Towards System-Level Testing with Coverage Guarantees for Autonomous Vehicles

István Majzik*, Oszkár Semeráth*†, Csaba Hajdu*, Kristóf Marussy*†, Zoltán Szatmári*, Zoltán Micskei*, András Vörös*†, Aren A. Babikian‡ and Dániel Varró*†‡

*Dept. of Measurement and Information Systems, Budapest University of Technolgy and Economics, Budapest, Hungary
†MTA-BME Lendület Cyber-Physical Systems Research Group, Budapest, Hungary
‡Dept. of Electrical and Computer Engineering, McGill University, Montréal, QC, Canada

***Abstract*—Since safety-critical autonomous vehicles need to interact with an immensely complex and continuously changing environment, their assurance is a major challenge. While systems engineering practice necessitates assurance on multiple levels, existing research focuses dominantly on component-level assurance while neglecting complex system-level traffic scenarios. In this paper, we aim to address the system-level testing of the situation-dependent behavior of autonomous vehicles by combining various model-based techniques on different levels of abstraction. (1) Safety properties are continuously monitored in challenging test scenarios (obtained in simulators or field tests) using graph query and complex event processing techniques. To precisely quantify the coverage of an existing test suite with respect regulations of safety standards, (2) we provide qualitative abstractions of causal, temporal, or geospatial data recorded in individual runs into situation graphs, which allows to systematically measure system-level situation coverage (on an abstract level) wrt. safety concepts captured by domain experts. Moreover, (3) we can systematically derive new challenging (abstract) situations which justifiably lead to runtime behavior which has not been tested so far by adapting consistent graph generation techniques, thus increasing situation coverage. Finally, (4) such abstract test cases are concretized so that they can be investigated in a real or simulated context.**

***Index Terms*—Model-based testing, Autonomous vehicles, Cyber-Physical Systems, System-level testing, Test coverage**

I. Introduction

Many autonomous vehicles are controlled by advanced machine learning techniques in order to continuously interact with a complex and dynamically changing environment. However, their assurance has become a major challenge due to their safety-critical nature. Upfront design-time verification is regarded as infeasible for autonomous vehicles [1]–[3]. Instead, runtime assurance based on runtime monitoring and simulation is dominantly used. The behavior of an autonomous vehicle is extensively tested in a large number of scenarios investigated in a simulator or real traffic conditions.

Unfortunately, incidents typically happen due to an unexpected combination of rare events. As a consequence, merely *increasing the number of tested kilometers by randomly selected test scenarios provides no safety guarantees* due to the extremely large spaces of input parameters. Moreover, while safety engineering best practice necessitates assurance on multiple levels, the vast majority of existing research focuses only on component-level assurance, while very little assurance is attempted on the system-level. For example, even if an autonomous vehicle is able to provide safe control in different weather conditions (component-level assurance), one still needs to assure that the car is compliant with the regulations in complex traffic situations (system-level assurance).

In this paper, we present a *system-level assurance technique* which provides black-box testing of autonomous vehicles with situation coverage guarantees by combining various model-based techniques. Assuming that challenging test scenarios can be investigated in simulators (or in real traffic), we (1) derive qualitative abstraction of causal, temporal, or geospatial data of concrete trajectories recorded in simulators (or in real traffic) into abstract (and domain-specific) situation graphs driven by recommendations of safety standards, and (2) use runtime monitoring based on graph query and complex event processing techniques to continuously check safety properties derived from domain-specific safety standards. (3) We measure (the situation) coverage of existing system-level tests on abstract situation graphs using model diversity metrics. Moreover, (4) we can systematically derive new challenging situation graphs as abstract test cases which justifiably lead to runtime behavior which has not been tested so far by adapting consistent graph generation techniques. Finally, (5) such abstract test situations need to be concretized by context generation to feed them into simulators as environment descriptions.

To our best knowledge, this paper is the first to address system-level assurance challenges for autonomous vehicles with (situation) coverage guarantees. To increase the practical relevance of our research, our framework (1) complies with recommendations of safety standards, (2) builds on various model-based techniques developed for different applications. Furthermore, the key concepts, challenges and ideas are presented in the context of an autonomous tram case study.

II. Safety of autonomous vehicles: An overview

Figure 1 illustrates the typical setup of an *autonomous vehicle* (based on [4] and standards [5], [6]): the automated driving system (ADS) is operating in a complex *environment (context)*. The environment of the system is observed by *sensors* (e.g. camera, LIDAR and radar subsystems to measure distance). The *goal* of the autonomous system (e.g. navigation or route selection) is determined by the driver or passenger through *human/machine interface* components. Then the ADS makes *decisions* based on the observed situation, and controls the vehicle through *actuators*.

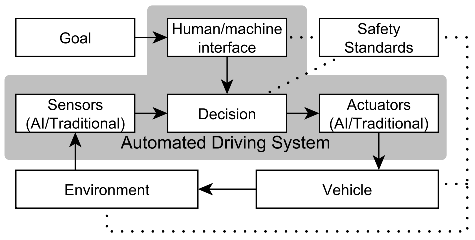

Fig. 1. High-level components in autonomous vehicles

In [4], requirements on the safe driving behavior of autonomous vehicles are classified into five categories (where safety on one level assumes safety on the previous level):

1) *Vehicle stability*: First, the stable control of the vehicle is required (which includes e.g. skid and roll stability).

2) *Assured clear distance ahead (ACDA)* is the path distance *ahead* of the vehicle that needs to be clear for driving so that the ADS can bring the vehicle to a halt. ACDA measures include different sight distances, which depend on road geometry and the executed maneuver, and the stopping sight distance.

3) *Minimum separation*: The ADS has to assure minimum separation between the subject vehicle and other dynamic and static objects (e.g. pedestrians vs. trees).

4) *Traffic regulations*: Traffic regulations are formal and dominantly safety-related traffic rules enforced by law in a geographical area. They control traffic conflict resolution (e.g. yielding rules at intersections, passing rules) and prescribe how different road users use the roadway (e.g. traffic directions, lane restrictions, and parking restrictions).

5) *Driving best practices*: Driving best practices are informal traffic rules that refine and complement the formal rules. Examples include rules about how early to signal turns and how to respond to tailgating. Among others, an ADS-operated vehicle should use best practices to anticipate, recognize, and properly respond to likely mistakes of human road users.

Assumptions. While the vast majority of existing research addresses Requirements 1-2, we assume that those safety levels are sufficiently addressed. Thus, we focus on system-level Requirements 3-4 by capturing traffic regulations and situations as (graph) models. Moreover, we assume that sensing and actuator components are adequately reliable, which is a significant challenge in itself but falls outside the scope of the current work. Finally, we assume that testing and simulation are the primary means of assurance for autonomous vehicles.

Running example. Our approach is illustrated on an autonomous tram driving scenario motivated by the widespread use of trams in many modern European cities. While assuring safety for autonomous trams wrt. Requirements 1 and 2 are easier compared to self-driving cars, it is almost equally challenging wrt. system level requirements (like minimum separation and traffic regulations) as the context of the vehicles are very complex. Furthermore, we believe that the presented concepts can be adapted for other autonomous vehicles.

III. Related work and open challenges

Many challenges of assuring autonomous systems are presented in [1], [2], [7], [8] and in domain-specific surveys [9], [10]. Here, we highlight only a subset of the open challenges relevant to our research. Decades of engineering experience on the assurance of software-intensive safety-critical systems prescribes that various V&V activities need to be carried out on multiple levels (e.g. component, system), thus we categorize related work accordingly.

a) Component level assurance for autonomous systems: As machine learning techniques e.g. deep neural networks (DNNs) frequently control autonomous driving components, recent research started to focus on supporting testing [11], [12], formal verification [13], [14] or deriving adversarial examples to reveal sensitivity problems [15] in DNNs. This line of research is mostly restricted to component-level assurance to investigate the behavior of a single autonomous component to address Requirements 1–2, while the challenge of system-level safety with many autonomous components and contextual objects is left open (Requirements 3–4). As such, while we adapt some high-level ideas, this adaptation is challenging due to the difference of assurance levels (see also [5], [16]).

b) System-level approaches: Formal verification of the causal behavior of self-driving cars was proposed in [17]. Traffic situations and possible reactions of vehicles are verified on an abstract level. Simulation is widely used to analyze the system-level behavior of autonomous systems [18]. However, the construction of diverse test scenarios is not yet solved.

c) System-level context generation: Several testing approaches [19]–[22] use search-based techniques to (1) assemble prototypical test contexts for autonomous agents, (2) simulate the agents in adversarial environments, (3) monitor their compliance to designated safety properties, and (4) synthesize more challenging test contexts by using the degree of deviation as an objective function (i.e. a test context is better if the autonomous agent behaves poorly). But existing context generators [19], [23] can only derive very simple test contexts which may help find bugs but provide no coverage guarantees. Real-world situation graphs need to represent complex temporal, spatial and causal information, which is currently not supported when generating relevant contexts and scenarios for system-level assurance of autonomous systems.

d) Runtime monitoring techniques: The complexity of assuring autonomous systems also lies in the continuous changes of requirements, goals, behavior, execution platform or the context (environment) of the system. As such, runtime monitoring techniques are frequently used [7], [24], [25] for detecting relevant complex events from raw data of sensor reads over a runtime model. In this paper, we adapt existing model-based techniques (like [26]) for runtime monitoring purposes for the assurance of autonomous vehicles.

e) The industrial perspective: The number of safely driven miles driven is a frequently used measure of reliability [3] used by leading industrial solutions like Waymo [27] or Voyage.auto [28]. While the self-driving industry aims to increase the number of miles for testing by driving in both a simulated environment and in real traffic, even billions of miles provide only limited assurance from a safety perspective as stochastic simulations will likely *miss the extremely rare combination of unexpected events and situations*.

Open challenges We identified several major open research challenges for the system-level assurance of autonomous vehicles to address the following safety requirements.

1) How to evaluate the adherence of complex context-dependent behavior to requirements of safety standards in test scenarios?
2) How to measure system-level test coverage for autonomous vehicles to justify that certain traffic situations have already been investigated by existing tests?
3) How to generate relevant test contexts for autonomous vehicles to systematically increase test coverage automatically by synthesizing challenging new situations?

IV. Towards System-Level Testing with Coverage

A high-level overview of our system-level testing approach is illustrated in Figure 2.

Simulation. We assume that *simulators* (or real test vehicles) are available to investigate the behavior of the autonomous vehicles in designated trajectories defined by *test contexts*.

Qualitative abstraction. We use graph queries to abstract geospatial, causal and temporal information observed in trajectories in test contexts into *situation graphs*, where relations are represented as labeled graphs. Those situation graphs are maintained during the simulation.

Runtime monitoring is then carried out on the changes of situation graphs by complex event processing techniques [26] with precise formal semantics.

Situation coverage. We measure the coverage of existing test scenarios on an abstract (situation) level by adapting model diversity metrics [25] and graph shapes [29]. Our situation coverage provides various semantic completeness guarantees on the level of covered situations.

Situation generation. We automatically generate challenging new situations as abstract test cases by using consistent and diverse graph generation techniques [30] and measure the coverage of possible situations.

Context generation. Finally, we aim to concretize abstract situations into concrete test contexts, that increases the coverage and robustness of the test suite.

A. Simulation of test scenarios

In our paper, we define *test scenarios* as a sequence of snapshots connected via behavioral or temporal events. Test scenarios can be captured e.g. as graphical scenarios [28], [31].

Simulators are widely used for testing vision subsystems, analyzing control systems and various other domains. In our setting, the test scenarios are executed on a simulator platform and the simulation data is collected for further evaluation/analysis. Figure 3 depicts a snapshot of the simulator (developed using the Gazebo robot simulation framework targeting ROS-based systems) executed on the tram example. Effective simulation requires two inputs: (1) configuration of the (physical) environment and the (2) test scenario to be executed. The inputs are provided in a model representation and converted to the input format of the simulator. The output of the simulator is the stream of kinematic data for the objects that will be transformed into a graph representation at the runtime monitoring step. In the example, the position data of the tram, pedestrians and other vehicles and also the speed and velocity data of the SUT are collected for further analysis.

B. Qualitative abstraction of scenarios

We propose to use qualitative abstraction to map the output of the simulator into abstract situations where requirements prescribed by safety standards can be monitored on a higher-level of abstraction.This qualitative abstraction is implemented as a chain of reactive graph query and transformation steps.

First, the *context model* describes concrete scene of the environment, which can be constructed on a test field or instantiated by a simulator. A context model explicitly represents the coordinates of the objects by numeric attributes. We expect that the context model is able to represent all information for the simulation, and the simulator is able to continuously synchronize the context model with kinematic data to act as a *runtime model*. Therefore, formalized dangerous situations (e.g. SUT gets near to another object) can be observed on the changes of the context models (e.g. by comparing coordinates).

Next, a *situation model* is obtained by abstracting values and structures of the context model into relations and objects to highlight information relevant to safety requirements. The situation model is continuously updated by model queries upon changes in the context model. Since situation models incorporate domain-specific knowledge from standards, situation coverage is specific to the taxonomy defined by safety experts.

Figure 4 depicts an extract from the metamodel of the tram example capturing some key concepts and relations relevant for safety monitoring. Two main types of objects are observed: stationary (e.g. *rails*, pedestrian *crossings* and *sidewalks*) and moving objects (such as *Trams* and *Pedestrians*). In the context model (left blue on Figure 4) positions are modeled explicitly. In the situation model (right green on Figure 4) spatial relations are abstracted to references (like *on* or *next*).

To compute the derived relations along the qualitative abstraction chain, we rely upon graph queries (using the VIATRA syntax [32]). Sample graph queries required for computing the *next* and *on* relations by spatial abstraction are depicted below. Similar qualitative abstraction can be applied to velocity, acceleration or other kinematic values.

```
pattern setNext(SceneObject a, SceneObject b) {
 SceneObject.x(a,ax); SceneObject.x(b,bx);
 SceneObject.y(a,ay); SceneObject.y(b,by);
 neg find setOn(a,b);
 check(√((ax − bx)² + (ay − by)²) < 25);}
pattern setOn(SceneObject a, SceneObject b) {
 SceneObject.x(a,ax);SceneObject.x(b,bx);check(|ax − bx| < ε);
 SceneObject.y(a,ay);SceneObject.y(b,by);check(|ay−by| < ε);}
```

Figure 5 depicts a sample scenario consisting of three snapshots of situation models derived from simulator data where the *FastMoving* tram approaches a *Crossing*, a *Pedestrian*

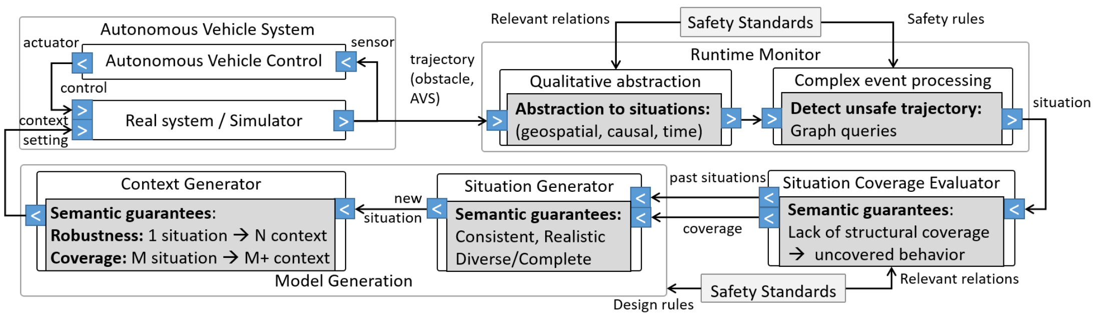
Autonomous Vehicle System
actuator
sensor
Autonomous Vehicle Control
control
context setting
Real system / Simulator
trajectory (obstacle, AVS)
Relevant relations
Safety Standards
Safety rules
Runtime Monitor
Qualitative abstraction
Abstraction to situations: (geospatial, causal, time)
Complex event processing
Detect unsafe trajectory: Graph queries
situation
Context Generator
Semantic guarantees:
Robustness: 1 situation → N context
Coverage: M situation → M+ context
new situation
Situation Generator
Semantic guarantees: Consistent, Realistic Diverse/Complete
past situations
coverage
Situation Coverage Evaluator
Semantic guarantees: Lack of structural coverage → uncovered behavior
Model Generation
Safety Standards
Relevant relations
Design rules

Fig. 2. Overview of the approach

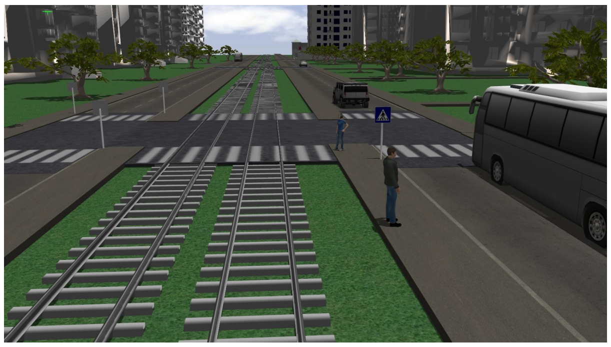

Fig. 3. Executing the example in a simulator

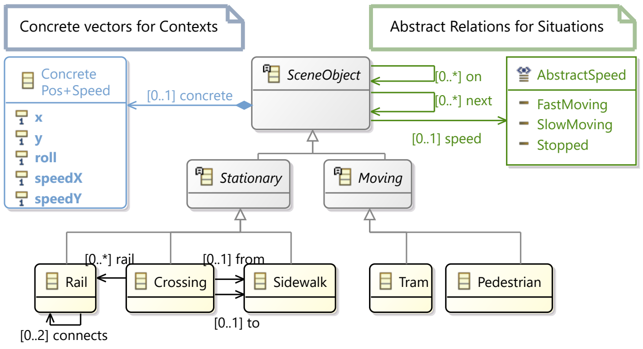

Fig. 4. Metamodel to represent the simulator output

walks through the *Crossing*, and as a reaction, the tram reduces its speed and becomes *Stopped*.

C. Runtime monitoring

Safety standards and regulations define high-level requirements such as minimum separation distance between objects and other practices. From this, safety engineers derive low-level, executable specifications to be used as a test oracle. Geospatial, causal and timing aspects of the behavior have to be incorporated by using qualitative abstraction to capture the many aspects of the specification.

Safety requirements to be monitored are expressed over the abstract situation model constructed in former steps. For that purpose, we first exploit the expressive language of graph patterns executed over the runtime model to detect complex situations (e.g. the pedestrian is on the crossing).

Next, we need to detect relevant temporal (or causal) patterns. The combination of incremental graph query and complex event processing [26] allows to turn relevant model changes into a stream of events to detect relevant temporal patterns, but the handling of continuous signals is problematic. To comply with runtime monitoring techniques based on formal specification languages, we chose Signal Temporal Logic (STL) [33] to monitor the temporal behavior of the autonomous vehicle, which handles continuous signals received directly from the simulator in addition to discrete event semantics. In our tram example, complex graph patterns are used to detect when *Pedestrians* are *on* or *next* to the crossing.

```
pattern pedestrianNextToCrossing(t,p) {
 Tram.next(t,r); Crossing.on(c,r);
 Sidewalk.next(s,c); Pedestrian.on(p,s); }
pattern pedestrianOnCrossing(t,p) {
 Tram.next(t,r); Crossing.on(c,r); Pedestrian.on(p,c); }
pattern tramStopped(t) { Tram.speed(t,::Stopped); }
pattern tramSlowed(t) { Tram.speed(t,::SlowSpeed); }
```

The following STL formulae represent safety properties that (φ_1) a tram should stop within 3 seconds when a pedestrian is on the crossing, while formula (φ_2) ensures that the tram remains stopped until the crossroad is free.

$$\begin{aligned}\varphi_1 &= \mathcal{G}(\texttt{pedestrianOnCrossing} \Rightarrow \mathcal{F}_{[0,3]}\ \texttt{tramStopped})\\ \varphi_2 &= \mathcal{G}\big(\texttt{pedestrianOnCrossing} \wedge \texttt{tramStopped} \Rightarrow\\ &\quad\ (\texttt{tramStopped}\ \mathcal{U}\ \neg\texttt{pedestrianOnCrossing})\big)\end{aligned}$$

A similar safety property can capture that the tram should slow down when the pedestrian is next to a crossing, while a liveness property may prescribe that the tram will eventually leave the crossing when no pedestrians are around.

Monitors can detect violations of such properties and experts can investigate the nonconforming behavior to safety standards. Monitors that do not detect violation can still be used to enhance the test coverage: non-traversed parts of the monitor can drive the test generation into the direction of untested behavior of the autonomous vehicle.

D. System-level coverage metrics over situation models

Due to the qualitative abstraction, one situation model corresponds to several (frequently, an infinite number of)

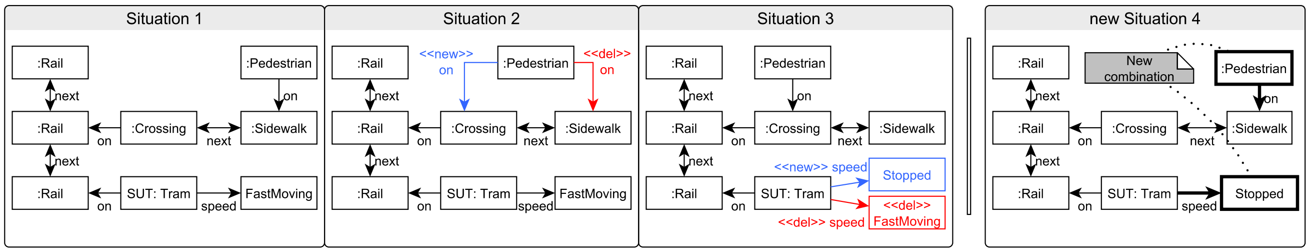

Fig. 5. Model representation of the abstract situations

trajectories derived from context models. As such, a situation model serves as an *abstract test case* which corresponds to multiple concrete trajectories of the simulated system. First, (1) we need to establish a concept of *system-level coverage* to ensure that if two abstract situations are different, then the corresponding simulations were investigating different concrete behavior from a safety perspective. Moreover, (2) we need to enforce the *robustness of the abstraction*, i.e. a specific concrete trajectory observed in the system (on the context model) needs to be uniformly mapped to one or more abstract situations which leads to general observability and uncertainty challenges.

First, we propose to use *shape analysis as an equivalence partitioning technique* on the level of situation models. Shape analysis [29] provides sophisticated graph abstraction to discover semantic differences between graph models observable by graph patterns (as predicates). If two situations S_1 and S_2 have different shapes, then there has to be a graph pattern $\mathtt{p}$ (with properties specific to the shape analysis technique) which can distinguish the two situations: $\mathtt{p}(S_1) \neq \mathtt{p}(S_2)$, i.e. a property holds in one case but not in the other. Additionally, shape analysis techniques ensure the finiteness of shapes. Therefore, using shape analysis techniques in testing (see e.g. [34]) a *finite abstract test suite can possibly discover all dangerous situations* that can be characterized by such graph predicates defined along concepts of safety standards.

We propose *situation coverage as a system-level metric* to investigate the (abstract) coverage of an existing test suite, which counts the number of different shapes of situation models reached while executing the test suite divided by the number of possible shapes (which can be large but still finite).

While the precise handling of situation coverage is outside the scope of the current paper, the key idea is illustrated in Figure 5, *Situation 1-3* describes three states of a system with different situations obtained along a single trajectory of the simulation. However, *new Situation 4* describes a new situation, where the tram is stopped while the pedestrian is on the sidewalk, which was not tested before by the simulator.

E. Generation of abstract situations

We propose to automatically derive new abstract situations (considered as abstract test cases) by consistent graph model generation techniques [30], [35] according to several test strategies [21]. Taking a metamodel (e.g. situation metamodel in Figure 4), and a set of consistency constraints as input, these generic techniques can automatically synthesize a set of consistent situation models as output. Consistency constraints are derived from (1) safety standards [5], [6], (2) domain-specific knowledge (e.g. the rails have to be connected, and they are next to each other), or (3) properties of the abstraction (e.g. triangular geometric constraints for *on*).

In particular, a situation model generator should derive a diverse set of consistent, realistic models, ideally, with some completeness guarantees such as provided in [30], [34]. As a result, the set of generated models continuously increases situation coverage. In the context of our running example, we can derive all situations to cover all abstract pedestrian-tram interaction in a given crossing.

F. Concretization into test contexts

Finally, the abstract test cases represented by situation models need to be turned into concrete (executable) test scenarios (represented by context models) that can be investigated in the simulator (or in a real-life setting). Thus, we need to derive (1) a separate concrete context model for each abstract situation to *systematically increase coverage*, and (2) several test contexts for the same situation to provide *robustness testing*.

Abstract situation models are concretized into context models by filling the attribute values based on the abstract relations. Physical and numeric constraints over the undefined attributes are derived in accordance with the abstract relations in the situation model. This constraint satisfaction problem can be solved by advanced Satisfiability Modulo Theories (SMT) solvers where each solution can be interpreted as a valid context model of the situation model. Then the simulator model can be generated from the context model by supplying simulator-specific information. Our goal is to generate a representative concrete context model for each situation model to ensure proper test coverage. In our example, Situation 4 can be turned into a context where the pedestrian stands on the sidewalk next to the crossing, while the tram's speed is set to zero.

Robustness tests can also be generated by configuring the solver to generate boundary values for attributes along the constraints that define the qualitative abstraction. In our example, such a robustness test would place a pedestrian at the edge of the sidewalk, right next to the crossing, when the qualitative abstraction may not decide successfully if he is on the sidewalk or already in the crossing. Finally, concrete test contexts can be derived to explain how a monitored property is satisfied.

V. Conclusion

In this vision paper, we addressed the system-level testing of autonomous vehicles. Our approach is based on a novel combination of model-based techniques: we adapt efficient model queries and transformations to compute qualitative abstractions of simulation runs with expressive monitoring techniques. Simulator input is provided by graph model generators constructing a diverse set of consistent input models with coverage guarantees. Concretization techniques are devised to produce one or more executable test contexts for simulator evaluation. To our best knowledge, this approach is the first proposal to address the system level assurance challenges of autonomous vehicles with (situation) coverage guarantees.

In future work, we plan to refine the building blocks of the approach presented in the paper. 1) Suitable domain-specific languages need to be defined to express scenarios, situations and contexts in a rich, but manageable way. Based on our previous experience [36], a key observation is that numerous iterations are required with the domain experts to find the right level of abstraction of these models, where safety concepts can be expressed via combinations of relations in the model. 2) Scalable tools are available for generating abstract situations [30] and transforming event streams [25], [26]. However, concretizing sequences of abstract situations is algorithmically and computationally challenging. 3) Finally, existing test strategies [19], [21] could be refined to include model diversity metrics [34] as fitness functions.

Acknowledgments This work was partially supported by the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC), the NSERC RGPIN- 04573-16 project, MTA-BME Lendület Cyber-Physical Systems Research Group, the ÚNKP-18-3 New National Excellence Program of the Ministry of Human Capacities, and the FRQNT scholarship.

References

[1] P. Helle, W. Schamai, and C. Strobel, "Testing of autonomous systems – challenges and current state-of-the-art," *INCOSE International Symposium*, vol. 26, no. 1, pp. 571–584, 2016.

[2] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE Int. J. Trans. Safety*, vol. 4, no. 1, 2016.

[3] N. Kalra and S. M. Paddock, "How many miles of driving would it take to demonstrate autonomous vehicle reliability?" RAND Corporation, Tech. Rep. RR-1478-RC, 2016.

[4] K. Czarnecki, "On-road safety of automated driving system - taxonomy and safety analysis methods," U. of Waterloo, Tech. Rep., 2018.

[5] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, SAE Std. J3016, 2018.

[6] *Road vehicles – Functional safety*, ISO Std. 26 262, 2018.

[7] G. Tamura, N. M. Villegas, H. A. Müller, J. P. Sousa, B. Becker, G. Karsai, S. Mankovskii, M. Pezzè, W. Schäfer, L. Tahvildari, and K. Wong, *Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems*, 2013, pp. 108–132.

[8] L. Briand, S. Nejati, M. Sabetzadeh, and D. Bianculli, "Testing the untestable: Model testing of complex software-intensive systems," in *Int. Conf. on Software Engineering Companion*, 2016, pp. 789–792.

[9] I. de Sousa Santos, R. M. de Castro Andrade, L. S. Rocha, S. Matalonga, K. M. de Oliveira, and G. H. Travassos, "Test case design for context-aware applications: Are we there yet?" *Information and Software Technology*, vol. 88, pp. 1 – 16, 2017.

[10] J. Guiochet, M. Machin, and H. Waeselynck, "Safety-critical advanced robots: A survey," *Robot Auton Syst*, vol. 94, pp. 43 – 52, 2017.

[11] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *SOSP*. ACM, 2017, pp. 1–18.

[12] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Int. Conf. on Software Engineering*, ser. ICSE. ACM, 2018, pp. 303–314.

[13] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *Computer Aided Verification*, 2017, pp. 97–117.

[14] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Proc. of CAV*, 2017, pp. 3–29.

[15] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial Intelligence Safety and Security*. Chapman and Hall/CRC, 2018, pp. 99–112.

[16] C. D. Nguyen, A. Perini, C. Bernon, J. Pavón, and J. Thangarajah, "Testing in multi-agent systems," in *Agent-Oriented Software Engineering X*, 2011, pp. 180–190.

[17] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *CoRR*, vol. abs/1708.06374, 2017.

[18] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Sim-ATAV: Simulation-based adversarial testing framework for autonomous vehicles," in *Int. Conf. on Hybrid Systems*, 2018, pp. 283–284.

[19] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Int. Conf. on Software Engineering (ICSE)*, 2018, pp. 1016–1026.

[20] M. Z. Iqbal, A. Arcuri, and L. Briand, "Environment modeling and simulation for automated testing of soft real-time embedded software," *Software & Systems Modeling*, vol. 14, no. 1, pp. 483–524, 2015.

[21] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik, "A concept for testing robustness and safety of the context-aware behaviour of autonomous systems," in *Agent and Multi-Agent Systems*, 2012, pp. 504–513.

[22] C. D. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck, "Evolutionary testing of autonomous software agents," *Autonomous Agents and Multi-Agent Systems*, vol. 25, no. 2, pp. 260–283, 2012.

[23] M. Mossige, A. Gotlieb, and H. Meling, "Testing robotized paint system using constraint programming: An industrial case study," in *Testing Software and Systems*, 2014, pp. 145–160.

[24] B. H. C. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. M. Villegas, *Using Models at Runtime to Address Assurance for Self-Adaptive Systems*. Springer, 2014, pp. 101–136.

[25] M. Búr, G. Szilágyi, A. Vörös, and D. Varró, "Distributed graph queries for runtime monitoring of cyber-physical systems," in *Fundamental Approaches to Software Engineering*, 2018, pp. 111–128.

[26] I. Dávid, I. Ráth, and D. Varró, "Streaming model transformations by complex event processing," in *Proc. of MODELS*, 2014, pp. 68–83.

[27] Waymo Inc., "On the road to fully self-driving," Tech. Rep., 2018, Waymo Safety Report.

[28] Voyage Inc., *Open Autonomous Safety*, 2019, https://oas.voyage.auto/.

[29] A. Rensink and D. Distefano, "Abstract graph transformation," *Electron Notes Theor Comput Sci*, vol. 157, no. 1, pp. 39–59, 2006.

[30] O. Semeráth, A. S. Nagy, and D. Varró, "A Graph Solver for the Automated Generation of Consistent Domain-Specific Models," in *Int. Conf. on Software Engineering*. ACM, 2018 2018.

[31] H. Waeselynck, Z. Micskei, N. Rivière, A. Hamvas, and I. Nitu, "Termos: A formal language for scenarios in mobile computing systems," in *MobiQuitous*, ser. LNICS, 2012, pp. 285–296.

[32] G. Bergmann, Z. Ujhelyi, I. Ráth, and D. Varró, "A graph query language for EMF models," in *Theory and Practice of Model Transformations*. Springer, 2011, pp. 167–182.

[33] A. Donzé, T. Ferrère, and O. Maler, "Efficient robust monitoring for STL," in *Computer Aided Verification*. Springer, 2013, pp. 264–279.

[34] O. Semeráth and D. Varró, "Iterative generation of diverse models for testing specifications of DSL tools," in *FASE*, ser. LNCS, vol. 10802. Springer, 2018, pp. 227–245.

[35] E. K. Jackson and J. Sztipanovits, "Towards a formal foundation for domain specific modeling languages," in *Int. Conf. on Embedded Software*. ACM, 2006, pp. 53–62.

[36] D. Honfi, G. Molnár, Z. Micskei, and I. Majzik, "Model-based regression testing of autonomous robots"," in *Int. SDL Forum"*, 2017, pp. 119–135.