Challenges in Autonomous Robotic System Verification

Huan Zhang*,†, Hao Wu*

[1]Computer Science Department, Maynooth University, Ireland

Abstract

This paper addresses the challenges in Autonomous Robotic Systems (ARS) verification. We focus on identifying the limitations of current ARS verification techniques and propose a new approach for verifying ARS using a newly developed specification language called Cyclone.

Keywords

Autonomous Robotic System (ARS), Formal Verification, Graphs

1. Introduction

The advent of ARS has revolutionized industrial production processes. Our society benefits from the efficiency of ARS in multiple areas, such as manufacturing and healthcare. However, as ARS become more complex and adaptive, ensuring their safety becomes increasingly important and challenging. The challenges arise not only from the internal interactions within the systems themselves, but also from their behavior in unpredictable environments. This poses a significant challenge for ARS verification [1, 2, 3]. Specifically, we aim to address the following challenges:

- **Challenge 1:** How can the properties of ARS be effectively verified to ensure that they operate as intended without causing harm?
- **Challenge 2:** What are the properties of ARS that have not yet been fully elucidated, and how can they be incorporated into the verification process?
- **Challenge 3:** Are current verification tools user-friendly enough to be used by users without a background in formal verification?

To tackle these challenges, we have begun reviewing current techniques for verifying ARS. More importantly, we are interested in gaining an overview of the types of properties that each verification technique can address.

2. Current Techniques

So far, we have collected and reviewed 54 papers on autonomous robotic system verification[1]. Among these papers, 24 use model checkers for verification [4, 5, 6, 7, 8, 9], while 21 focus on formal specifications [10, 11, 12, 13]. The remaining papers focus on contract-based verification (5 papers) and runtime verification (4 papers) [10, 14, 15, 16, 17, 18, 19]. Some papers cover different types of ARS, such as autonomous navigation, medical, aerospace, and industrial robots [20, 21]. Many verification techniques focus on three key aspects of an ARS: autonomy, decision-making, and control.

PhD Symposium of the 19th International Conference on Integrated Formal Methods (iFM) at the University of Manchester, UK, 12 November 2024.

*Corresponding author.

†Huan Zhang is supported by John & Pat Hume Scholarship.

huan.zhang.2024@mumail.ie (H. Zhang); haowu@cs.nuim.ie (H. Wu)

http://classicwuhao.github.io/ (H. Wu)

0000-0001-5010-4746 (H. Wu)

[1]Please note that our survey is still ongoing.

For example, Matt Luckcuck et al. propose an approach that uses model checkers to verify the safety properties of an ARS [22, 23]. They focus on establishing a standardized verification framework across multiple aspects of an ARS, such as perception, control, and decision-making.

Ingrand uses model checking to verify the low-level architecture of a robotic system [24]. This approach particularly focuses on control algorithms, communication protocols, and autonomous properties. It ensures that the requirements of the fundamental layer of a robotic system (including real-time decision-making and sensor-actuator coordination) are met. Foughali and Zuepke propose a cross-disciplinary approach that is able to verify real-time systems used by autonomous robots [25]. Their work covers aspects such as autonomy, perception, and control.

After reviewing these papers, we have identified three limitations in the scope of ARS verification. (1) The key aspects of an ARS are not formally defined, which can directly impact the confidence in using verification techniques. (2) The use of a set of verification tools is quite challenging for regular users. Often, these tools require users to have knowledge of a particular formalism, and the learning curve is typically steep. (3) Current techniques are prone to producing incorrect results when unpredictable conditions occur. For example, when the weather changes, an ARS may behave entirely differently. However, such conditions are typically not captured by current techniques.

3. Our Proposed Approach

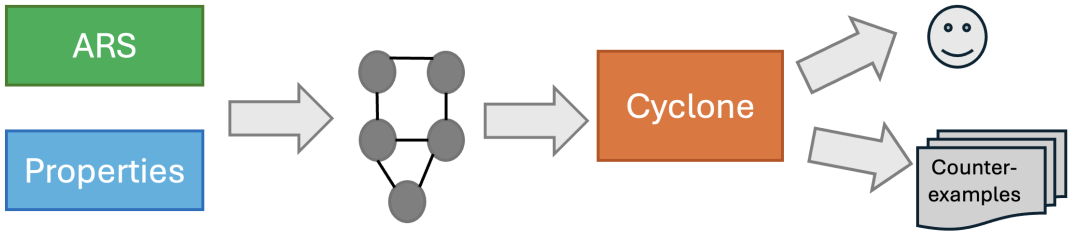

Figure 1: The overview of our proposed approach. An ARS along with its properties are represented using a graph that can be mapped to a Cyclone specification for verification.

To tackle these limitations, we have designed a new specification language called Cyclone [26][2]. Cyclone is based on graphs, and a verification task can be represented as a graph. Cyclone uses a novel algorithm for *bounded checking*. It works by representing a verification task as a graph, which is then reduced to a Satisfiability Modulo Theories (SMT) problem that can be efficiently verified/solved by an SMT solver [27]. An overview of this proposed approach is illustrated in Figure 1.

Cyclone's architecture is shown in Figure 2. Cyclone is written in Java and consists of more than 100k lines of code including build scripts, test cases, configurations and other related projects. We designed about 180 grammar rules for the Cyclone specification language and use ANTLR for generating lexers and parsers. The type checker ensures the type safety of a specification, and produces relevant error messages if there are any semantic errors in a specification. We have designed a *state matrix* as an intermediate representation (IR) to capture all necessary information needed from the front-end of the Cyclone compiler. Once this IR is produced, we use a novel algorithm to generate a set of graph conditions with respect to the bounds chosen by users. These graph conditions can then efficiently be verified by an off-the-shelf SMT solver[3]. Finally, the trace generator produces a *trace* or *counter-example* for the specification (depending on the mode it is running as).

We now use a bouncing example to illustrate some of the features of Cyclone. Figure 3 plots a transition graph for bouncing ball model. Listing 1 shows the corresponding Cyclone specification. The two states [4](`Fall` and `Bounce`) in Figure 3 are directly mapped to the nodes and each transition

[2]Cyclone can be accessed through our online editor: https://cyclone.cs.nuim.ie, and a short tutorial is also included.
[3]Currently, Cyclone uses Z3 [27] as its default solver. Work is in progress to also support the CVC5 solver.
[4]The computation could terminate at any of the two nodes.

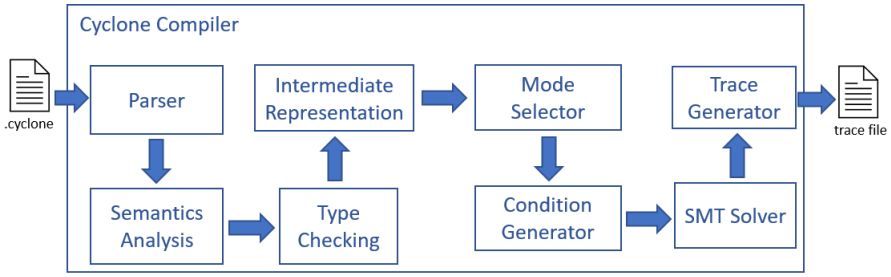

Figure 2: Architecture of Cyclone.

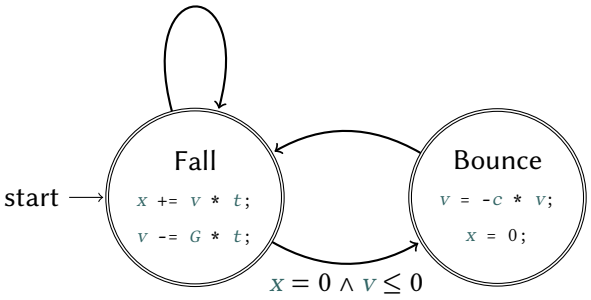

Figure 3: A transition graph for a bouncing ball.

is mapped to an edge. The calculation of the position of the ball is shown in each node. For example, the *Bounce* node describes that when the ball hits ground (x=0), it loses some energy and change its direction to bounce back. For this simple model, we use Cyclone to verify whether the position of the ball always stays positive. In other words, the ball is either in the air or on the ground. This property is described using a graph invariant [5] (Line 22 in Listing 1) in the specification. To verify this specification, Cyclone uses a new algorithm that effectively reduces the transition graph along with its invariants into a set of formulas that can be solved by an SMT solver [27].

To comprehensively evaluate Cyclone's robustness, we have designed and collected over 200 sample problems for testing purposes. We have found and fixed more than 120 issues in the past year. We have also introduced Cyclone into the curriculum at Maynooth University and compared to other verification tools. More than 200 students have learned and used Cyclone in multiple modules such as Software Verification and Theory of Computation. We collect the feedback from students, more than 85% of them think Cyclone is easier to learn. Some of the highlighted evaluation results can be found in [26].

Based on our preliminary evaluation, this approach provides us with three main advantages: (1) The behaviors of a complex system can be visualized as a transition graph, making it easier to share and communicate with other users. (2) Properties can be specified using graph invariants, and Cyclone is capable of generating test cases for the defined graph (if needed) or verifying it by discovering counter-example(s). (3) The learning curve is not as steep as other verification tools. In fact, we have successfully used Cyclone to verify Event-B hybrid models [28] and now aim to extend it to ARS.

Listing 1: A Cyclone specification modelling a transition graph of a bouncing ball hybrid system depicted in Figure 3. The variable block (Line 2–6) contains all necessary variables for computing the ball's position x, which can never be below 0 (Line 22). The node and edge blocks (Line 8–20) model different states of the ball. The goal block (Line 24–26) instructs Cyclone to explore a counter-example within a set of given bounds. Note that t is the discretisation unit for time.

```
1  graph Bouncing_Ball {
2      real x where x >= 0;          // position of the ball
3      real v;                       // velocity of the ball
```

[5] When a specification contains at least one graph invariant, Cyclone switches to the counter-example discovery mode.

```
    real t where t >= 0;           // time sequence
    const real G = 9.81;           // gravitational force
    real c where c >= 0 && c <= 1; // constant (energy loss)

    normal start final node Fall {
        x += v * t;
        v -= G * t;
    }

    normal final node Bounce {
        v = -c * v;
        x = 0;
    }

    edge { Fall -> Fall }
    edge { Fall -> Bounce, where x == 0 && v <= 0; } // the ball starts to bounce back when it reaches ground and its
         velocity becomes negative.
    edge { Bounce -> Fall }

    invariant inv { x >= 0; } // position of the ball is never < 0

    goal { // defines different bounds for Cyclone to explore.
        check for 2,3,4,5
    }
}
```

4. Work in Progress

Currently, we are investigating the use of Cyclone to verify a scheduling algorithm in an elevator system as a case study. Our aim is to use Cyclone to help engineers uncover design flaws at a very early stage. This involves modeling a scheduling procedure using a transition graph and defining its invariants. In the future, we plan to (1) complete our survey on ARS verification, (2) identify other key aspects of an ARS that need to be verified, and (3) design a framework that can transform an ARS's specifications along with its properties into a graph that can be directly mapped to a Cyclone specification.

References

[1] M. Webster, L. A. Dennis, C. Dixon, M. Fisher, Assurance of autonomous robots: Verifying performance with model checking and simulation, in: 2022 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2022, pp. 11505–11511.

[2] M. Farrell, M. Webster, L. A. Dennis, M. Fisher, S. M. Veres, Robust model predictive control with formal methods for autonomous systems, Artificial Intelligence 278 (2020) 103179.

[3] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems: A survey, in: ACM Computing Surveys (CSUR), volume 52, ACM, 2019, pp. 1–41.

[4] Y. Yu, P. Manolios, L. Lamport, Model checking TLA+ specifications, in: Advanced Research Working Conference on Correct Hardware Design and Verification Methods, Springer, 1999, pp. 54–66.

[5] V. Fitera, S. Popescu, A.-I. Ene, A survey on model checking techniques: From design to industry applications, Software: Practice and Experience 52 (2022) 1865–1892.

[6] L. Bozzelli, S. La Torre, M. Napoli, Advanced techniques in model checking for pushdown systems, in: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), IEEE, 2021, pp. 1–10.

[7] F. Belardinelli, A. Lomuscio, Verification of multi-agent systems via predicate abstraction: An automated approach, Journal of Artificial Intelligence Research 68 (2020) 725–764.

[8] E. A. Emerson, K. S. Namjoshi, Advances in model checking: Theory and practice, in: Proceedings

of the 2020 ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, ACM, 2020, pp. 135–138.
[9] J.-P. Katoen, The probabilistic model checking landscape, Proceedings of the ACM on Programming Languages 3 (2019) 1–31.
[10] A. Platzer, Logical foundations of cyber-physical systems, volume 662, Springer, 2018.
[11] D. Marmsoler, B. Rumpe, Formal specification and verification of model transformations with Alloy: an evaluation of usability and scalability, Software and Systems Modeling 18 (2019) 2205–2226.
[12] C. C. Plat, G. Rodrigues, Formal specification and verification of smart contracts with Scilla, Formal Aspects of Computing 32 (2020) 169–191.
[13] T. Amorim, E. Cavalcante, A formal specification approach for verifying autonomous systems using TLA+, in: 2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2021, pp. 297–303.
[14] V. Auer, G. Weiss, Contracts and runtime verification for cross-organization collaboration, Information Systems 88 (2020) 101455.
[15] S. Schneider, V. Sivaraman, H. Treharne, J. Woodage, A comprehensive contract verification framework for smart contracts, in: International Conference on Software Engineering and Formal Methods, Springer, 2019, pp. 405–420.
[16] K. R. M. Leino, Dafny: An automatic program verifier for functional correctness, in: International conference on logic for programming artificial intelligence and reasoning, Springer, 2010, pp. 348–370.
[17] H. H. Le, Y. Falcone, A. Rollet, A survey on runtime verification of distributed systems, ACM Computing Surveys (CSUR) 54 (2021) 1–33.
[18] V. Genovese, F. Biondi, Y. Falcone, Runtime verification of hyperproperties for deterministic and non-deterministic systems, in: International Conference on Runtime Verification, Springer, 2021, pp. 215–235.
[19] D. Attard, N. Buhagiar, Secure smart contract verification through semantic decomposition of logic vulnerabilities, Journal of Information Security and Applications 66 (2022) 103188.
[20] J.-C. Laprie, Dependable computing and fault tolerance: Concepts and terminology, in: Fault-Tolerant Computing, IEEE Computer Society Press, 1995, pp. 2–11.
[21] T. M. Powers, Prospects for a kantian machine, IEEE Intelligent Systems 21 (2006) 46–51.
[22] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, M. Fisher, A summary of formal specification and verification of autonomous robotic systems, in: International Conference on Autonomous Systems and Formal Methods, IEEE, 2019, pp. 1–10.
[23] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems, in: International Conference on Logic and Computation in Autonomous Systems, IEEE, 2018, pp. 50–65.
[24] F. Ingrand, Verification of autonomous robots: A roboticist's bottom-up approach, in: International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 120–135.
[25] M. Foughali, A. Zuepke, Formal verification of real-time autonomous robots: An interdisciplinary approach, in: International Conference on Formal Verification and Autonomous Systems, Springer, 2022, pp. 200–215.
[26] H. Wu, F. Thomas, M. Dominique, Cyclone: A new tool for verifying/testing graph-based structures, in: 18th International Conference on Tests and Proofs, Springer, 2024.
[27] L. De Moura, N. Bjørner, Z3: An efficient SMT solver, in: International conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.
[28] H. Wu, Z. Cheng, Verifying Event-B hybrid models using Cyclone, in: International Conference on Rigorous State-Based Methods, Springer, 2023, pp. 179–184.