \langle -increasingAcc3 \rangle\, X$
φ_{1_3}	$= \mu X.\, \langle increasingAcc4 \rangle\, \mathtt{tt} \vee \langle -increasingAcc4 \rangle\, X$
φ_2	$= \mu X.\, \langle decreasingAcce0 \rangle\, \varphi_{2_1} \vee \langle -decreasingAcce0 \rangle\, X$
φ_{2_1}	$= \mu X.\, \langle decreasingAcce2 \rangle\, \varphi_{2_2} \vee \langle -decreasingAcce2 \rangle\, X$
φ_{2_2}	$= \mu X.\, \langle decreasingAcce3 \rangle\, \varphi_{2_3} \vee \langle -decreasingAcce3 \rangle\, X$
φ_{2_3}	$= \mu X.\, \langle decreasingAcce4 \rangle\, \mathtt{tt} \vee \langle -decreasingAcce4 \rangle\, X$

lead vehicle accelerates, all following vehicles subsequently accelerate in a cascading manner. Conversely, the second formula verifies, in a dual manner to the first, that if there is a deceleration, it propagates throughout the entire platoon.

Table 4 displays the results of verifying two logical formulas, φ_1 and φ_2, across our formal models. More in detail, column **Model ID** indicates the identifier for each model, where each ID corresponds to a sample window size used during model construction. The other two columns, labelled φ_1 and φ_2, represent the outcomes of the formula verifications, with a check mark signifying that the formula is satisfied for that model. From the Table 4, it is evident that both formulas, φ_1 and φ_1, hold for all the tested models, regardless of the sample window size. This result suggests that the behaviour being checked remains consistent across these preliminary models. However, it's important to note that this represents an initial, simplified analysis. In future work, we plan to verify more complex formulas using traces that include data from real-world vehicles and scenarios involving potential attacks. Results will allow for a deeper exam-

Table 4. Results for φ_1 and φ_2 properties reporterd in Table 3.

Model ID	φ_1	φ_2
50	✓	✓
150	✓	✓
200	✓	✓
250	✓	✓
300	✓	✓
350	✓	✓
400	✓	✓
450	✓	✓
500	✓	✓
550	✓	✓

ination of the system's robustness and security under varied and more realistic conditions.

6 Challenges and Limitations

Although our preliminary approach shows promising results in verifying platoon behaviour, it also presents several challenges and limitations. Overcoming these issues is crucial to enhance our method's robustness, scalability, and practical applicability.

The model-checking process inherently faces the "state explosion" problem, where the number of system states grows exponentially with the addition of each vehicle or behavioural parameter. The verification process can be computationally prohibitive in large platoons or complex environments, limiting its use in real-time applications or more extensive networks. Abstraction techniques can be employed to simplify the model by focusing only on critical states or transitions. Techniques such as symbolic and bounded model checking can reduce the number of states by analyzing only a subset of behaviours or time intervals. Additionally, compositional verification, which verifies each component or vehicle independently before integrating the results, can help manage complexity in larger systems.

Furthermore, our method discretizes the continuous data, such as velocity and acceleration, into categories of trends. It makes the analysis task easier but might lose important information and slight deviations or subtle behavioural changes that could indicate problems. Adaptive discretization can address the issues where granularity changes dynamically with real-time data trends. Another helpful method could be the application of multiresolution discretization, using finer granularity in the most critical sections, *e.g.,* during rapid acceleration or braking-in, to avoid missing essential details without overwhelming the verification process.

Finally, verification based only on traces of simulated executions cannot truly capture real-world conditions, such as variable road conditions, environmental considerations, or unexpected interactions with human-driven vehicles. This limitation makes the verification outcome for platoon behaviour less representative of performance in diverse environments. A hybrid testing approach that combines real-world sensor data with simulated scenarios could improve the accuracy of execution traces. Additionally, using digital twins that mirror real-time conditions from live environments could yield more realistic simulations of actual platoon behaviour. This approach would likely strengthen the model's robustness by enabling verification across a broader range of potential scenarios.

7 Conclusion

Given the wide spread of driverless vehicle technology and the critical role these emerging technologies play in increasing road safety, it is necessary to develop new methods to verify the reliable performance of these autonomous systems. For these reasons, our work proposes a preliminary approach to verify vehicle platoon behaviour using its execution traces. More in detail, we propose a preliminary model checking-based approach aiming to rigorously verify a platoon's abstract model built from its execution traces and using behavioural properties expressed in temporal logic. The results achieved seem promising. In the future, we plan to improve platoon modelling jointly with its property specifications to guarantee and verify the security aspects of platoon systems.

Acknowledgment. This work is partially funded by the European Union - NextGenerationEU - National Recovery and Resilience Plan (NRRP) - MISSION 4 COMPONENT 2, INVESTMENT N. 1.1, CALL PRIN 2022 PNRR D.D. 1409 14-09-2022 - FORESEEN: FORmal mEthodS for attack dEtEction in autonomous driviNg systems, CUP N.I53D23006130001.

References

1. ISO: 20035:2019, Intelligent transport systems - Cooperative adaptive cruise control systems (CACC) - Performance requirements and test procedures (2019)
2. Accelerate safety measures to reduce road traffic deaths: Who (2024). https://www.who.int/southeastasia/news/detail/02-09-2024-accelerate-safety-measures-to-reduce-road-traffic-deaths-who
3. Ali, P.J.M.: Investigating the impact of min-max data normalization on the regression performance of k-nearest neighbor with different similarity measurements. ARO-The Sci. J. Koya Univ. **10**(1), 85–91 (2022)
4. Alvarez, L., Horowitz, R.: Safe platooning in automated highway systems part i: safety regions design. Veh. Syst. Dyn. **32**(1), 23–55 (1999)
5. Bergenhem, C., Huang, Q., Benmimoun, A., Robinson, T.: Challenges of platooning on public motorways. In: 17th World Congress on Intelligent Transport Systems, pp. 1–12 (2010)

6. Braiteh, F.E., Bassi, F., Khatoun, R.: Platooning in connected vehicles: a review of current solutions, standardization activities, cybersecurity, and research opportunities. IEEE Trans. Intell. Veh. (2024)
7. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (2001)
8. Cleaveland, R., Sims, S.: The NCSU concurrency workbench. In: CAV. Springer (1996)
9. Hamilton, J.D.: Time Series Analysis. Princeton University Press, Princeton (2020)
10. Harris, C.R., et al.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2
11. Ju, Z., Zhang, H., Li, X., Chen, X., Han, J., Yang, M.: A survey on attack detection and resilience for connected and automated vehicles: from vehicle dynamics and control perspective. IEEE Trans. Intell. Veh. **7**(4), 815–837 (2022)
12. Kim, H., Jeong, Y., Choi, W., Lee, D.H., Jo, H.J.: Efficient ECU analysis technology through structure-aware can fuzzing. IEEE Access **10**, 23259–23271 (2022)
13. Lee, G., Jung, J.I.: Decentralized platoon join-in-middle protocol considering communication delay for connected and automated vehicle. Sensors **21**(21), 7126 (2021)
14. Milner, R.: Communication and concurrency. PHI Series in computer science, Prentice Hall (1989)
15. Palmieri, M., Quadri, C., Fagiolini, A., Bernardeschi, C.: Co-simulated digital twin on the network edge: a vehicle platoon. Comput. Commun. **212**, 35–47 (2023)
16. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
17. Petrillo, A., Pescapé, A., Santini, S.: A collaborative approach for improving the security of vehicular scenarios: the case of platooning. Comput. Commun. **122**, 59–75 (2018)
18. Schwab, A., Lunze, J.: Vehicle platooning and cooperative merging. IFAC-PapersOnLine **52**(5), 353–358 (2019)
19. Seabold, S., Perktold, J.: Statsmodels: econometric and statistical modeling with python. In: 9th Python in Science Conference (2010)
20. Stirling, C.: An introduction to modal and temporal logics for CCS. In: Concurrency: Theory, Language, and Architecture, pp. 2–20 (1989)
21. Thormann, S., Schirrer, A., Jakubek, S.: Safe and efficient cooperative platooning. IEEE Trans. Intell. Transp. Syst. **23**(2), 1368–1380 (2020)
22. Tsugawa, S.: An overview on an automated truck platoon within the energy its project. IFAC Proc. Vol. **46**(21), 41–46 (2013)
23. Wang, Z., Wei, H., Wang, J., Zeng, X., Chang, Y.: Security issues and solutions for connected and autonomous vehicles in a sustainable city: a survey. Sustainability **14**(19), 12409 (2022)
24. Winner, H., Danner, B., Steinle, J.: Adaptive cruise control. Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort, pp. 478–521 (2009)