
# Proceedings of the PhD Symposium at the 19th International Conference on Integrated Formal Methods (iFM) 2024

Manchester University, Manchester, UK

12 November 2024

# Preface

The PhD Symposium at the International Conference on Integrated Formal Methods (iFM) 2024 aims at providing PhD students an opportunity to present and discuss their research in the fields of theory, implementation, integration or application of formal methods. It is targetted towards PhD students and young researchers at an early career stage (up to 2 years after PhD completion). The goal of the symposium is to provide a possibility to the participants to present their research projects. Moreover: The PhD symposium offers the participants an excellent opportunity to introduce their work to fellow researchers in an international setting, and to get feedback from senior researchers in the field. The doctoral symposium provides an environment to exchange knowledge and experiences with fellow PhD-students in a related topic – both regarding research topics, but regarding being a PhD candidate and working towards an PhD, and about future career plans.

It is the 19th time the community meets at iFM. Like in the previous iterations of co-located PhD symposia at iFM, we were able to engage outstanding figures from the scientific community as presenters for the invited talks, who were able to provide participants with valuable input based on their scientific and academic expertise and experience. *Prof. Dr. Paula Herber* from the University of Münster illuminated the opportunities and challenges of an academic career in her lecture "How to Become a Professor," while *Dr. Renate A. Schmidt* from the University of Manchester gave a more scientifically oriented overview presentation titled "Research in Knowledge Base Extraction: Tools, Applications and Lessons Learnt."

We received a total of 10 submissions, out of which the programme committee selected 8 for presentation and publication. This year, we supported three categories of papers that could be submitted: 1) *Thesis Proposal Abstracts* summarizing research questions and outlining a research project. They are ideal for early-stage PhD students to get feedback on their research project during the initial planing and orientation phase. 2) *Result Reports* summarizing preliminary results of early-stage research. Papers on unexpected results or ineffective methods were particularly welcome. 3) *Master Summaries* summarizing the research question, method, and results of an impactful Master's thesis together with a discussion about possible next research steps. They were intended for new and future PhD students to communicate their thesis results together an experienced supervisor. For all formats, supervisors and colleagues were allowed to act as co-authors.

Mădălina Erașcu and Mattias Ulbrich,
co-chairs

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

# Organisation

of the PhD Symposium at iFM 2024 in Manchester, UK on 12 November 2024.

## Programme Committee Co-Chairs

| | |
|---|---|
| Mădălina Eraşcu | West University of Timisoara, Romania |
| Mattias Ulbrich | Karlsruhe Institute of Technology, Germany |

## Programmme Committee

| | |
|---|---|
| Erika Abraham | RWTH Aachen, Germany |
| Ștefan Ciobâcă | UAIC Iași, Romania |
| Grigory Fedyukovich | Floria State University, USA |
| Asmae Heydari Tabar | Karlsruhe Institute of Technology, Germany |
| Eduard Kamburjan | University of Oslo, Norway |
| Benjamin Kaminski | Saarland University, Germany, and University College London, UK |
| Gergely Kovasznai | Eszterházy Károly University, Eger, Hungary |
| Ondrej Lengal | Brno University of Technology, Czech Republic |
| Luigia Petre | Åbo Akademi University, Norway |
| Philipp Rümmer | University of Regensburg, Germany |
| Nestan Tsiskaridze | Stanford University, USA |

# Table of Contents



Due to technical reasons, hyperlinks within the individual papers do not work in this overview file.

# Combining Quantitative and Qualitative Analysis for Safe and Resilient Intelligent Hybrid Systems

Pauline Blohm[1], Paula Herber[1] and Anne Remke[1]

[1]University of Münster, Münster, Germany

**Abstract**

Model-driven development frameworks such as MATLAB Simulink are widely used in industrial design processes to conquer the increasing complexity of embedded control systems such as self-driving cars or critical infrastructures. As these systems are often safety-critical, formal methods to ensure safety, performance and resilience are highly desirable, in particular also in the presence of dynamic and uncertain environments. Formal verification has the potential to a) ensure that embedded systems function correctly for all possible system parameters and input scenarios, and b) provide statistical guarantees in the presence of uncertainty and probabilistic behavior. However, the application of existing formal verification and stochastic analysis techniques to embedded control systems is a major challenge, in particular if they are hybrid, i.e. combine discrete and continuous behavior, and include learning components to adapt to dynamic environments. To tackle this challenge, we aim at providing a quantitative analysis for intelligent Simulink models via a transformation to Stochastic Hybrid Automata (SHA) that gives us access to established analysis techniques for stochastic systems, such as reachability analysis or (statistical) model checking. To incorporate dynamic adaptations via learning, we investigate techniques to integrate domain-specific abstractions of the learning components into the SHA model. To ensure resilience of learning hybrid systems, we aim at combining the strengths of qualitative and quantitative analyses.

**Keywords**

Hybrid Systems, Formal Verification, Stochastic Failures, Learning, Simulink, Hybrid Automata

## 1. Problem

The demands on the functionality and flexibility of embedded control systems are steadily increasing. At the same time, they are more and more used in critical infrastructures, for example, controlling the supply of energy or water, and in safety-critical systems such as self-driving cars and other autonomous vehicles. With that, we increasingly use embedded control systems not only for our convenience or for profit, but also trust our lives and personal well-being to these systems. At the same time, learning components are nowadays often used to cope with dynamic environments. This makes it crucial to ensure the safety, performance, and resilience of these systems under all circumstances. Qualitative analysis techniques such as deductive verification can provide safety guarantees for hybrid systems, however, they typically only consider the worst case scenario. In contrast, quantitative analysis techniques like analytical reachability analysis or statistical model checking (SMC) can provide statistical guarantees for safety or performance properties even in the presence of uncertainty, however, they might not provide guarantees for all possible scenarios.

The integration of learning components and uncertainties further complicates formal verification of hybrid systems. Qualitative analysis techniques often rely on abstractions, such as contracts, to handle learning components or uncertainties. While these abstractions are necessary to provide safety guarantees, they usually abstract from all quantitative information, yielding imprecise and overly pessimistic results. In contrast, quantitative techniques exploit statistical information like the distribution of events or failure and repair times. However, they suffer from state-space explosion, in particular if learning components have to be verified to ensure safety under all circumstances or with high accuracy. Furthermore, quantitative analyses techniques typically do not provide us with

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

pauline.blohm@uni-muenster.de (P. Blohm); paula.herber@uni-muenster.de (P. Herber); anne.remke@uni-muenster.de (A. Remke)

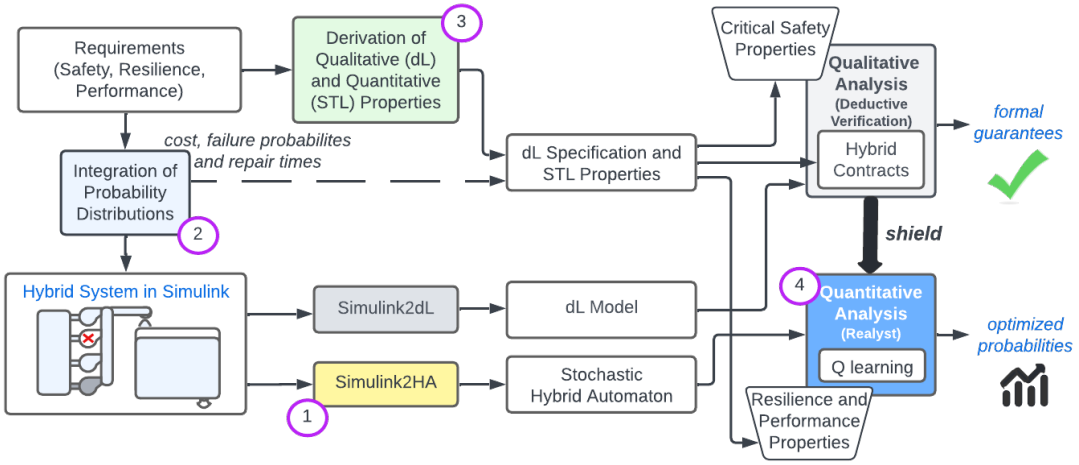



**Figure 1:** Combining the Strengths of Quantitative and Qualitative Analysis.

techniques for modular reasoning.

## 2. Proposed Solution

To ensure the safety and resilience of hybrid systems even in the presence of learning and in uncertain environments, we aim at combining the strengths of qualitative analysis with the strength of quantitative analysis. We focus on hybrid systems modelled in Simulink as it is widely adopted for embedded control systems that combine discrete and continuous behavior. Previous work of one of the co-authors has presented an approach for a qualitative analysis of Simulink models via a transformation to Differential Dynamic Logic (dL) [1, 2] which is implemented in the tool *Simulink2dL*. The resulting *dL Model* can then be analyzed using *Deductive Verification* to obtain formal guarantees that a system satisfies a given *Safety Property*. Our aim is to complement this with efficient and scalable quantitative analysis [3, 4, 5] and also to combine these techniques to provide an approach for comprehensive safety, resilience and performance analysis for intelligent hybrid systems. The concept of the thesis is shown in Fig. 1 and consists of four main parts:

1. We plan to provide an automated transformation *Simulink2SHA* from Simulink to Stochastic Hybrid Automata. With this transformation, we define a formal semantics for Simulink, and it gives us access to established *Quantitative Analysis* techniques, such as reachability analysis or (statistical) model checking.
2. We aim at investigating how to introduce stochastic components via *Probability Distributions* into the Simulink model to model uncertainties like component failure or sensor noise. With that, we can also investigate resilience and performance of a model under verification.
3. We aim at investigating how *Qualitative* and *Quantitative Properties* for the dL and SHA models can be derived from safety, resilience and performance requirements.
4. We plan to provide a technique to combine *Qualitative Analysis* results (e.g. from deductive verification) with a *Quantitative Analysis* for (potentially learning) Simulink models. In particular, we aim at investigating two different approaches: a) the integration of shielded SMC-based learning, which has been proposed by one of the co-authors in previous work [6], and b) the use of contracts or other domain-specific abstractions to safely integrate learning components, as has been proposed by two of the co-authors in [2, 7].

As a first step, we have presented an approach for a manual transformation from Simulink to SHA, which has been accepted at iFM 2024 [8]. Open research problems we plan to address in this thesis are how to model uncertainties such that the resulting models are still analyzable, how to capture qualitative and quantitative properties of intelligent systems appropriately and also how to combine

qualitative with quantitative analysis techniques for SHA with learning components. One key challenge is that, while modular reasoning would be highly desirable to handle complex systems, there exists no established concept to include quantitative and statistical information in contracts.

## 3. Related Work

There have been quite some efforts to enable the formal verification of systems that are modeled in Simulink [9, 10, 11, 12, 13, 14, 15]. Yet, all of these approaches, including the Simulink Design Verifier [16], are limited to discrete subsets of Simulink. Formal verification methods that support hybrid systems modeled in Simulink are, e.g., proposed in [1, 17, 18, 19, 20, 21]. However, they either focus on techniques for a special class of systems and do not provide general transformation rules for a broader set of blocks or focus on the qualitative analysis of safety properties and they neither take stochastic components nor learning into consideration.

There also has been a number of works on statistical model checking (SMC) for Simulink, for example [22, 23, 24]. Still they do not provide a stochastic model with formal semantics and thus cannot make use of more advanced quantitative analysis techniques. In [25], the authors propose a transformation from Simulink into stochastic timed automata (STA) and perform SMC with UPPAAL SMC on the resulting network of STA. However, they do not consider stochastic blocks and transform a given Simulink model into a deterministic STA model where all probabilities are one.

There also exists a broad variety of approaches to formally ensure safety of learning components using shielding or runtime monitoring [26, 27, 28]. These approaches do not directly support Simulink though, and they do not consider formal analysis techniques that take stochastic failures and learning into account.

Uppaal Stratego [29] uses priced timed automata to model stochastic behavior, and provides tooling for statistical model checking [30], timed games [31] and learning-based strategy synthesis [32]. While Uppaal Stratego comes with a graphical interface and is designed for usability, the underlying formalisms are less expressive than stochastic hybrid automata, in particular w.r.t. continuous system behavior governed by differential equations and controlled by continuous and stochastic variables.

Finally, there has been some work on combining rigorous formal and statistical methods. In [33], the authors incorporate statistical hypothesis testing to compute promising configurations of program verifiers automatically. However, they do not support hybrid systems, and they do not consider both safety and performance properties. In [34], the authors present a formal framework for an integrated qualitative and quantitative model-based safety analysis. However, they do not support hybrid systems and do not consider deductive verification methods.

## 4. Progress and Current State

The first step towards safe and resilient hybrid system is enabling a quantitative analysis for (stochastic) Simulink models. As Simulink does not offer elaborate quantitative analysis, such as reachability analysis or statistical model checking, a transformation into a formal model is desired. To tackle this problem, we are currently working on a modular approach to transform Simulink models into SHA. In [8], we present an approach that enables us to transform a subset of Simulink models to SHA and analyze the SHA using the tool RealySt [3] to obtain reachability probabilities for critical safety properties. This is an important first step towards ensuring safety and resilience of hybrid systems in the presence of uncertainties. However, it still has some limitations, e.g. we only provide transformation rules for a subset of Simulink blocks and the parallel composition has to be performed manually. To tackle these limitations, we are currently working on a tool to automatically transform a given Simulink model to an SHA using the transformation rules provided in [8]. Additionally, we plan to define transformation rules for a larger subset of Simulink blocks and provide better support for the integration of stochasticity into Simulink models, e.g. by providing parameterized subsystems that model specific stochastic behaviour.

As next steps, we plan to address the research challenges defined above, namely the integration of stochastic and learning components in Simulink, the derivation of qualitative and quantitative properties that are important for safety, resilience and performance of intelligent Simulink models in uncertain and dynamic environments, and the development of combined quantitative and quantitative analysis techniques that enable us to formally analyze these properties.

## References


[1] T. Liebrenz, P. Herber, S. Glesner, Deductive verification of hybrid control systems modeled in Simulink with KeYmaera X, in: Int. Conference on Formal Engineering Methods, volume 11232 of *LNCS*, Springer, 2018, pp. 89–105. doi:10.1007/978-3-030-02450-5_6.

[2] J. Adelt, T. Liebrenz, P. Herber, Formal Verification of Intelligent Hybrid Systems that are modeled with Simulink and the Reinforcement Learning Toolbox, in: Formal Methods, volume 13047 of *LNCS*, Springer, 2021, pp. 349–366. doi:10.1007/978-3-030-90870-6_19.

[3] J. Delicaris, J. Stübbe, S. Schupp, A. Remke, Realyst: A C++ tool for optimizing reachability probabilities in stochastic hybrid systems, in: 16th EAI Int. Conference on Performance Evaluation Methodologies and Tools, volume 539 of *LNICS*, Springer, 2023, pp. 170–182. doi:10.1007/978-3-031-48885-6_11.

[4] C. Da Silva, S. Schupp, A. Remke, Optimizing reachability probabilities for a restricted class of stochastic hybrid automata via flowpipe construction, ACM Trans. Model. Comput. Simul. 33 (2023). doi:10.1145/3607197.

[5] M. Niehage, A. Remke, The best of both worlds: Analytically-guided simulation of hpngs for optimal reachability, in: Performance Evaluation Methodologies and Tools, Springer Nature, 2024, pp. 61–81. doi:10.1007/978-3-031-48885-6_5.

[6] M. Niehage, A. Hartmanns, A. Remke, Learning optimal decisions for stochastic hybrid systems, in: ACM-IEEE Int. Conference on Formal Methods and Models for System Design, ACM, 2021, pp. 44–55. doi:10.1145/3487212.3487339.

[7] P. Blohm, J. Adelt, P. Herber, Safe Integration of Learning in SystemC using Timed Contracts and Model Checking, in: R. von Hanxleden, S. A. Edwards, J. Brandt, Q. Zhu (Eds.), 21st ACM-IEEE International Symposium on Formal Methods and Models for System Design, ACM / IEEE, 2023, pp. 12–22. doi:10.1145/3610579.3611078.

[8] P. Blohm, P. Herber, A. Remke, Towards Quantitative Analysis of Simulink Models using Stochastic Hybrid Automata, in: International Conference on Integrated Formal Methods, accepted for publication, 2024.

[9] D. Araiza-Illan, K. Eder, A. Richards, Formal verification of control systems' properties with theorem proving, in: UKACC Int. Conference on Control, IEEE, 2014, pp. 244–249. doi:10.1109/CONTROL.2014.6915147.

[10] 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. doi:10.1007/978-3-540-78800-3_24.

[11] J.-C. Filliâtre, A. Paskevich, Why3 — where programs meet provers, in: European Symposium on Programming, Springer, 2013, pp. 125–128. doi:10.1007/978-3-642-37036-6_8.

[12] P. Herber, R. Reicherdt, P. Bittner, Bit-precise formal verification of discrete-time MATLAB/Simulink models using SMT solving, in: Int. Conference on Embedded Software, IEEE, 2013, pp. 1–10. doi:10.1109/EMSOFT.2013.6658586.

[13] S. K. Lahiri, S. A. Seshia, The UCLID decision procedure, in: Int. Conference on Computer Aided Verification, Springer, 2004, pp. 475–478. doi:10.1007/978-3-540-27813-9_40.

[14] R. Reicherdt, S. Glesner, Formal verification of discrete-time MATLAB/Simulink models using Boogie, in: Int. Conference on Software Engineering and Formal Methods, volume 8702 of *LNCS*, Springer, 2014, pp. 190–204. doi:10.1007/978-3-319-10431-7_14.

[15] M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, K. R. M. Leino, Boogie: A modular reusable

verifier for object-oriented programs, in: Int. Symposium on Formal Methods for Components and Objects, Springer, 2005, pp. 364–387. doi:10.1007/11804192_17.

[16] The MathWorks, Simulink Design Verifier, https://de.mathworks.com/products/simulink-design-verifier.html, 2024.

[17] A. Chutinan, B. H. Krogh, Computational techniques for hybrid system verification, in: IEEE Trans. on Automatic Control, volume 48(1), IEEE, 2003, pp. 64–75. doi:10.1109/TAC.2002.806655.

[18] S. Minopoli, G. Frehse, SL2SX translator: from Simulink to SpaceEx models, in: Int. Conf. on Hybrid Systems: Computation and Control, ACM, 2016, pp. 93–98. doi:10.1145/2883817.2883826.

[19] L. Zou, N. Zhan, S. Wang, M. Fränzle, Formal Verification of Simulink/Stateflow Diagrams, in: Int. Symposium on Automated Technology for Verification and Analysis (ATVA), volume 9364 of *LNCS*, Springer, 2015, pp. 464–481. doi:10.1007/978-3-319-47016-0.

[20] M. Chen, X. Han, T. Tang, S. Wang, M. Yang, N. Zhan, H. Zhao, L. Zou, MARS: A toolchain for modelling, analysis and verification of hybrid systems, in: Provably Correct Systems, Springer, 2017, pp. 39–58. doi:10.1007/978-3-319-48628-4_3.

[21] T. Liebrenz, P. Herber, S. Glesner, A service-oriented approach for decomposing and verifying hybrid system models, in: Int. Conference on Formal Aspects of Component Software, volume 12018 of *LNCS*, Springer, 2019, pp. 127–146. doi:10.1007/978-3-030-40914-2_7.

[22] P. Zuliani, A. Platzer, E. M. Clarke, Bayesian statistical model checking with application to stateflow/simulink verification, Formal Methods in System Design 43 (2013) 338–367. doi:10.1007/s10703-013-0195-3.

[23] A. Legay, L.-M. Traonouez, Statistical model checking of simulink models with Plasma Lab, in: Formal Techniques for Safety-Critical Systems: 4th International Workshop, Springer, 2016, pp. 259–264. doi:10.1007/978-3-319-29510-7_15.

[24] B. Boyer, K. Corre, A. Legay, S. Sedwards, PLASMA-lab: A flexible, distributable statistical model checking library, in: Quantitative Evaluation of Systems: 10th International Conference, Springer, 2013, pp. 160–164. doi:10.1007/978-3-642-40196-1_12.

[25] P. Filipovikj, N. Mahmud, R. Marinescu, C. Seceleanu, O. Ljungkrantz, H. Lönn, Simulink to uppaal statistical model checker: Analyzing automotive industrial systems, in: International Symposium on Formal Methods, Springer, 2016, pp. 748–756. doi:10.1007/978-3-319-48989-6_46.

[26] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, U. Topcu, Safe Reinforcement Learning via Shielding, Proceedings of the AAAI Conference on Artificial Intelligence 32 (2018). doi:10.5555/3504035.3504361.

[27] B. Könighofer, F. Lorber, N. Jansen, R. Bloem, Shield synthesis for reinforcement learning, in: International Symposium on Leveraging Applications of Formal Methods, Springer, 2020, pp. 290–306. doi:10.1007/978-3-030-61362-4_16.

[28] D. Phan, J. Yang, M. Clark, R. Grosu, J. Schierman, S. Smolka, S. Stoller, A Component-Based Simplex Architecture for High-Assurance Cyber-Physical Systems, in: 2017 17th International Conference on Application of Concurrency to System Design (ACSD), IEEE, 2017, pp. 49–58. doi:10.1109/ACSD.2017.23.

[29] A. David, P. G. Jensen, K. G. Larsen, M. Mikučionis, J. H. Taankvist, Uppaal stratego, in: Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2015, pp. 206–211. doi:10.1007/978-3-662-46681-0_16.

[30] A. David, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, Uppaal smc tutorial, International Journal on Software Tools for Technology Transfer 17 (2015) 397–415.

[31] F. Cassez, A. David, E. Fleury, K. G. Larsen, D. Lime, Efficient on-the-fly algorithms for the analysis of timed games, in: M. Abadi, L. de Alfaro (Eds.), Concurrency Theory, Springer, Berlin, Heidelberg, 2005, pp. 66–80. doi:10.1007/11539452_9.

[32] P. Ashok, J. Křetínskỳ, K. G. Larsen, A. Le Coënt, J. H. Taankvist, M. Weininger, SOS: Safe, optimal and small strategies for hybrid markov decision processes, in: International Conference on Quantitative Evaluation of Systems, Springer, 2019, pp. 147–164. doi:10.1007/978-3-030-30281-8_9.

[33] A. Knüppel, T. Thüm, I. Schaefer, GUIDO: Automated Guidance for the Configuration of Deductive Program Verifiers, in: IEEE/ACM Int. Conference on Formal Methods in Software Engineering

(FormaliSE), IEEE, 2021, pp. 124–129. doi:10.1109/FormaliSE52586.2021.00018.

[34] M. Gudemann, F. Ortmeier, A framework for qualitative and quantitative formal model-based safety analysis, in: IEEE Int. Symposium on High Assurance Systems Engineering, IEEE, 2010, pp. 132–141. doi:10.1109/HASE.2010.24.

# Hybrid Games with Triggers*

Qais Hamarneh

*Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany*

**Abstract**

Hybrid games are a highly expressive way to model the interaction between cyber-physical systems. This high expressivity comes at the price of decidability. This work proposes an extension to hybrid games that adds the conditions prompting agents to move. We call this extension hybrid games with triggers (HGT). We show how this extension makes it possible to translate a hybrid game into a discrete game with countable state space.

**Keywords**

Hybrid Games, Hybrid systems, Discrete Games, Verification

## 1. Introduction

Modelling the interaction between multiple cyber-physical systems (CPS) is a critical problem in computer science, particularly in formal methods. Many approaches were presented to model and reason about this interaction, such as dynamic differential logic [1], and its extension to differential game logic [2], algebraic [3] and coalgebraic [4] approaches among many others.

This work is based on modelling an interaction between multiple CPSs as a hybrid game [5], a multi-agent extension of hybrid automata [6]. Hybrid games allow for discrete and continuous system evolution. Typically, the discrete evolution represents the agents' control, whereas the continuous evolution represents the motion dynamics. While hybrid games are very expressive, this expressivity comes at the price of decidability. In [7], Heinziger et al. show that the reachability in even very restrictive fragments of hybrid automata, like a stopwatch automaton, is undecidable.

There have been multiple attempts to discretize hybrid systems [8] by restricting either the discrete dynamics like in rectangular hybrid games [5] or the continuous dynamics like in o-minimal hybrid games [9]. This work presents a novel extension of hybrid games that allows discretization without restricting system dynamics. It does this by augmenting the hybrid game definition with information about the agents' rationale. This rationale is expressed as quantifier-free formulas of real arithmetic called *triggers*. A trigger is a condition that the agent must act once satisfied. We call this extension *hybrid games with triggers* (**HGT**).

HGTs are inspired by de Alfaro et al.'s timed games [10]. In this version of timed games, each agent declares a time delay, after which they would take an action unless some other agent played first. In essence, our extension replaces the time delay with arithmetic formulas, similar to the guards and invariants of hybrid automata [6], and the timed game with a hybrid game.

Using a logical formula instead of a time delay brings multiple advantages. (1) Formulas reduce the need for perfect information by covering cases without calculating the time required to reach them. For instance, a car safety trigger could look like this:

$$\mathit{free\text{-}ahead}(\mathit{position}) \leq \mathit{braking\text{-}distance}(\mathit{speed}). \tag{1}$$

(2) Unlike time, a small set of formulas like 1 could be sufficient to model a complete system. (3) As we show in this paper, given a countable language of real arithmetic [11], triggers reduce the hybrid game into a discrete game with countable state space.

*PhD Symposium of the 19th International Conference on Integrated Formal Methods (iFM)*
*at the University of Manchester, UK, 12 November 2024.*
qais.hamarneh@kit.edu (Q. Hamarneh)
https://mase.kastel.kit.edu/team_qais_hamarneh.php (Q. Hamarneh)
0009-0009-9718-1664 (Q. Hamarneh)

In the next Section 1.1, we briefly overview the related work. Section 2 defines the hybrid games this work is based on. The main contribution of this work is presented in sections 3 and 4. In Section 3, we introduce the syntax and semantics of hybrid games with triggers, and we informally define the algorithm to create a discrete game based on an HGT in Section 4. We conclude in Section 5 with a summary and a look at future work.

### 1.1. Related Work

This work can be seen as a hybrid extension of de Alfaro et al.'s timed games [10]. Multiple points were taken directly from the timed games, like the winning conditions and what happens when multiple triggers are satisfied simultaneously. However, the discretization algorithm is entirely different. The discretization in the timed games relies on the existence of a finite bisimulation of timed automata as a region automata. Hybrid automata do not always offer a finite bisimulation [12].

As already discussed, other ways to discretize hybrid games exist [5][9], but where these methods restrict the game dynamics to allow discretization, our approach rely on adding more information to the game instead of restricting it.

Tight durational concurrent game structures (TDCGS) [13] are intuitively similar to hybrid game with triggers. Transitions in TDCGS carry an integer time delay. This time, however, is treated as a cost and does not reflect the evolution of the continuous dynamics.

## 2. Preliminaries

### Hybrid Game

This definition of a hybrid game is adopted from [5]. The hybrid game is defined over a finite set of real-valued variables $X$. The set of all valuations $v : X \to \mathbb{R}$ is called $Val(X)$. We extend the notation $v$ to arithmetic terms over $X$. $Form_X$ is the set of all real arithmetic quantifier-free formulas over $X$.

Given a valuation $v \in Val(X)$ and an arithmetic term $\theta$, we write $v[\theta/x]$ for the valuation where all variables have the same value as in $v$, except $v[\theta/x](x) = v(\theta)$. This definition is extended to ordered sets of assignments, where the assignments are executed in order. Given a set of differential equations $F$, we write $v[F,t]$ for the valuation updated according to the equations in $flow$ after some time $t \in \mathbb{R}_{\geq 0}$ has passed. A hybrid game $\mathscr{G}$ is a tuple:

$$\mathscr{G} = (Loc,\ l_0,\ X,\ v_0,\ flow,\ inv,\ Agt,\ Act,\ E)$$

| | |
|---|---|
| $Loc$ | a finite nonempty set of locations. |
| $l_0 \in Loc$ | the initial location. |
| $X$ | a finite nonempty set of real-valued variables with typical elements $x_0, x_1$. |
| $v_0 : X \to \mathbb{R}$ | the initial valuation. |
| $flow$ | a continuous transition relation that assigns each location $l \in Loc$ a set of differential equations $flow(l) = \{\ \dot{x_i} = \theta_i \mid x_i \in X\}$. |
| $inv : Loc \to Form_X$ | associate each location with an invariant. |
| $Agt =$ | $\{\ 1, 2, \ldots, k\ \}$ a finite nonempty set of agents. |
| $Act =$ | $Act_1 \uplus Act_2 \uplus \ldots \uplus Act_k$ a disjoint union of finite nonempty sets of agents' actions. The typical actions of agent $i \in Agt$ are $a_i$, $b_i$. |
| $E$ | a finite nonempty set of edges representing discrete transition relation with typical elements $(l, \varphi, a_i, A, l')$ such that: • $l, l' \in Loc$, • $\varphi \in Form_X$ called a *guard*, • $a_i \in Act_i$ an action for some $i \in Agt$, and • $A$ is a finite (possibly empty) ordered set of assignments called a *jump*. |

We use the functions $begin(e)$, $guard(e)$, $act(e)$, $jump(e)$ and $end(e)$ to reference the components of an edge $e \in E$.

A valuation $v$ *enables* the edge $e = (l, \varphi, a, A, l') \in E$ (we say the action $act(e)$ is enabled) if:

$$\bullet\ v \vDash inv(l),\quad \bullet\ v \vDash \varphi,\ \text{and}\ \bullet\ v[A] \vDash inv(l')$$

In a hybrid game, a *configuration* is a pair $\langle l, v \rangle$ representing the game's location $l \in Loc$ and valuation $v \in Val(X)$. $\langle l_0, v_0 \rangle$ is the initial configuration. Two types of transitions are possible: A time transition $\langle l, v \rangle \xrightarrow{t} \langle l, v[flow(l), t] \rangle$ for $t \in R_{\geq 0}$ is *legal* if for all $t' \in [0, t]$, $v[flow(l), t'] \vDash inv(l)$. An edge transition $\langle l, v \rangle \xrightarrow{e} \langle end(e), v[jump(e)] \rangle$ for an edge $e \in E$ with $begin(e) = l$ is a legal transition if $v$ enables $e$. The agent $i$ that takes the action $act(e) \in Act_i$ is called *to blame* for the edge transition. The other agents are called *blameless*. A **play** is an infinite sequence of configurations $(\langle l_0, v_0 \rangle, \langle l_1, v_1 \rangle, \langle l_2, v_2 \rangle, \ldots)$ which starts at the initial configuration and for each $i \in \mathbb{N}_0$, there exists a legal transition $\langle l_i, v_i \rangle \to \langle l_{i+1}, v_{i+1} \rangle$.

**Remark 1.** *In this paper, we assume all differential equations to be solvable. Even with solvable differential equations, the question of whether there exists a play that reaches a certain configuration is undecidable in a hybrid game [7].*

#### 2.0.1. Example

Figure 1 shows an orange and a green robot moving on tracks in a warehouse. The warehouse contains 6 tracks, 2 horizontal $H = \{ h_1, h_2 \}$ and 4 vertical $V = \{ v_1, v_2, v_3, v_4 \}$. The robots continuously pick items from storage units (small circles) and take them to the processing units (small squares). The robots must also avoid collisions with each other. To model this system as a hybrid game, we define the locations as the current tracks of the two robots $Loc = (H \cup V) \times (H \cup V)$. The current location is $(h_1, v_3)$. The invariants define the end points of each track. The flow in each location describes the motion of each robot based on its current speed. Intersection points are edges that allow the robots to change tracks. Self-loop edges change a robot's speed without changing its track.

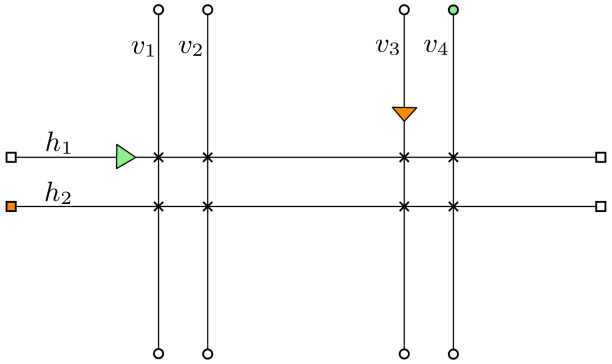


Figure 1: Robots on tracks.

## 3. Hybrid Game with Triggers (HGT)

In this section, we introduce an extension to the definition of hybrid games. We call this extension *hybrid games with triggers* or (HGT) for short. A trigger is an agent-declared quantifier-free formula of real arithmetic that, once satisfied, prompts the agent to take an action. Intuitively, the trigger is the reason the agent moves. An autonomous vehicle could set a trigger to be *not enough free space ahead or the desired intersection is reached.* According to the case that comes first, the car would have to take an action. An edge transition only happens when some agent has a satisfied trigger. With each edge transition, each agent gets to update their triggers. We call the set of all possible triggers $Form_{Trig} \subseteq Form_X$.

The definition of HGT extends the definition of hybrid games as follows:

$$\mathcal{G} := (Loc,\ l_0,\ X,\ v_0,\ flow,\ inv,\ Agt,\ Act_\perp,\ E_\perp,\ Form_{Trig}, trig_0)$$

The function $trig_0 : Agt \to Form_{Trig}$ is the initial triggers function. The set $Act_\perp$ includes a *stutter* action $\perp \notin Act$, available for every agent. If an agent's trigger is satisfied and this agent has no enabled actions in $Act$, the agent must take the stutter action $\perp$. This indicates that there is a self-loop edge for every location $l$, $(l, True, \perp, \emptyset, l) \in E_\perp$. A **configuration** in a HGT is a triplet $\langle l, v, trig \rangle$. The initial configuration is $\langle l_0, v_0, trig_0 \rangle$.

**Trigger Formula:** A formula $\varphi$ is called a trigger formula if and only if for every valuation $v$ and every map $flow$ assigning a differential equation to each variable in $X$, there exists a minimum time $t \in \mathbb{R}_{\geq 0}$ to satisfy $\varphi$, i.e. such that $v[flow, t] \models \varphi$ and for all $0 \leq t' < t$, $v[flow, t'] \not\models \varphi$, or if $\varphi$ is never satisfied, i.e. $v[flow, t] \not\models \varphi$ for all $t \in \mathbb{R}_{\geq 0}$. In other words, a formula $\varphi$ is a trigger formula ($\varphi \in Form_{Trig}$) if and only if its solution set is a closed set under the usual topology in $\mathbb{R}^{|X|}$.

This restriction eliminates formulas like $x > 2$ for a variable $x \in X$ where no exact time exists when it is first satisfied if the valuation $v(x) = 0$ and $flow(x) = 1$. On the contrary, $x \geq 2$ is a valid trigger formula.

Given a valuation $v$, a flow $flow$ and a trigger $\varphi$, we define the function *time to trigger* or $ttt(v, flow, \varphi)$ to return the minimum time required to satisfy the trigger $\varphi$ if it exists and $\infty$ otherwise. This definition is extended to configurations $\langle l, v, trig \rangle$ to return the minimum time required to satisfy any of the formulas if such a time exists:

$$ttt(\langle l, v, trig \rangle) = \min \{ ttt(v,\ flow,\ trig(i)) \mid i \in Agt \}.$$

The definition of *legal* transitions in HGT is more restrictive than that in hybrid games. A transition is $\langle l, v, trig \rangle \to \langle l', v', trig' \rangle$ legal if and only if it fulfils the following conditions:

- $\langle l, v \rangle \to \langle l', v' \rangle$ is legal in the hybrid game,
- a time transition $\langle l, v, trig \rangle \xrightarrow{t} \langle l, v[flow(l), trig], trig \rangle$ does not change the triggers function $trig$, and $t \leq ttt(\langle l, v, trig \rangle)$, and
- an edge transition $\langle l, v, trig \rangle \xrightarrow{e} \langle end(e), v[jump(e)], trig' \rangle$ if $v \models trig(i)$ for the agent $i \in Agt$ to blame for the transition.

Along with each edge transition, each player $i$ gets to choose a new trigger $\varphi_i \in Form_{Trig}$ creating a new trigger function mapping $trig'(i) = \varphi$. Similar to [10], if more than one trigger is satisfied at the same time, the agent who gets to take an action is chosen at random.

### 3.0.1. Example

We go back to the example shown in Figure 1. In the current configurations, the green robot has the trigger:

$$green.pos = intersection(h_1, v_4) \vee free\text{-}ahead(green.pos) \leq braking\text{-}distance(green.spd)$$

The orange robot's trigger is similar but has $intersection(h_2, v_3)$.

In this example, we can notice that robots (agents) do not need to calculate the time needed for the trigger to be satisfied when selecting one. Another observation is that a small set of trigger formulas is often sufficient for many systems. This feature makes reasoning about the system significantly easier.

**Remark 2.** *Contrary to guards and invariants, triggers are not part of the game structure but rather part of the players' strategies. A player could choose different triggers in the same location (see Example 3.0.1). While it is possible to extend the game structure by adding more locations and more restrictive guards and invariants to embed the triggers into the game structure, this is not always possible with a finite set of locations.*

**Winning Conditions:** We adopt the winning conditions from [10]. The idea of these winning conditions is that an agent cannot win by preventing time from progressing. A play is a winning play for agent $i$ if time diverges $td$ and the play fulfils their goal $\phi_i$ or if time converges $tc$, but the agent $i$ is not to blame ($blameless_i$) for the time convergence, i.e. the agent $i$ only takes a finite number of actions during the entire play. The set of winning plays for agent $i$ with the desired outcome $\phi_i$ is $(WC(\phi_i) \cap td) \cup (blameless_i \setminus td)$, where $WC(\phi)$ is the set of plays that fulfils $\phi$. As noted in [10], according to these winning conditions, both agents can lose in a two-agent game where one agent has the goal $\phi$ and the other $\neg\phi$. This is when the two agents infinitely take turns blocking time from progressing. I.e. time converges, and neither agent is *blameless*.

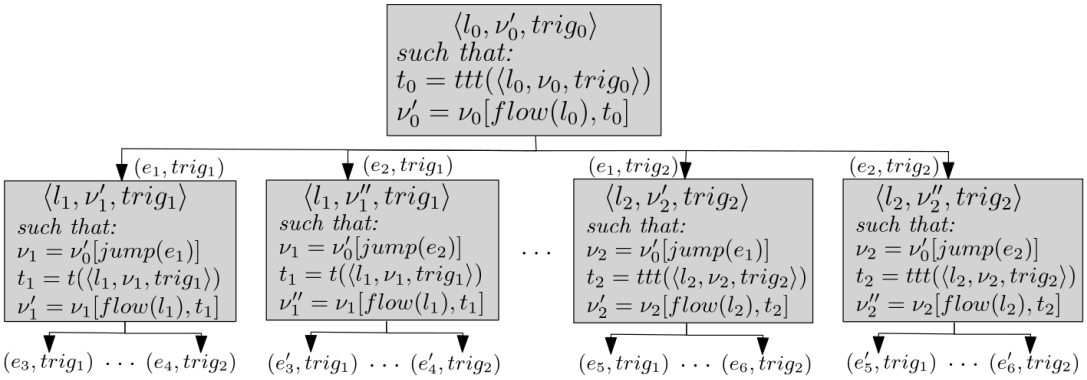



**Figure 2:** Time-abstract discrete game structure.

## 4. Discretizing a Hybrid Game with Triggers

In this section, we show an intuitive way to define a discrete game, such that agent $i \in Agent$ has a winning strategy in the HGT if and only if the same $i$ has a winning strategy in the discrete game. The intuition of the discrete game is to skip any time steps where no triggers are satisfied. This allows time to move in discrete steps. At any configuration $\langle l, v, trig \rangle$ with $v \not\models trig(i)$ for all $i \in Agt$, the game can progress by $ttt(\langle l, v, trig \rangle)$. If a trigger is satisfied, any enabled edge can be taken and the trigger function gets updated.

Due to space limitations, we only briefly describe the discrete game in this paper.

The discrete game is structured as a concurrent game structure (CGS) [14]. The players are the agents of the HGT $Agt$ with the addition of the player $Random \notin Agt$ to select the agent who gets to take an action when more than one trigger is satisfied. While only one action is taken in each step, the game is concurrent to allow all agents to select new triggers simultaneously. The actions available for the agents are $Act \times Form_{Trig}$. The actions available for $Random$ are the set $Agt$.

The set of states $S$ of the discrete game is defined inductively:

- $s_0 = \langle l_0, v_0, trig_0 \rangle \in S$ is the initial state.
- If the state $s = \langle l, v, trig \rangle \in S$, then:
  - if no trigger is satisfied $v \not\models trig(i)$ for all $i \in Agt$ and $ttt(\langle l, v, trig \rangle) \in \mathbb{R}_{\geq 0}$, then $\langle l, v[flow(l), ttt(\langle l, v, trig \rangle)], trig \rangle \in S$,
  - otherwise for every $e \in E_\perp$ enabled at $s$ and for every function $trig' : Agt \to Trigger$, the state $\langle end(e), v[assign(e)], trig' \rangle \in S$.

A time-abstract game is visualized in Figure 2, where the game evolves only based on the players' choices.

Given that the number of edges $E_\perp$ is finite and the number of trigger formulas is countable [11], each layer of the tree is countable. The state space of the entire discrete game is, therefore, countable. Each discrete game state is labelled with formulas that its valuation satisfies and are relevant to the agents' winning conditions. These formulas serve as atomic propositions in the discrete game. Additionally, the states are labelled according to their position in the tree. We use a set of atomic propositions inspired by [10] to prevent agents from winning by blocking the passing of time. The boolean proposition *tick* is true if the global time has passed an integer value compared to the state's parent in the tree. The atomic propositions $blame_i$ for $i \in Agt$ express the agent whose action reached this state.

A play is winning for an agent $i$ if (1) the play satisfies the desired outcome $\phi$ (expressed as a temporal property) and has an infinite number of states marked with *tick* or (2) the play has a finite number of states marked with *tick* and a finite number of states marked with $blame_i$. The existence of a winning strategy for the agent $i$ can be then expressed in the notation of ATL* [14] as follows:

$$\langle\!\langle \{ i \} \rangle\!\rangle (\phi \wedge \Box \Diamond tick) \vee (\Diamond \Box \neg tick \wedge \Diamond \Box \neg blame_i) \tag{2}$$

This reduces the winning conditions in a hybrid game with triggers to an ATL* model checking problem over countable state space. This is shown to be decidable in [15, 16].

## 5. Discussion and Conclusion

Extending hybrid games with triggers has multiple advantages and applications beyond the decidable fragment of hybrid games brought on by triggers.

In addition to the significant decidability results, triggers could help improve system understandability. An AI system, for instance, could be trained to choose (or form) a trigger formula and not act again until this formula is satisfied. Such a feature would have major benefits to AI verification and explainability.

In summary, hybrid games with triggers (HGT) offer a powerful framework for reasoning about and verifying multi-agent hybrid systems. We show that by incorporating agents' rationale into the game model, HGT can effectively reduce a hybrid game to a decidable discrete game without restricting its continuous or discrete dynamics.

## Acknowledgments

This research was supported by the Innovation Campus for Future Mobility (www.icm-bw.de) and by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1608 Convide (www.sfb1608.kit.edu/index.php).

## References

[1] A. Platzer, Differential dynamic logic for hybrid systems, J. Autom. Reason. 41 (2008) 143–189. URL: https://doi.org/10.1007/s10817-008-9103-8. doi:10.1007/S10817-008-9103-8.

[2] A. Platzer, Differential game logic, ACM Transactions on Computational Logic (TOCL) 17 (2015) 1–51.

[3] P. Höfner, B. Möller, An algebra of hybrid systems, The Journal of Logic and Algebraic Programming 78 (2009) 74–97.

[4] R. Neves, L. S. Barbosa, Hybrid automata as coalgebras, in: Theoretical Aspects of Computing–ICTAC 2016: 13th International Colloquium, Taipei, Taiwan, ROC, October 24–31, 2016, Proceedings 13, Springer, 2016, pp. 385–402.

[5] T. A. Henzinger, B. Horowitz, R. Majumdar, Rectangular hybrid games, in: J. C. M. Baeten, S. Mauw (Eds.), CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings, volume 1664 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 320–335. URL: https://doi.org/10.1007/3-540-48320-9_23. doi:10.1007/3-540-48320-9_23.

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, Theor. Comput. Sci. 138 (1995) 3–34. URL: https://doi.org/10.1016/0304-3975(94)00202-T. doi:10.1016/0304-3975(94)00202-T.

[7] T. A. Henzinger, P. W. Kopke, A. Puri, P. Varaiya, What's decidable about hybrid automata?, J. Comput. Syst. Sci. 57 (1998) 94–124. URL: https://doi.org/10.1006/jcss.1998.1581. doi:10.1006/JCSS.1998.1581.

[8] R. Alur, T. A. Henzinger, G. Lafferriere, G. J. Pappas, Discrete abstractions of hybrid systems, Proceedings of the IEEE 88 (2000) 971–984.

[9] P. Bouyer, T. Brihaye, F. Chevalier, O-minimal hybrid reachability games, Logical Methods in Computer Science 6 (2010).

[10] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, M. Stoelinga, The element of surprise in timed games, in: R. M. Amadio, D. Lugiez (Eds.), CONCUR 2003 - Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings, volume 2761 of *Lecture Notes in*

*Computer Science*, Springer, 2003, pp. 142–156. URL: https://doi.org/10.1007/978-3-540-45187-7_9. doi:10.1007/978-3-540-45187-7_9.

[11] A. Tarski, A decision method for elementary algebra and geometry, in: B. F. Caviness, J. R. Johnson (Eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition, Springer Vienna, Vienna, 1998, pp. 24–84.

[12] T. A. Henzinger, Hybrid automata with finite bisimulations, in: Z. Fülöp, F. Gécseg (Eds.), Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 324–335.

[13] F. Laroussinie, N. Markey, G. Oreiby, Model-checking timed atl for durational concurrent game structures, in: International Conference on Formal Modeling and Analysis of Timed Systems, Springer, 2006, pp. 245–259.

[14] R. Alur, T. A. Henzinger, O. Kupferman, Alternating-time temporal logic, J. ACM 49 (2002) 672–713. URL: https://doi.org/10.1145/585265.585270. doi:10.1145/585265.585270.

[15] S. Schewe, Atl* satisfiability is 2exptime-complete, in: International colloquium on automata, languages, and programming, Springer, 2008, pp. 373–385.

[16] F. Mogavero, A. Murano, G. Perelli, M. Y. Vardi, Reasoning about strategies: On the model-checking problem, ACM Transactions on Computational Logic (TOCL) 15 (2014) 1–47.

# Exploring LLMs and Semantic XAI for Industrial Robot Capabilities and Manufacturing Commonsense Knowledge*

Muhammad Raza Naqvi[1,*,†], Arkopaul Sarkar[1,†], Farhad Ameri[2,†], Linda Elmhadhbi[3,†], Thierry Louge[1,†] and Mohamed Hedi Karray[1,†]

[1]*Laboratoire Génie de Production(LGP) Université de Technologie de Tarbes(UTTOP), 47 Av. d'Azereix, 65000, Tarbes, France.*

[2]*School Of Manufacturing Systems: Networks, Arizona State University, Mesa, Arizona, AZ 85212, United States*

[3]*INSA Lyon, Université Lumiere Lyon 2, Université Claude Bernard Lyon 1, Université Jean Monnet Saint-Etienne,DISP UR4570, 0 Av.Albert Einstein, Villeurbanne, 69621, Rhone, France.*

**Keywords**
Advertise Capabilities, Operational Capabilities, Manufacturing Common-Sense knowledge, Semantic Explainable AI,

## 1. Context and Motivations

In the context of Industry 4.0, flexible manufacturing is especially essential for developing future factories with enhanced planning, scheduling, and control [1]. The quick and effective adaptation in the production line in response to customers' requirements or facing unwanted situations will considerably promote flexibility in manufacturing. For instance, due to the COVID-19 crisis[1], some companies have been re-purposing their production lines to join the fight against the pandemic[2] (e.g., perfumes to hand sanitizers and vehicles to ventilators). The question is how the human expert who runs the factory can know if the currently available resources (machines, tools, equipment, and technicians) can quickly and efficiently switch to a specific production process based on new work orders [2].

Similar disruptions may arise from extreme weather, longer-term climate change, declining international order, economic crises, changing societal priorities, cyber threats, or terrorism. Furthermore, flexibility in the manufacturing industries is more critical than ever to tackle ever-growing product variability, supply chain volatility, and unpredictability of customer requirements. Product life cycles are becoming more and more dynamic; also, at the same time, the number of product variants continues to grow. There is a need for more flexible, trustworthy, and efficient manufacturing processes.Traditional manufacturing paradigms such

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

*Corresponding author. razisyed4@gmail.com



[1]https://www.weforum.org/agenda/2020/03/from-perfume-to-hand-sanitiser-tvs-to-face-masks-how-companies-are-changing-track-to-fight-covid-19/

[2]https://www.unido.org/news/covid-19-critical-supplies-manufacturing-repurposing-challenge

as lean manufacturing, just-in-time production, and KANBAN [3] systems often struggle to adapt to unforeseen disruptions. These systems can fall into a "rupture condition" when faced with unexpected challenges like pandemics, economic crises, or cyber threats. Rapid reconfiguration of operations highlights the need for flexibility in manufacturing to manage product variability, supply chain volatility, and unpredictability of customer demand.

Most production planning still depends on industry standards, best practices, manufacturing know-how, and machinists' experiential knowledge. Despite the advent of Artificial Intelligence (AI), industries still depend on human expertise to make process planning decisions in an unprecedented situation because of uncertain or low expectations of AI investments (Trust). Personal judgment overrides AI-based decision-making (Human-reasoning).

However, human experts will only trust and accept AI results if the automated decision-making process is transparent [3][4]. If the AI system decides that a production step or plan matches or does not match the capabilities of a particular machine, the human expert should understand why this decision is made (i.e., which of the machine's capabilities leads to such a decision). Accordingly, a trustworthy AI, commonly called explainable AI (XAI) [5], should be able to explain why a particular decision was reached in a way that human experts can understand, for example, in the case of real-time decision-making for manufacturing operations, why some processes were performed, what process is currently being performed, and what processes should occur in the future. However, the explainability of the AI model standalone is not enough; decision logic must also be transparent and grounded in interpretable knowledge structures to ensure accurate understanding, paving the way for integrating semantics alongside these models.

Semantics is the study of meaning [6]; it corresponds to providing structured and meaningful explanations that are not only data-driven but also align with real-world concepts. Ontologies are crucial to providing this semantic structure and adding context to the explanations. Ontologies are defined as an explicit formal semantic representation of knowledge through logical axioms [7].

The use of semantics in explaining various types of supervised and unsupervised learning models was presented by Seeliger et al. [8]. In a narrower scope, Bianchi et al. [9] reviewed methods of embedding knowledge graphs in ML models to achieve explainability. Regarding the quality of the explanations, past research in behavioral psychology [10][11] showed that three qualities make the explanations more intuitive for humans. Such explanations must be more straightforward in that they include fewer general reasons, mention well-known events as reasons, and be coherent and consistent with prior knowledge. This stresses the need to adopt XAI techniques that are not solely data-driven (statistical learning) but embed semantic-driven symbolic reasoning in the prediction models from the ground up. Research is scarce in this direction, with only a handful of other projects targeting XAI-based hybrid AI for manufacturing, such as AI4EU44[4]or XMANAI[5]. Humans expect explainable decisions based on what they consider commonsense (i.e., simple facts about people and everyday life, data evidence, and causal reasoning).

[3] https://leanmanufacturingtools.org/kanban/
[4] https://www.ai4europe.eu/
[5] https://ai4manufacturing.eu/

Commonsense knowledge (CSK) explanation is the most meaningful and sought-after explanation technique [12]. According to the state of the art, incorporating and implementing CSK capabilities into AI can enhance the overall manufacturing potential and accelerate the growth of AI applications in the industry [13]. However, representing CSK related to manufacturing is challenging [14]. In the literature, significant progress has been made in four areas related to CSK in AI, mainly reasoning about taxonomic categories, logic, time and space, and actions [15].

Yet, there seems to be no CSK for the industry that integrates all main aspects of manufacturing process requirements. The manufacturing commonsense knowledge (MACS) should cover generic manufacturing background knowledge that human experts, such as machinists, planners, and shopfloor managers, carry as part of their experience to know or assume to understand and reason about information (or situation) that they are dealing with (e.g., a machine needs some kind of energy, a machine can breakdown, the production process needs raw materials, etc.). Generating commonsense explanations requires integrating richer domain knowledge [16] represented through ontologies.

The ontologies should cover production-relevant domains, including machine capabilities, production process, product specifications, raw materials, etc.

We examine current methods for manufacturers to increase transparency in AI system decision-making as a state of the art methodology structure shown in Fig.1. Two promising areas of XAI are ontology-based (O-XAI) and semantic-based (S-XAI), which use semantic information to provide human-readable explanations of AI decisions. Translating AI algorithm decision paths to meaningful explanations using semantics, O-XAI, and S-XAI helps humans identify cross-cutting concerns influencing AI system decisions. We discuss the pros and cons of using O-XAI and S-XAI systems in manufacturing and future research potential to guide researchers and practitioners in utilizing these explainable systems for decision-making [18].

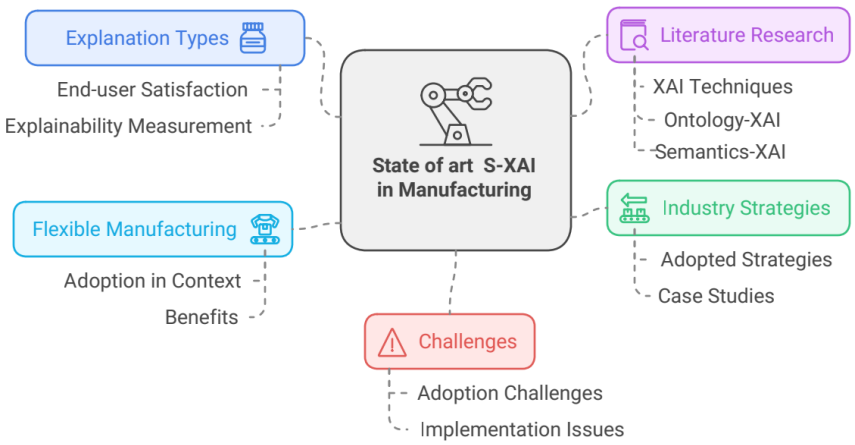



**Figure 1:** Semantics based XAI in manufacturing

"CSK and Hybrid AI for Trustful and flexible manufacturing 4.0" (Chaikmat 4.0)[6] is a research project funded by the French National Agency of Research ANR that aims to add flexibility and transparency to manufacturing through trustful automatic decision.

In the context of flexible manufacturing, the selection of resources must also consider the dynamically changing conditions of the shop floor, including factors such as the age of the machines, maintenance histories, and the availability of operators. These considerations are crucial to determining the actual performance of machines and equipment. CHAIKMAT's core objective is to evaluate whether available machines can execute specific production processes effectively.

The project seeks to add flexibility and transparency to manufacturing through reliable automatic decision-making systems. These systems are designed to provide human experts with meaningful explanations about decisions, using MACS to make the process clear and understandable. To meet these challenges, CHAIKMAT,s proposes a hybrid approach that bridges the gap between human expertise and AI-based decision-making. This strategy proposes efficient resource use and enhanced industrial flexibility, laying the groundwork for more advanced manufacturing operations [17].

## 2. Thesis Objectives

The primary objective of this thesis, within the scope of the project, is to investigate the use and effects of ontology-based models in manufacturing in the context of semantic reasoning, explainability, and efficient formalization of machine capabilities. The focus is primarily on robotics, with the intention of improving the explainability of how robots are assigned tasks based on their capabilities. This involves developing and validating ontological models to encapsulate MACS and robotics capabilities for decision-making and explainability. The following are the objectives that have been outlined below:

- Formalization of machine specifications that include capabilities, capacities, functions, quality, and process characteristics, focusing on robotics.
- Establish a framework for identifying MACS patterns, extracting MACS, formalizing MACS using standard vocabulary, converting them into semantic rules, and leveraging MACS patterns within decision-making processes.
- Utilization of machine capabilities and MACS patterns, alongside a neural network framework, to explain whether an existing set of machines can perform a specific task or not based on robot capabilities.

## 3. Research Questions

Developing methods for utilizing machine specifications and MACS patterns, along with integrating a semantics-based explainable AI (S-XAI) framework, is crucial to achieving the primary objectives of this thesis. These elements will enhance decision-making processes, increase user

[6] https://chaikmat-anr.uttop.fr

trust, and provide transparency. The following research questions have been formulated to guide this investigation:

- How can we formalize key notions such as Capability, Capacity, Function, Quality, and Process Characteristics in the context of robotics?
- How can MACS be extracted, formalized, and integrated into decision-making processes to enhance explainability?
- How can ontologies and knowledge graphs can facilitate the creation of explanations that align with human understanding while enhancing the use of S-XAI in manufacturing decision-making processes?

## 4. Contribution

This thesis presents three critical contributions to improving the explainability of the decision-making process in manufacturing by integrating robotics capabilities, MACS, and S-XAI.

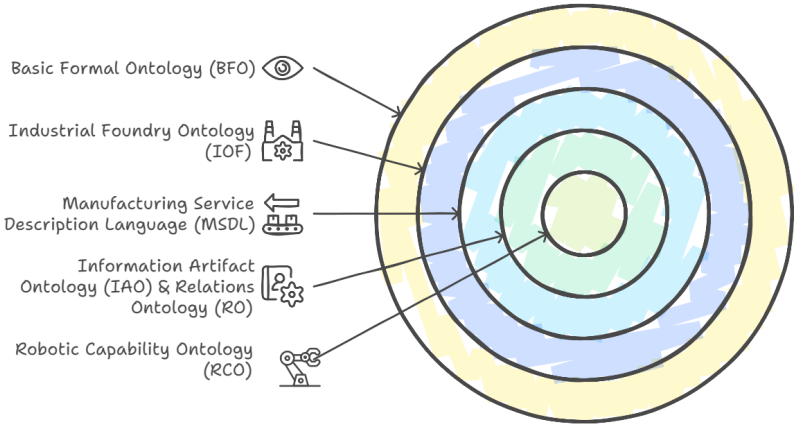



**Figure 2:** Robot Capability Ontology

Our first contribution is the development of the Robotic Capability Ontology (RCO) [19], an application ontology specifically designed to model robotic capabilities [19]. RCO utilizes the Manufacturing Service Description Language (MSDL) [20], a domain reference ontology created for manufacturing services and aligned with the Basic Formal Ontology (BFO) [21], Industrial ontology Foundry (IOF) [22], Information Artifact Ontology (IAO) [23], and Relations ontology (RO) [24] as shown in Fig,2. MSDL's modular structure and domain-neutral classes allow RCO to describe and expand upon robotic capabilities accurately. This enables the design of an ontological model that precisely captures robotic specifications from its specification manual as advertised capabilities to actual capabilities in an operational environment as operational capabilities.

Our second contribution introduces the concept of MACS. We propose a methodology for identifying MACS patterns, extracting, formalizing, and modeling this knowledge, and a standardized vocabulary that organizes MACS into semantic rules, as shown in Fig.3. These semantic

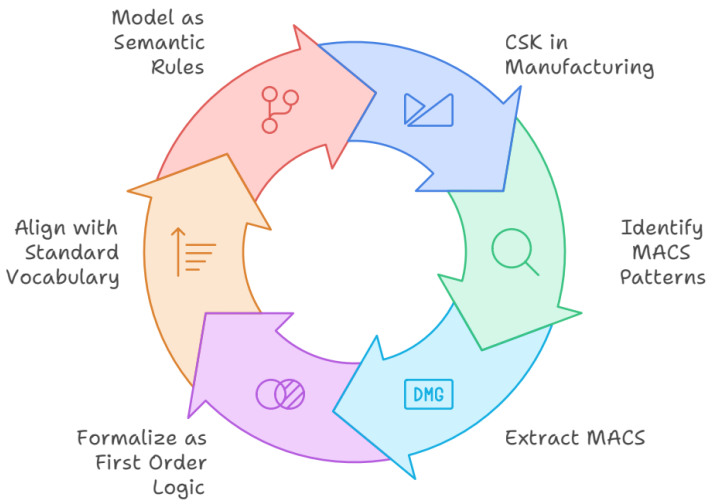



**Figure 3:** Manufacturing Commonsense Knowledge

rules are then converted into schema languages such as SPARQL and Datalog rules, which are used to create a MACS Knowledge Graph (MACS-KG) to demonstrate the applicability of the proposed method.

Our third contribution is developing a (S-XAI) framework as shown in Fig.4. that incorporates a neural network model trained on historical data about different tasks to predict robotic operational capabilities such as 'Repeatability' and 'Precision' on a given set of coordinates for a new task.

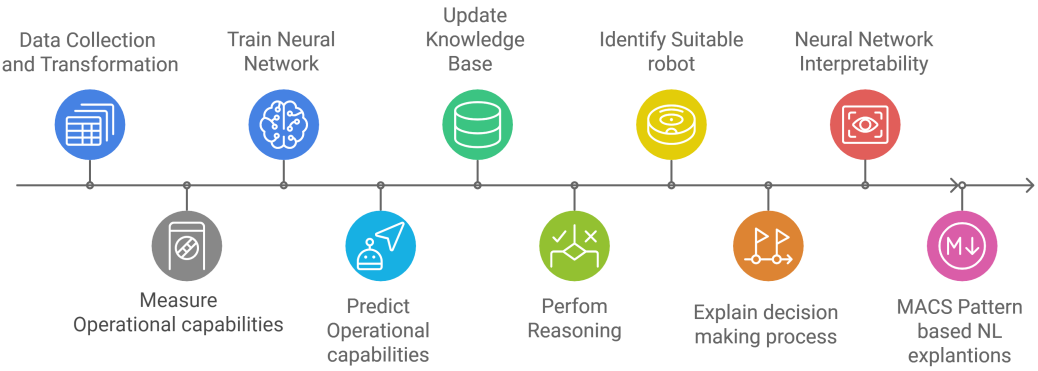



**Figure 4:** Semantic-XAI

Also, utilizing XAI techniques like Local Interpretable Model-Agnostic Explanations (LIME), Partial Dependency Plots (PDP), and Permutation Features Importance (PF) for the prediction of operational capabilities from neural networks alongside natural language explanations based on MACS patterns, the system provides clear, logical, and understandable explanations for each prediction.

## 5. Summary

We propose an S-XAI framework to address the concern about explainability issues related to the decision-making process in the manufacturing industry. The framework combines neural networks for predictive analysis of robot operational capabilities with rule-based reasoning grounded in MACS. This approach brings transparency and explainability to decision-making, ensuring stakeholders can trust and understand automated decisions.

A vital feature of the framework is the ability to integrate symbolic and sub-symbolic reasoning paradigms, enabling real-time, explainable decisions in manufacturing environments. By incorporating a neural network, the system will be scalable with the increasing data volume and continuously improve through active learning. At the same time, the use of manufacturing commonsense knowledge ensures that explanations provided to users are contextually relevant and easy to comprehend, fostering user trust and system acceptance.

## Acknowledgments

This work is performed within the CHAIKMAT project funded by the French National Research Agency (ANR) under grant agreement " ANR-21-CE10-0004-01"

## References

[1] Nogalski, B., Niewiadomski, P., & Szpitter, A. (2020). Agility Versus Flexibility? The Perception of Business Model Maturity in Agricultural Machinery Sector Manufacturing Companies. Central European Management Journal, 2020(3), 57–97. https://doi.org/10.7206/cemj.2658-0845.27

[2] Järvenpää, E., Siltala, N., Hylli, O., & Lanz, M. (2017). Capability Matchmaking Procedure to Support Rapid Configuration and Re-configuration of Production Systems. Procedia Manufacturing, 11, 1053–1060. https://doi.org/10.1016/j.promfg.2017.07.216

[3] Ma, S., Lei, Y., Wang, X., Zheng, C., Shi, C., Yin, M., & Ma, X. (2023). Who Should I Trust: AI or Myself? Leveraging Human and AI Correctness Likelihood to Promote Appropriate Trust in AI-Assisted Decision-Making. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 29, 1–19. https://doi.org/10.1145/3544548.3581058.

[4] Yang, Y. (2024). Who Should I Trust: Human-AI Trust Model in AI Assisted Decision-Making. Lecture Notes in Education Psychology and Public Media, 41(1), 236–241. https://doi.org/10.54254/2753-7048/41/20240805

[5] R, J. (2024). Transparency in AI Decision Making: A Survey of Explainable AI Methods and Applications. Advances in Robotic Technology, 2(1), 1–10. https://doi.org/10.23880/art-16000110

[6] Loebner, S. Understanding Semantics. Routledge. (2013) https://doi.org/10.4324/9780203528334

[7] Ontology. (2020). Ontology. https://doi.org/10.4135/9781526421036869920

[8] Seeliger, A., Pfaff, M., Krcmar, H.: Semantic web technologies for explainable machine learning models: a literature review. CEUR Workshop Proc. 2465(October), 30–45 (2019)

[9] Bianchi, F., Rossiello, G., Costabello, L., Palmonari, M., Minervini, P.: Knowledge graph embeddings and explainable AI. arXiv. (Apr. 2020). https://doi.org/10.3233/SSW200011
[10] Lombrozo, T.: Explanation and abductive inference. In: The Oxford Handbook of Thinking and Reasoning, pp. 260–276, New York, NY, US: Oxford University Press (2012)
[11] Thagard, P.: Explanatory coherence. Behav. Brain Sci. 12(3), 435–467 (1989). https://doi.org/10.1017/S0140525X00057046
[12] Reiter, E. (2019). Natural Language Generation Challenges for Explainable AI. Proceedings of the 1st Workshop on Interactive Natural Language Technology for Explainable Artificial Intelligence (NL4XAI 2019). https://doi.org/10.18653/v1/w19-8402
[13] Logic, language and commonsense. (1987). Artificial Intelligence, 169–176. https://doi.org/10.1016/b978-0-08-034112-5.50020-6
[14] Rehse, J.-R., Mehdiyev, N., Fettke, P.: Towards explainable process predictions for industry 4.0 in the DFKI-smart-lego-factory. KI - Künstliche Intell. 33(2), 181–187 (2019). https://doi.org/10.1007/s13218-019-00586-1
[15] Davis, E.: Logical formalizations of commonsense reasoning: a survey. J. Artif. Intell. Res. 59, 651–723 (2017). https://doi.org/10.1613/jair.5339
[16] Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C. A., & Meersman, R. (2019). Correction to: On the Move to Meaningful Internet Systems. On the Move to Meaningful Internet Systems: OTM 2019 Conferences, C1–C1. https://doi.org/10.1007/978-3-030-33246-4_47
[17] A. Sarkar, M. R. Naqvi, L. Elmhadhbi, D. Sormaz, B. Archimede, M. H. Karray, CHAIK-MAT 4.0 - Commonsense Knowledge and Hybrid Artificial Intelligence for Trusted Flexible Manufacturing, Springer International Publishing, 2023, p. 455–465. https://doi:10.1007/978-3-031-17629-6_47.
[18] M. R. Naqvi, L. Elmhadhbi, A. Sarkar, B. Archimede, M. H. Karray, Survey on ontology-based explainable ai in manufacturing, Journal of Intelligent Manufacturing (2024). https://doi:10.1007/s10845-023-02304-z.
[19] M. R. Naqvi, A. Sarkar, F. Ameri, S. N. Araghi, M. H. Karray, Application of msdl in modeling capabilities of robots (2023). https://ceur-ws.org/Vol-3595/paper7.pdf.
[20] Ameri, F., & Dutta, D. (2006). An Upper Ontology for Manufacturing Service Description. Volume 3: 26th Computers and Information in Engineering Conference. https://doi.org/10.1115/detc2006-99600
[21] J. N. Otte, J. Beverley, A. Ruttenberg, Bfo: Basic formal ontology, Applied ontology 17. https://doi.org/10.3233/AO-220262 (2022) 17–43.
[22] Drobnjakovic, M., Kulvatunyou, B., Ameri, F., Will, C., Smith, B., & Jones, A. (2022). The industrial ontologies foundry (IOF) core ontology. embeddings and explainable AI. arXiv. (Apr. 2020). https://doi.org/10.3233/SSW200011
[23] Smith, B., Malyuta, T., Rudnicki, R., Mandrick, W., Salmen, D., Morosoff, P., ... & Parent, K. (2013). IAO-Intel: an ontology of information artifacts in the intelligence domain.
[24] W. J. Wildman, An introduction to relational ontology, The trinity and an entangled world: Relationality in physical science and theology (2010) 55–73.

# Towards Correct-by-Construction Machine-Learnt Models

Thomas Flinkow*, Barak A. Pearlmutter and Rosemary Monahan

*Department of Computer Science, Maynooth University, Maynooth, Co. Kildare, Ireland*

**Abstract**

Various neural network verifiers have been developed to ensure that a neural network satisfies desired properties after training. A promising approach for creating correct-by-construction machine-learnt models is to incorporate explicit logical constraints into the training process via so-called differentiable logics. This paper provides an overview of our research area, our preliminary results, as well as an outline of future research directions.

**Keywords**

formal verification, machine learning, differentiable logics

## 1. Introduction

It has been shown that neural networks fail to learn background knowledge from data alone and are susceptible to adversarial inputs [1, 2], which has implications for their use in safety-critical domains.

Numerous verifiers for neural networks have emerged in the past few years, such as Reluplex [3], Marabou [4, 5], Branch-and-Bound [6], NNV [7], and $\alpha$, $\beta$-CROWN [8–13], winner of the recent Neural Network Verification Competitions (VNN-COMP) [14–17]. For an overview of state-of-the art verifiers, we refer the interested reader to [18–21].

Verification of neural networks is typically limited to neural networks with fixed weights that have ceased learning [22]. A step in the direction of correct-by-construction neural networks are so called *differentiable logics*, used to incorporate logical constraints into the machine learning process.

## 2. Background

**Machine learning.** In gradient-based machine learning, optimal parameters $\boldsymbol{\theta}^+$ (such as neural network weights) are determined by minimising a *loss function*, $\mathcal{L}$, which quantifies the error between the predicted output and the desired output. This optimisation is typically achieved using gradient descent methods. The goal is to find the set of parameters $\boldsymbol{\theta}^+$ that minimises the loss function, formally expressed as

$$\boldsymbol{\theta}^+ = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}), \tag{1}$$

where $\boldsymbol{x}$ represents the input data and $\boldsymbol{y}$ denotes the corresponding desired output.

**Differentiable logics.** The idea of learning with constraints is to incorporate a logical constraint $\phi$ into this optimisation process by translating the logical constraint into an additional loss term $\mathcal{L}_\phi$.

$$\boldsymbol{\theta}^+ = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) + \lambda \mathcal{L}_\phi(\boldsymbol{x}, \boldsymbol{y}). \tag{2}$$

Note that the additional loss term introduces a new hyperparameter $\lambda$ that is responsible for balancing the different loss terms. As explained in Section 3, in our experimental evaluation [23] we used the adaptive loss-balancing approach GradNorm [24] in order to find close-to-optimal values for $\lambda$.

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

*Corresponding author.

thomas.flinkow@mu.ie (T. Flinkow); barak@pearlmutter.net (B. A. Pearlmutter); rosemary.monahan@mu.ie (R. Monahan)

0000-0002-8075-2194 (T. Flinkow); 0000-0003-0521-4553 (B. A. Pearlmutter); 0000-0003-3886-4675 (R. Monahan)

Various translations that map logical constraints into real-valued, differentiable functions have been defined in the literature, such as semantic loss [25], DL2 [26], designed specifically for incorporating constraints into neural networks, or fuzzy logic based ones [27–30], which exploit the fact that fuzzy logics are real-valued logics that often use operators that happen to be differentiable-almost-everwhere.

**Specialised network architectures.** Note that incorporating logical constraints into the machine learning pipeline via additional loss terms as done in Eq. (2) does not guarantee constraint satisfaction; other approaches exist that incorporate logical constraints into the network architecture, such as proposed by Li and Srikumar [31], DeepProbLog [32], Logic Tensor Networks (LTNs) [33, 34], MultiPlexNet [35], CNN [36], and CNN$^+$ [37].

## 3. Contributions to Date

The theory of these differentiable logics is well-studied [38–41] in the literature with respect to various interesting properties, such as (1) the shadow-lifting [42] property of a conjunction $x \wedge y$, which requires the truth value of the conjunction to increase when the truth value of one of its contituents increases, (2) whether implication operators admit classical logic reasoning such as Modus Ponens and Modus Tollens [39], and (3) the logical consistency [38] of operators, which looks at the maximum truth value obtainable for tautologies when using certain operators.

Given the wide range of possible logic translations available, our initial research question was: what is the optimal translation for use in training?

To address this question, we provide in [23] an experimental comparison of differentiable logic operators. Additionally, we provide a Python implementation [43] of various differentiable logics in PyTorch [44], implemented in a way that makes it easy to train any neural network on any dataset with arbitrary constraints.

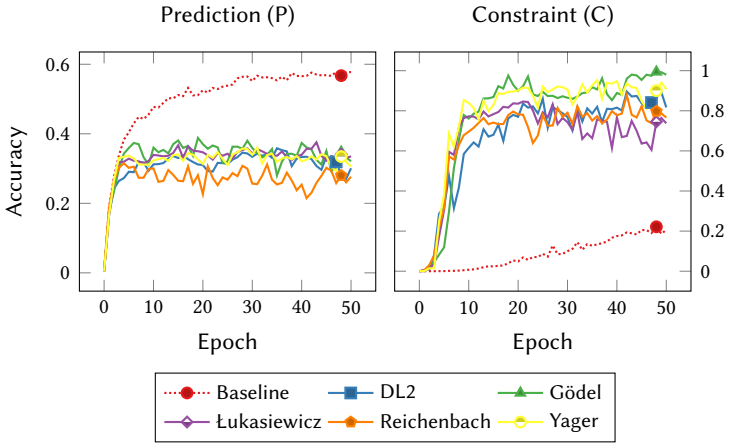
In order for our experimental comparison to be as fair as possible, we utilised Projected Gradient Descent (PGD) [45] to use a constraint counterexample in training as initially suggested by [26], which allows each logic to have the most impact on the learning process, and additionally we use the adaptive loss-balancing approach GradNorm [24] in order to estimate the parameter $\lambda$ from Eq. (2) to balance the different loss terms, allowing each logic to perform at its best.

**Experimental results.** We obtained somewhat surprising results: while we expected to confirm theoretic results from the literature, we found that shadow-lifting conjunctions were not necessarily the best choice; neither were those implications that closely follow Modus Ponens and Modus Tollens reasoning. In general, training with any differentiable logic will lead to improved constraint satisfaction (albeit at an expense of prediction accuracy, as reported previously by Tsipras et al. [46]). However, the performance of the differentiable logics depends highly on the specific task at hand.

For example, we compared the performance of five different logic translation for training a neural network on the German Traffic Sign Recognition Benchmark (GTSRB) [47] to satisfy the constraint "the sum of probabilities of all elements in a group of related traffic signs should either be very high or very low". Here, we consider groups of related traffic signs (e.g. the group of all speed limit signs) in order to add background knowledge into the network.

As can be seen in Fig. 1, training with any differentiable logic leads to improved constraint accuracy and reduced prediction accuracy, however, the difference between the different logics is not as pronounced as expected from their theoretical properties.

**Conclusions for future research.** Instead of trying to find a single best one-size-fits-all differentiable logic that should be used in all scenarios, it might prove to be more fruitful to investigate what logical constraints mandate what properties the logic translation should exhibit. In the following section, we collate some interesting research areas which we have identified and which we plan to investigate in the future.




| Logic | P | C |
|---|---|---|
| Baseline | 56.73 | 22.15 |
| DL2 | 31.92 | 84.12 |
| Gödel | **34.76** | **99.39** |
| Łukasiewicz | 34.13 | 74.54 |
| Reichenbach | 28.00 | 79.76 |
| Yager | 33.44 | 90.06 |

**Figure 1:** Training a network to satisfy a logical constraint on GTSRB with different logics. Surprisingly, the best-performing logic is the Gödel fuzzy logic, despite not having favourable theoretical properties such as shadow-lifting. Here, "Prediction Accuracy" is the percentage of correct predictions, and "Constraint Accuracy" the percentage of the constraint being satisfied.

## 4. Areas for Future Work

**Specifications for machine learning.** A common problem in the machine learning context is the lack of well-defined, general-purpose specifications [48–50] beyond often-used properties such as local robustness, which requires the neural network to be stable against slight perturbations to an input.

Additionally, despite there being complete verification techniques based on SMT or abstract interpretation, these require being able to specify a meaningful region of the input space. This is often infeasible in all but the most low-dimensional, interpretable settings such as the verification [3] of the experimental neural network compression [51] of the airborne collision avoidance system ACAS Xu [52], where meaningful regions of the input space can be expressed via constraints on the position and velocities of different aeroplanes.

For high-dimensional input spaces such as encountered in image classification, distinguishing meaningful images from noise is usually impossible, and verification is therefore usually limited to point-wise verification, which cannot provide any guarantees for the network behaviour on unseen data.

Going forward, it might prove to be beneficial to explore types of general-purpose properties (such as robustness or monotonicity) one might expect a neural network to satisfy across various applications.

**Expressivity of differentiable logics.** Logical constraints used in training are usually expressed in propositional logic, as in the ROAD-R dataset [53] for autonomous driving, which incorporates background knowledge such as ¬(Pedestrian ∧ Cyclist) or ¬(Traffic light green ∧ Traffic light red) into video frames. These constraints are sufficient to correct the network predictions if they do not align with the background knowledge, however, the authors note that future extensions of the dataset will investigate more expressive properties beyond propositional logic.

While properties such as local robustness [54] around point $\boldsymbol{x}_0$ are usually expressed as

$$\forall \boldsymbol{x}.\ ||\boldsymbol{x} - \boldsymbol{x}_0||_\infty \leq \epsilon \rightarrow ||\mathcal{N}(\boldsymbol{x}) - \mathcal{N}(\boldsymbol{x}_0)||_\infty \leq \delta, \tag{3}$$

the universal quantification is normally handled outside of the constraints by employing PGD to approximate the worst possible perturbation in the neighbourhood of $\boldsymbol{x}_0$ as initially suggested by Fischer et al. [26], however, a unifying approach capable of handling general universal (and existential) quantifiers is provided by Ślusarz et al. [40].

Going beyond first-order logic, especially in contexts such as video or natural language processing, one might like to employ temporal properties to model time-dependent behaviours. There are already

differentiable temporal logics [42, 55–57]. We plan to investigate the ways in which these logics differ and identify the strengths and weaknesses of each.

Additionally, Farrell et al. [50] suggest there could be a need for probabilistic properties. To this end, approaches have been developed such as DeepProbLog [32] that allow for incorporating probabilistic constraints into neural networks.

**Certified training.** Using PGD to find the worst perturbation around a point as done for Eq. (3) does not provide any guarantees as it minimises a lower bound on the worst-case loss [19]. Instead of finding a worst perturbation, it would be interesting to investigate approaches based on certified training such as proposed by [58–61].

This area will be the immediate focus of our work, as we expect it to provide a solid foundation that all subsequent research efforts into expressive specifications and logics can benefit from.

## Acknowledgments

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under grant number 20/FFP-P/8853.

## References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, 2014. doi:10.48550/arXiv.1312.6199. arXiv:1312.6199.

[2] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and Harnessing Adversarial Examples, 2015. doi:10.48550/arXiv.1412.6572. arXiv:1412.6572.

[3] G. Katz, C. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer, Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks, in: R. Majumdar, V. Kunčak (Eds.), Computer Aided Verification, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2017, pp. 97–117. doi:10.1007/978-3-319-63387-9_5.

[4] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, C. Barrett, The Marabou Framework for Verification and Analysis of Deep Neural Networks, in: I. Dillig, S. Tasiran (Eds.), Computer Aided Verification, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2019, pp. 443–452. doi:10.1007/978-3-030-25540-4_26.

[5] H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, P. Huang, O. Lahav, M. Wu, M. Zhang, E. Komendantskaya, G. Katz, C. Barrett, Marabou 2.0: A Versatile Formal Analyzer of Neural Networks, 2024. doi:10.48550/arXiv.2401.14461. arXiv:2401.14461.

[6] R. Bunel, I. Turkaslan, P. Torr, M. Pawan Kumar, J. Lu, P. Kohli, Branch and bound for piecewise linear neural network verification, Journal of Machine Learning Research 21 (2020).

[7] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, T. T. Johnson, NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems, in: S. K. Lahiri, C. Wang (Eds.), Computer Aided Verification, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 3–17. doi:10.1007/978-3-030-53288-8_1.

[8] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, L. Daniel, Efficient Neural Network Robustness Certification with General Activation Functions, 2018. doi:10.48550/arXiv.1811.00866. arXiv:1811.00866.

[9] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, C.-J. Hsieh, Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond, 2020. doi:10.48550/arXiv.2002.12920. arXiv:2002.12920.

[10] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, C.-J. Hsieh, Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers, 2021. doi:10.48550/arXiv.2011.13824. arXiv:2011.13824.

[11] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, J. Z. Kolter, Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification, 2021. doi:10.48550/arXiv.2103.06624. arXiv:2103.06624.

[12] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, J. Z. Kolter, General Cutting Planes for Bound-Propagation-Based Neural Network Verification, 2022. doi:10.48550/arXiv.2208.05740. arXiv:2208.05740.

[13] Z. Shi, Q. Jin, Z. Kolter, S. Jana, C.-J. Hsieh, H. Zhang, Neural Network Verification with Branch-and-Bound for General Nonlinearities, 2024. doi:10.48550/arXiv.2405.21063. arXiv:2405.21063.

[14] S. Bak, C. Liu, T. Johnson, The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results, 2021. doi:10.48550/arXiv.2109.00498. arXiv:2109.00498.

[15] M. N. Müller, C. Brix, S. Bak, C. Liu, T. T. Johnson, The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results, 2022. doi:10.48550/arXiv.2212.10376. arXiv:2212.10376.

[16] C. Brix, S. Bak, C. Liu, T. T. Johnson, The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results, 2023. doi:10.48550/arXiv.2312.16760. arXiv:2312.16760.

[17] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, C. Liu, First three years of the international verification of neural networks competition (VNN-COMP), International Journal on Software Tools for Technology Transfer 25 (2023) 329–339. doi:10.1007/s10009-023-00703-4.

[18] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, X. Yi, A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability, Computer Science Review 37 (2020) 100270. doi:10.1016/j.cosrev.2020.100270.

[19] C. Urban, A. Miné, A Review of Formal Methods applied to Machine Learning (2021). doi:10.48550/arXiv.2104.02466. arXiv:2104.02466.

[20] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer, Algorithms for Verifying Deep Neural Networks, Foundations and Trends in Optimization 4 (2021) 244–404. doi:10.1561/2400000035.

[21] A. Albarghouthi, Introduction to Neural Network Verification, 2021. doi:10.48550/arXiv.2109.10317. arXiv:2109.10317.

[22] M. Kwiatkowska, Safety verification for deep neural networks with provable guarantees, in: Leibniz International Proceedings in Informatics, LIPIcs, volume 140, 2019. doi:10.4230/lipics.concur.2019.1.

[23] T. Flinkow, B. A. Pearlmutter, R. Monahan, Comparing Differentiable Logics for Learning with Logical Constraints, 2024. doi:10.48550/arXiv.2407.03847. arXiv:2407.03847.

[24] Z. Chen, V. Badrinarayanan, C.-Y. Lee, A. Rabinovich, GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks, in: Proceedings of the 35th International Conference on Machine Learning, PMLR, 2018, pp. 794–803. URL: https://proceedings.mlr.press/v80/chen18a.html.

[25] J. Xu, Z. Zhang, T. Friedman, Y. Liang, G. V. den Broeck, A Semantic Loss Function for Deep Learning with Symbolic Knowledge, 2018. doi:10.48550/arXiv.1711.11157. arXiv:1711.11157.

[26] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, M. Vechev, DL2: Training and Querying Neural Networks with Logic, in: Proceedings of the 36th International Conference on Machine Learning, PMLR, 2019, pp. 1931–1941.

[27] E. Giunchiglia, M. C. Stoian, T. Lukasiewicz, Deep Learning with Logical Constraints, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Vienna, Austria, 2022, pp. 5478–5485. doi:10.24963/ijcai.2022/767.

[28] Z. Li, Z. Liu, Y. Yao, J. Xu, T. Chen, X. Ma, J. Lü, Learning with Logical Constraints but without Shortcut Satisfaction, in: The Eleventh International Conference on Learning Representations, 2022.
[29] H. He, W. Dai, M. Li, Reduced Implication-bias Logic Loss for Neuro-Symbolic Learning, 2023. doi:10.48550/arXiv.2208.06838. arXiv:2208.06838.
[30] M. Stoian, E. Giunchiglia, T. Lukasiewicz, Exploiting T-norms for Deep Learning in Autonomous Driving, in: CEUR Workshop Proceedings, volume 3432, 2023, pp. 369–380.
[31] T. Li, V. Srikumar, Augmenting Neural Networks with First-order Logic, in: A. Korhonen, D. Traum, L. Màrquez (Eds.), Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 292–302. doi:10.18653/v1/P19-1028.
[32] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. De Raedt, DeepProbLog: Neural Probabilistic Logic Programming, in: Advances in Neural Information Processing Systems, volume 31, Curran Associates, Inc., 2018.
[33] L. Serafini, A. d'Avila Garcez, Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge, 2016. doi:10.48550/arXiv.1606.04422. arXiv:1606.04422.
[34] S. Badreddine, A. d'Avila Garcez, L. Serafini, M. Spranger, Logic Tensor Networks, Artificial Intelligence 303 (2022) 103649. doi:10.1016/j.artint.2021.103649. arXiv:2012.13635.
[35] N. Hoernle, R. M. Karampatsis, V. Belle, K. Gal, MultiplexNet: Towards Fully Satisfied Logical Constraints in Neural Networks, Proceedings of the AAAI Conference on Artificial Intelligence 36 (2022) 5700–5709. doi:10.1609/aaai.v36i5.20512.
[36] E. Giunchiglia, T. Lukasiewicz, Multi-Label Classification Neural Networks with Hard Logical Constraints, Journal of Artificial Intelligence Research 72 (2021) 759–818. doi:10.1613/jair.1.12850.
[37] E. Giunchiglia, A. Tatomir, M. C. Stoian, T. Lukasiewicz, CCN+: A neuro-symbolic framework for deep learning with requirements, Int. J. Approx. Reasoning 171 (2024). doi:10.1016/j.ijar.2024.109124.
[38] M. M. Grespan, A. Gupta, V. Srikumar, Evaluating Relaxations of Logic for Neural Networks: A Comprehensive Study, 2021. doi:10.48550/arXiv.2107.13646. arXiv:2107.13646.
[39] E. van Krieken, E. Acar, F. van Harmelen, Analyzing Differentiable Fuzzy Logic Operators, Artificial Intelligence 302 (2022) 103602. doi:10.1016/j.artint.2021.103602. arXiv:2002.06100.
[40] N. Ślusarz, E. Komendantskaya, M. Daggitt, R. Stewart, K. Stark, Logic of Differentiable Logics: Towards a Uniform Semantics of DL, in: EPiC Series in Computing, volume 94, EasyChair, 2023, pp. 473–493. doi:10.29007/c1nt.
[41] R. Affeldt, A. Bruni, E. Komendantskaya, N. Ślusarz, K. Stark, Taming Differentiable Logics with Coq Formalisation, 2024. doi:10.48550/arXiv.2403.13700. arXiv:2403.13700.
[42] P. Varnai, D. V. Dimarogonas, On Robustness Metrics for Learning STL Tasks, in: 2020 American Control Conference (ACC), 2020, pp. 5394–5399. doi:10.23919/ACC45564.2020.9147692.
[43] T. Finkow, GitHub repository: tflinkow/comparing-differentiable-logics, 2024. URL: https://github.com/tflinkow/comparing-differentiable-logics.
[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.
[45] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards Deep Learning Models Resistant to Adversarial Attacks, 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.
[46] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, A. Madry, Robustness May Be at Odds with Accuracy, 2018. URL: https://openreview.net/forum?id=SyxAb30cY7.
[47] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, The German Traffic Sign Recognition Benchmark: A multi-class classification competition, in: The 2011 International Joint Conference on Neural

Networks, IEEE, San Jose, CA, USA, 2011, pp. 1453–1460. doi:10.1109/IJCNN.2011.6033395.
[48] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, X. Yue, Formal Specification for Deep Neural Networks, in: S. K. Lahiri, C. Wang (Eds.), Automated Technology for Verification and Analysis, volume 11138, Springer International Publishing, Cham, 2018, pp. 20–34. doi:10.1007/978-3-030-01090-4_2.
[49] M. Leucker, Formal Verification of Neural Networks?, in: G. Carvalho, V. Stolz (Eds.), Formal Methods: Foundations and Applications, Lecture Notes in Computer Science, Springer International Publishing, 2020, pp. 3–7. doi:10.1007/978-3-030-63882-5_1.
[50] M. Farrell, A. Mavridou, J. Schumann, Exploring Requirements for Software that Learns: A Research Preview, in: A. Ferrari, B. Penzenstadler (Eds.), Requirements Engineering: Foundation for Software Quality, Lecture Notes in Computer Science, Springer Nature Switzerland, 2023, pp. 179–188. doi:10.1007/978-3-031-29786-1_12.
[51] K. D. Julian, M. J. Kochenderfer, M. P. Owen, Deep Neural Network Compression for Aircraft Collision Avoidance Systems, Journal of Guidance, Control, and Dynamics 42 (2019) 598–608. doi:10.2514/1.g003724. arXiv:1810.04240.
[52] M. P. Owen, A. Panken, R. Moss, L. Alvarez, C. Leeper, ACAS Xu: Integrated Collision Avoidance and Detect and Avoid Capability for UAS, in: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), 2019, pp. 1–10. doi:10.1109/dasc43569.2019.9081758.
[53] E. Giunchiglia, M. C. Stoian, S. Khan, F. Cuzzolin, T. Lukasiewicz, ROAD-R: The autonomous driving dataset with logical requirements, Machine Learning 112 (2023) 3261–3291. doi:10.1007/s10994-023-06322-z.
[54] M. Casadio, E. Komendantskaya, M. L. Daggitt, W. Kokke, G. Katz, G. Amir, I. Refaeli, Neural Network Robustness as a Verification Property: A Principled Case Study, in: S. Shoham, Y. Vizel (Eds.), Computer Aided Verification, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 219–231. doi:10.1007/978-3-031-13185-1_11.
[55] Y. Xie, F. Zhou, H. Soh, Embedding Symbolic Temporal Knowledge into Deep Sequential Models, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 4267–4273. doi:10.1109/ICRA48506.2021.9561952.
[56] Z. Xu, Y. S. Rawat, Y. Wong, M. Kankanhalli, M. Shah, Don't Pour Cereal into Coffee: Differentiable Temporal Logic for Temporal Action Segmentation, in: Advances in Neural Information Processing Systems, 2022.
[57] D. Li, M. Cai, C.-I. Vasile, R. Tron, Learning Signal Temporal Logic through Neural Network for Interpretable Classification, in: 2023 American Control Conference (ACC), 2023, pp. 1907–1914. doi:10.23919/ACC55779.2023.10156357.
[58] E. Wong, Z. Kolter, Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope, in: Proceedings of the 35th International Conference on Machine Learning, PMLR, 2018, pp. 5286–5295. URL: https://proceedings.mlr.press/v80/wong18a.html.
[59] E. Wong, F. Schmidt, J. H. Metzen, J. Z. Kolter, Scaling provable adversarial defenses, in: Advances in Neural Information Processing Systems, volume 31, Curran Associates, Inc., 2018. URL: https://papers.nips.cc/paper_files/paper/2018/hash/358f9e7be09177c17d0d17ff73584307-Abstract.html.
[60] M. Mirman, T. Gehr, M. Vechev, Differentiable Abstract Interpretation for Provably Robust Neural Networks, in: Proceedings of the 35th International Conference on Machine Learning, PMLR, 2018, pp. 3578–3586. URL: https://proceedings.mlr.press/v80/mirman18b.html.
[61] A. Raghunathan, J. Steinhardt, P. Liang, Certified Defenses against Adversarial Examples, 2020. doi:10.48550/arXiv.1801.09344. arXiv:1801.09344.

# Towards Logical Specification and Checking of Evasive Malware

Andrei Mogage[1,2], Dorel Lucanu[1,*]

[1]Alexandru Ioan Cuza University, Iași, Romania
[2]Bitdefender, Iași, Romania

**Abstract**

The thesis proposes a new approach of combining formal methods and malware analysis for quickly determining if an application has specific malicious capabilities. The proposed solution is a Formal Tainting-Based Framework that uses a combination of binary instrumentation, taint analysis, and temporal logic in order to selectively extract behavioral properties of a malware. These are then formalized in order to check if the application expresses certain capabilities. The findings are accompanied by a concrete implementation, which proved effective and efficient against real-life malware, as highlighted by an evaluation. Furthermore, the framework has been used in actual cyber forensics investigations, reducing the time and efforts of security researchers. In this paper we also investigate how the framework can be applied in the context of evasive malware, allowing us to bypass anti-analysis techniques in order to reach the malicious core of the malware. This feature has not been previously covered by our other papers.

**Keywords**

temporal logic, taint analysis, malware analysis, formal methods

## 1. Introduction

### 1.1. Context and Challenges

The purpose of this thesis is to combine knowledge from academia and expertise from the cyber-security industry to enhance a critical task: malware analysis. This can be a complex process considering that highly complex and evasive malware have been on an increasing trend for the last few years [1, 2, 3].

However, a rapid verification of a potential capability is more desired, in situations where quick triage is necessary. For instance, a more useful and meaningful solution is to automatically get a verdict/hint of what the program is capable of, e.g.: *Does it steal data? Is it a ransomware?* Moreover, it is very helpful *to know the arguments that led to that verdict.* Knowing which semantic correlations have been made in order to reach a verdict might allow us to determine the correctness of the assessment. For instance, we want to know which chain of related events has led to the verdict that the program has the capability of stealing data. This is in contrast with the AI-based approaches, e.g. machine learning, where such correlations are merely statistical.

There are some problems that might interfere with this process:

- *Obfuscation* - This limits the usability of static analysis or even some dynamic analysis tools.
- *Anti-analysis techniques* - This limits or prevents the analysis, regardless of the analysis type, depending on the techniques implemented.
- *Traceability* - While an analysis might capture the called APIs, such as a sandbox, there is no direct correlation between passing the output of an API to the input of another. This happens because the individual instructions are usually not traced, due to performance reasons (i.e., minimizing overhead).

*PhD Symposium of the 19th International Conference on Integrated Formal Methods (iFM)*
*at the University of Manchester, UK, 12 November 2024.*
*PhD Supervisor
andrei.mogage@gmail.com (A. Mogage)
0000-0002-3533-7573 (A. Mogage); 0000-0001-8097-040X (D. Lucanu)

The issue has also been the subject of extended research, both through academic [4, 5] and industry contexts [6].

## 1.2. Contribution

Our contribution consists in a Formal Tainting-Based Framework (FTBF), which uses binary instrumentation to control an application, taint analysis to extract relevant behavioral properties into a trace, a new temporal logic to analyze the tainted trace, Tainting-Based Logic (TBL), and a formally described behavior (capability) that the user wishes to check. While taint analysis is more commonly used to detect vulnerabilities or input reaching certain *sinks*, we use it to extract complex behavioral properties, that are later formally checked against malicious capabilities. Even though we only include the highlights in this paper, the entire framework is completely defined in [7] (submitted and accepted at iFM 2024). To our best knowledge, this is the first time when dynamic instrumentation, taint analysis, and temporal-logic-based checking are integrated into a formal framework for malware analysis. The result is a sound confirmation of whether the application has a specific capability, ensured through the formal checking process. The formalization component ensures that the results are sound, providing accurate details, rather than *educated guesses* (e.g., based on naive static heuristics from sandbox reports). Nonetheless, the system is built in such a fashion that it requires no knowledge of formal methods for a user to be able to use it, but rather a basic understanding of how some capabilities can be implemented at low-level.

In order to provide a concrete example of an ideal solution, we may refer to the task of an analyst whose purpose is to determine whether the analyzed application possesses certain traits or exerts some interesting behavior:

- *Is it a ransomware?* - i.e., encrypts files and asks for a ransom in return;
- *Does it exfiltrate data?* - i.e., transfers sensitive data without authorization;
- *Does it deploy techniques to deter security solutions?* - i.e., attempts to disrupt the activity of a security solution in order to ensure the success of other malicious activities;
- *Is it an Active Persistent Threat (APT)?* - i.e., ensures the persistence over system reboot (or even reinstall), keeps a low profile (hard to detect its resources - files, registries, processes), contacts suspicious servers for further commands.

Assuming that the analysis tool can deter the anti-analysis techniques, thus managing to avoid execution refusal, our extension will firstly taint sensitive data (API output, registers, memory) and monitor its propagation towards the end of the execution. The tainting events(i.e., tainting and tracking) will be formalized, by introducing predicates that express behavioral properties. Finally, we may devise certain formulas that make use of temporal operators in order to check whether the analyzed application, through its formalized output, has certain patterns of behavioral capabilities. This, of course, implies a minimal knowledge of how certain attacks can be implemented, but the entire flow is effortless and extendable. Furthermore, the entire process of formally defining rules can be simplified by using artificial intelligence for obtaining hints regarding capabilities. The system is also accompanied by a concrete implementation (for the Windows OS) and an evaluation process that proves its benefits. The implementation has actually been integrated into two Dynamic Binary Instrumentation (DBI) tools, Intel Pin [8] and COBAI [9]. The framework is sustained by experiments using actual malware families, evaluating its efficiency in capturing various capabilities (e.g. code injection, encryption, deobfuscation, privilege escalation). Moreover, the system has been used in real-life scenarios, during cyber forensics, proving useful for speeding up the process of analysis and filtering data. Another important aspect is providing the capability extraction features while handling anti-analysis techniques implemented by evasive malware. This aspect has not been included in [7], but is discussed in Section 3.3, where we combine FTBF with COBAI to achieve this.

In the following, we intuitively describe the formal framework in Section 2, along with necessary elements for applying it towards malware analysis. We summarize our evaluation results against malware families and discuss the transparency features in Section 3, and we conclude in Section 4.

## 2. A Formal Tainting-Based Framework for Malware Analysis

The Formal Tainting-Based Framework (FTBF) uses Dynamic Binary Instrumentation (DBI) to instrument the target application, while informing the taint analysis component of various events that occur during execution (instrumented instructions, memory updates, etc.). According to a taint policy provided as input, some data will be tainted and monitored throughout the execution. All tainting-related events are then formalized using Tainting-Based Logic (TBL) and placed into a formal tainted trace. Lastly, this trace is checked against *rules that specify capabilities* (specified in TBL), which ultimately provides a verdict whether the application has the capability described by the rule.

The syntax of TBL consists of three main categories of sentences:

1. *behavioral patterns*, whose instances describe behavioral properties of executions;
2. *potential/capability patterns*, whose instances describe capabilities of the executions;
3. *rules*, which relates behavioral properties and capabilities of an execution.

An example of a behavioral pattern is:

$$(\mathit{TaintedAPI}(\texttt{CreateFile}) \wedge \mathsf{X}(\mathit{Tainted}(T))) \\ \text{andthen } \mathit{PropToAPI}(\texttt{ReadFile}, T)$$

which, intuitively, describes that in the current state a file is created or accessed the first time, and it is read at some later state. The parameter $T$ is the tainting tag used to track the file handle. Despite its name, `CreateFile` is also used to open existing files. andthen is part of TBL and newly introduced, and describes the fact that a property $\phi_2$ occurs at a certain time in the future after another one $\phi_1$. X is borrowed from Temporal Logics and refers to the next state.

An example of a capability pattern is:

$$\texttt{FileRead}(\mathrm{T})$$

where the predicate describes the capability of a program to read the contents of a file, where the $T$ variable will be instantiated by a behavioral pattern (see the definition for capability rules below).

A simple example of rule is

$$(\mathit{TaintedAPI}(\texttt{CreateFile}) \wedge \mathsf{X}(\mathit{Tainted}(T))) \\ \text{andthen } \mathit{PropToAPI}(\texttt{ReadFile}, T) \\ \vdash \\ \texttt{FileRead}(T)$$

which helps us to deduce that our program has the capability of reading the contents of a file. The behavioral pattern is the same introduced earlier, where we expect a propagation of the taint symbol, $T$ (and its associated tainted data), from the `CreateFile` API to `ReadFile`. $\vdash$ is an entailment relation, i.e., $A \vdash B$ means *if A, then B*.

FTBF has been introduced as a general framework for checking program capabilities. Our main goal, however, is to channel its usage towards malware analysis. Specifically, to apply FTBF in those scenarios where a specific malware capability (such as keylogging, encryption, or data ex-filtration) needs to be quickly detected.

The requirements for doing so are related to taint policies and capability rules. We need to define policies that capture factual behaviors related to malicious activities, along with rules that define a behavioral pattern encoding a malicious capability. In doing so, we use the following sentences:

$$\begin{aligned} \texttt{Behavioral Facts} : \{&\mathit{Tainted}, \mathit{Untainted}, \mathit{TaintedAPI}, \mathit{TaintedAPICond}, \mathit{PropToAPI}, \\ &\mathit{PropToAPICond}, \mathit{PropToReg}, \mathit{UntaintedReg}, \mathit{PropToMem}, \\ &\mathit{UntaintedMem}, \mathit{TaintedMemAccess}, \mathit{TaintedCodeExecuted}\}. \\ \texttt{Capabilities} : \{&\mathit{DisableWindowsEvent}, \mathit{C2Communication}, \mathit{RansomwareBehavior}, \\ &\mathit{DebuggerDetection}, \mathit{DisableUAC}, \mathit{PayloadDeobfuscation}, \end{aligned}$$

*CodeInjection*, *PrivilegeEscalation*, *SGXSupplyChainAttack*}

`Constants : {CreateFile, ReadFile, VirtualAlloc, …}` ∪

`{ eax, rbx, esi,…}`

The first category of constants refers to Windows APIs, whereas the latter describes CPU registers.

## 3. Evaluation

In this section, we present the potential of the analysis framework by testing it against a suite of malicious applications.

### 3.1. Method

We have selected eight malware families and created one rule for each family, such that we will capture different capabilities, depending on the target application. The selection process has been random, but with a focus on selecting malware families that are still relevant (i.e., seen in recent attacks or that represented a strong inspiration for subsequent cyber threats).

The number of samples for each family varies between 6 and 37, depending on their public availability. All applications have been collected from two publicly available sources: VirusTotal [10] and Malpedia [11, 12]. Each malware family is briefly described below:

- *Al-Khaser* [13]: An application which deploys a myriad of malicious techniques, borrowed from complex malware, in order for analysts to test analysis solutions. While not an actual malware (the only one in this situation in our test suite), it is relevant for capturing anti-analysis capabilities;
- *Avaddon* [14]: A ransomware family initially seen in 2020, with a strong RaaS (Ransomware-as-a-Service) model, which affected a high number of victims spanning multiple industries and countries. Even if the RaaS has been shutdown in 2021, the malware is still being distributed worldwide and several new ransomware families emerged later on and share common practices and even source code [15];
- *RokRat* [16]: A Remote Administration Tool (RAT), first reported in 2017, RokRat is a malware distributed in multiple malicious campaigns and makes use of numerous attack vectors and techniques in order to infiltrate a victim and then start an entire chain of receiving and executing commands. While it has not suffered significant changes at its core lately, it has been combined with new techniques to ensure its success in affecting victims [17];
- *Darkside* [18]: A ransomware family renowned for high-impact attacks targeting large corporations and governments, culminating with the attack on the Colonial Pipeline in the United States;
- *CobaltStrike Beacon* [19]: A beacon is a payload from Cobalt Strike [20] which mimics attacker activities seen in the wild. However, beacons have been used directly in actual attacks as an initial step for deploying other malware tools on the systems of affected victims;
- *Phant0m* [21]: A tool capable of tampering with the Event Logging Service integrated into the Windows OS, resulting in the operating system not logging critical events that occur. This leads to attacks that are more stealthy and harder to investigate;
- *HermeticWiper* [22]: A malware involved in cyber-attacks targeting Ukraine, emerged during the recent war, which disrupts the functionality of systems;
- *Makop* [23]: An infamous ransomware family and group, which has been active for at least the last 4 years.

For each malware category, we have verified one malicious capability:

- *Al-Khaser* - Debugger detection;

- *Avaddon* - Disable User Account Control (UAC);
- *RokRat* - Code Injection;
- *Darkside* - C2 Communication;
- *CobaltStrike Beacon* - Deobfuscation;
- *Phant0m* - Disable Windows Event Logging;
- *HermeticWiper* - Privilege escalation;
- *Makop* - Ransomware Behavior.

All experiments have been conducted inside a virtual machine (using VMWare Workstation 16), with a Windows 10 as guest with 8 GB of RAM, 4 cores and virtualization enabled, while the host uses Windows 11 on an Intel i7-11800H CPU @2.30GHz and 32 GB of RAM.

### 3.2. Results

For simplicity, we have summarized the results for each test in Table 1. The tables contain the name of the malware family, along with the duration (in seconds) necessary to analyze and capture the capabilities described by the rules and the duration of a native execution. All experiments were successful (i.e. the capability has been correctly identified). Moreover, each analysis instance was achieved under 30 seconds, except for a sample for Makop ransomware, where some anti-analysis tricks were deployed, which slowed the analysis process.

**Table 1**
Summary of results

| Malware Family | Avg. Analysis (s.) | Avg. Native execution (s.) |
|---|---|---|
| Al-Khaser | 7.73 | TIMEOUT |
| Avaddon | 14.375 | TIMEOUT |
| RokRat | 22.69 | 19.72 |
| DarkSide | 15.87 | 14.68 |
| CobaltStrike | 4.13 | TIMEOUT |
| Phant0m | 6.22 | 1.22 |
| HermeticWiper | 9 | TIMEOUT |
| Makop | 20.42 | TIMEOUT |

These experiments bring forward the potential of such an analysis solution to quickly identify if a program expresses a specific capability or not. Naturally, the analysis process will bring an additional overhead compared with the program executing outside an execution environment. Because of this, we have also tested the duration under native execution. However, while the analysis time represents only duration until the capability is expressed, we could not verify at which exact moment the same capability is expressed natively without tampering with the applications. Therefore, we have chosen to estimate the total time necessary to complete the execution, with a provided timeout of **10 minutes**. This leads to three scenarios of comparison:

- **TIMEOUT** on native execution - this was caused by applications that either take longer than 10 minutes to complete their execution, or continue executing endlessly, e.g. waiting for network commands or scanning possibly new files. This category only applies to native execution, while all analysis instances ended successfully.
- Native duration is **longer** than analysis duration - this is directly related to the fact that the execution of the analyzed program is terminated as soon as the capability has been expressed, as an optimization technique.
- Native duration is **shorter** than analysis duration - this represents the overhead of the analysis solution.

**Table 2**
Evaluation scenarios per malware family

| Malware Family | TIMEOUT | LONGER NATIVE | LONGER ANALYSIS |
|---|---|---|---|
| Al-Khaser | 15 | 0 | 0 |
| Avaddon | 8 | 0 | 0 |
| RokRat | 1 | 1 | 24 |
| DarkSide | 0 | 4 | 12 |
| CobaltStrike | 37 | 0 | 0 |
| Phant0m | 0 | 22 | 0 |
| HermeticWiper | 6 | 0 | 0 |
| Makop | 19 | 0 | 0 |

We have summarized how many of samples per malware family fall under these three scenarios in Table 2.

Nonetheless, an actual real-life comparison is not with a native execution, but rather with the effort, especially regarding the time consumed, of an analyst. Another important remark relates to the generality of these rules, i.e. how well they can capture a capability. This is directly influenced by how well a rule is constructed and if it considers multiple scenarios, because there might be multiple ways of achieving an expecting results, e.g. by combining different APIs or instructions.

In this regard, a prerequisite for a user of this analysis solution is to have some general information on such combinations that might lead to an expected output. Nonetheless, the effort required to construct a rule, along with the analysis duration, are significantly lower than manually analyzing an application or by filtering and correlating a myriad of results provided by other automated solutions, such as a sandbox. Furthermore, the process of crafting rules for capturing capabilities can be simplified by using Large Language Models (LLMs), as presented in [24]. The user can query a LLM for an overall idea of how capabilities can be implemented, and then proceed with the formalization process.

### 3.3. Transparency

Although we have chosen Pin as the DBI component, we have also experimented with COBAI [9] as well, due to its main focus on transparency, highly needed for evasive applications. This aspect has not been previously discussed in [7], here is the first time we discuss the transparency of FTBF.

An analysis environment is considered ”transparent” if a malware is unable to detect its presence. Usually, these attacks result in execution refusal or code executing outside the analysis environment. Despite some reports stating that DBI might be unsuitable against evasive malware (e.g., [25]), our empiric evaluation against malware or personally developed applications that deploy anti-analysis techniques reveal that COBAI can be properly used against this category of malware. This also takes into consideration various aspects of implementation details, Windows OS architecture, and malware analysis in general. More details about this have been included in [9]. In order to ensure transparency for both itself and the rest of the environment, COBAI deploys a series of heuristics based on anti-analysis techniques seen in malware, but also in analysis-testing frameworks (such as Al-Khaser [13] or Pafish [26]), or in academic literature [27, 28, 29]. We exemplify some of these heuristics below, where we mention how a malware would try to detect the analysis process:

- *System Artifacts*: Querying the presence of certain sandbox or VM-specific artifacts, such as files (i.e., hypervisor services), registries (i.e., sandbox configuration), network resources (i.e., IP or MAC addresses), processes (i.e., those related with the virtual machine or sandbox), or hardware properties (i.e., CPU virtualization flags);
- *Debuggers*: Scanning for a list of consecrated debugger names based on the list of active processes, files, or the currently loaded modules;
- *DBIs*: Scanning the memory pages, threads, loaded modules, and other resources in order to detect any possible anomaly;

- *Time-based Techniques*: Using time for either measuring the time between certain commands (which would be increased during the analysis, depending on the overhead), or verifying if the time has been somehow accelerated (i.e., skipping `sleep` commands);
- *Human Interaction*: Verifying that certain events occur, that would be normally expected under a legitimate scenario, such as cursor movement, mouse clicks, or keystrokes;

In order to counter-attack these techniques, COBAI will monitor specific instructions or APIs and provide fake results when the program attempts at querying certain artifacts that would reveal the analysis framework or environment. During some experiments with evasive malware, the results reveal a good combination of COBAI and FTBF: COBAI will ensure transparency, in order for FTBF to be able to capture the malicious capabilities that follow.

## 4. Conclusion

The Formal Tainting-Based Framework can be a powerful formalism and tool for quick and safe verification of malware capabilities. Having minimal knowledge on how capabilities can be implemented, a user can use taint policies and capability rules to determine if a malware express a specific behavior.

The entire process uses binary instrumentation, taint analysis, and the proposed Tainting-Based Logic, efforts that combine industrial efforts and academic knowledge, leading to sound and precise results. Our claims are sustained by evaluation results using multiple malware families, with a rule for each distinct capability. The results hint at significantly lowering the burden of a cyber researcher. This fact is also highlighted through real-life use-cases, where we used the current implementation and rule set to determine which malware samples might have a particular capability. Furthermore, we also tackled the case of evasive malware, where one must first counter-attack the anti-analysis techniques in order to reach the actual payload of a malware. This case has been handled by using COBAI as the DBI component, which ensures transparency of the framework and the rest of the environment.

## References


[1] J. Coker, Evasive malware threats on the rise despite decline in overall attacks, 2020. URL: https://www.infosecurity-magazine.com/news/evasive-malware-rise-decline/.

[2] W. T. Inc, New research: Fileless malware attacks surge by 900% and cryptominers make a comeback, while ransomware attacks decline, 2021. URL: https://www.globenewswire.com/en/news-release/2021/03/30/2201173/0/en/New-Research-Fileless-Malware-Attacks-Surge-by-900-and-Cryptominers-Make-a-Comeback-While-Ransomware-Attacks-Decline.html.

[3] H. N. Security, The hidden picture of malware attack trends, 2023. URL: https://www.helpnetsecurity.com/2023/04/06/malware-attack-trends-q4-2022/.

[4] A. Afianian, S. Niksefat, B. Sadeghiyan, D. Baptiste, Malware dynamic analysis evasion techniques: A survey, ACM Comput. Surv. 52 (2019). URL: https://doi.org/10.1145/3365001.

[5] F. A. Aboaoja, A. Zainal, A. M. Ali, F. A. Ghaleb, F. J. Alsolami, M. A. Rassam, Dynamic extraction of initial behavior for evasive malware detection, Mathematics 11 (2023). URL: https://www.mdpi.com/2227-7390/11/2/416. doi:10.3390/math11020416.

[6] Defeating evasive malware whitepaper, 2023. URL: https://www.vmray.com/resources/defeating-evasive-malware-whitepaper/.

[7] A. Mogage, D. Lucanu, A formal tainting-based framework for malware analysis, Springer Lecture Notes in Computer Science (2024). The 19th International Conference on Integrated Formal Methods (iFM).

[8] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, K. Hazelwood, Pin: Building customized program analysis tools with dynamic instrumentation, in: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 190–200. URL: https://doi.org/10.1145/1065010.1065034.

[9] V. Craciun, A. Mogage, D. Lucanu, Full transparency in DBI frameworks, 2023. URL: https://doi.org/10.48550/arXiv.2306.13529. arXiv:2306.13529.
[10] Virustotal: How it works, 2023. URL: https://docs.virustotal.com/docs/how-it-works.
[11] D. Plohmann, M. Clauss, S. Enders, E. Padilla, Malpedia: A Collaborative Effort to Inventorize the Malware Landscape, The Journal on Cybercrime & Digital Investigations 3 (2018). URL: https://journal.cecyf.fr/ojs/index.php/cybin/article/view/17.
[12] Malpedia - usage, 2023. URL: https://malpedia.caad.fkie.fraunhofer.de/usage/references.
[13] Noteworthy, LordNoteworthy/al-khaser, 2024. URL: https://github.com/LordNoteworthy/al-khaser, original-date: 2015-11-12T18:35:16Z.
[14] J. Yuste, S. Pastrana, Avaddon ransomware: An in-depth analysis and decryption of infected systems, Computers & Security 109 (2021) 102388. URL: https://www.sciencedirect.com/science/article/pii/S0167404821002121.
[15] One source to rule them all: Chasing AVADDON ransomware | mandiant, 2023. URL: https://cloud.google.com/blog/topics/threat-intelligence/chasing-avaddon-ransomware.
[16] Introducing ROKRAT, 2017. URL: https://blog.talosintelligence.com/introducing-rokrat/.
[17] etal, Chain reaction: ROKRAT's missing link, 2023. URL: https://research.checkpoint.com/2023/chain-reaction-rokrats-missing-link/.
[18] DarkSide ransomware explained: How it works and who is behind it, 2023. URL: https://www.csoonline.com/article/570723/darkside-ransomware-explained-how-it-works-and-who-is-behind-it.html.
[19] MAR 10339794-1.v1 - cobalt strike beacon | CISA, 2021. URL: https://www.cisa.gov/news-events/analysis-reports/ar21-148a.
[20] Cobalt strike beacon | cobalt strike features, 2024. URL: https://www.cobaltstrike.com/product/features/beacon.
[21] H. Dalabasmaz, Phant0m (github), 2023. URL: https://github.com/hlldz/Phant0m.
[22] HermeticWiper: A detailed analysis of the destructive malware that targeted Ukraine, 2022. URL: https://www.threatdown.com/blog/hermeticwiper-a-detailed-analysis-of-the-destructive-malware-that-targeted-ukraine/.
[23] P. Paganini, Dissecting the malicious arsenal of the Makop ransomware gang, 2023. URL: https://securityaffairs.com/143452/malware/dissecting-makop-ransomware.html.
[24] A. Mogage, A.I. Assisted Malware Capabilities Capturing, Procedia Computer Science (2024). In the proceedings of the 28th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2024).
[25] J. Kirsch, Z. Zhechev, B. Bierbaumer, T. Kittel, PwIN - pwning intel piN: Why DBI is unsuitable for security applications, in: J. Lopez, J. Zhou, M. Soriano (Eds.), Computer Security, Springer International Publishing, 2018, pp. 363–382. doi:10.1007/978-3-319-99073-6_18.
[26] A. Ortega, a0rtega/pafish, 2024. URL: https://github.com/a0rtega/pafish, original-date: 2012-07-01T11:06:40Z.
[27] A. S. Filho, R. J. Rodríguez, E. L. Feitosa, Evasion and countermeasures techniques to detect dynamic binary instrumentation frameworks 3 (2022) 11:1–11:28. URL: https://dl.acm.org/doi/10.1145/3480463. doi:10.1145/3480463.
[28] D. C. D'Elia, E. Coppa, S. Nicchi, F. Palmaro, L. Cavallaro, Sok: Using dynamic binary instrumentation for security (and how you may get caught red handed), in: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, 2019, pp. 15–27.
[29] D. C. D'Elia, E. Coppa, F. Palmaro, L. Cavallaro, On the dissection of evasive malware, IEEE Transactions on Information Forensics and Security 15 (2020) 2750–2765. doi:10.1109/TIFS.2020.2976559.

# Isomorphic Transfer Infrastructure for Nested Types in Isabelle/HOL (Work in Progress)

Gergely Buday[1], Andrei Popescu[1]

[1]School of Computer Science, University of Sheffield, Sheffield, UK

**Abstract**

This paper addresses (part of) the problem of simplifying reasoning with proof assistants by transferring theorems that are stated in a heavy form, using explicit invariants, to lightweight counterparts where the invariants are handled implicitly by the type system. Specifically, we provide some abstract assumptions that allow one to establish isomorphisms for nested applications of defined types in Gordon's Higher-Order Logic (HOL). This allows the seamless isomorphic transfer of results across type definitions in the presence of nesting. Our results have been formalized in the Isabelle/HOL theorem prover, and we plan to integrate them with Isabelle's Lifting and Transfer tool.

**Keywords**

Higher-Order Logic (HOL), type definitions, theorem proving, Isabelle/HOL

## 1. Introduction

The definition of new types by carving out subsets of existing types, known as *typedef*, is a fundamental mechanism in Higher-Order Logic (HOL), a logic that provides the foundation for several interactive theorem provers, including HOL4 [1], HOL Light [2] and Isabelle/HOL [3]. While most of the uses of typedef are hidden under the (automated) definition of (co)inductive datatypes [4, 5, 6, 7] and therefore not directly invoked by the users, there still remains an important scenario where typedef is invoked explicitly.

Namely, say one has a type (perhaps an algebraic datatype) $T$ that does not capture precisely the intended concept (because it contains too many elements), but only via an *invariant* defined in terms of a predicate on $T$, or equivalently, a set $I$ that has type $T\ set$.[1] Then one defines the more precise type $S$ as a typedef, to consist of exactly the inhabitants of $T$ that belong to the set $I$—i.e., $I$ is effectively turned into a type.

Let us consider two examples, which will act as our running examples throughout this paper.

**Example 1.** ***(Distinct Lists)*** Polymorphic lists are introduced as the following datatype, where we use ML-style notation feature by Isabelle/HOL as well as all the HOL-based provers (in particular, $\alpha$ denotes a type variable):

$$\text{datatype}\ \ \alpha\ list = Nil \mid Cons\ \alpha\ (\alpha\ list)$$

For a list $xs$, we let $length\ xs$ be its length and, given a natural number $i < length$, we let $xs!i$ be the $(i-1)$'th element in $xs$ (so the indexing starts from 0). In some developments, one may be interested in working with nonrepetitive ("distinct") lists, i.e., lists whose elements do not repeat—to this end, one defines the (polymorphic) predicate $distinct : \alpha\ list \to bool$ by $distinct\ xs \equiv \forall i\ j.\ i < j \wedge j < length\ xs \longrightarrow xs!i \neq xs!j$. The new (polymorphic) type $\alpha\ dlist$ of distinct lists is defined as a typedef:

$$\text{typedef}\ \ \alpha\ dlist = \{xs : \alpha\ list \mid distinct\ xs\}$$

This command introduces the new type $\alpha\ dlist$ together with an abstraction-representation pair of (polymorphic) constants[2] $(Rep_{dlist}, Abs_{dlist})$ with $Rep_{dlist} : \alpha\ dlist \to \alpha\ list$ and $Abs_{dlist} : \alpha\ list \to \alpha\ dlist$

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

g.buday@sheffield.ac.uk (G. Buday); a.popescu@sheffield.ac.uk (A. Popescu)



[1]The type $T\ set$ of sets of elements of a type $T$ is a copy of the type $T \to bool$, where $set$ is a type operator.

[2]In HOL, items that have a fixed interpretation, including fixed values such as 0 or defined functions, are called "constants"; they are to be contrasted with "variables", which do not have a fixed interpretation but range over given types.

and the following axioms stating that these functions are mutually inverse bijections between the new type $\alpha$ *dlist* and the subset $\{xs : \alpha\ list \mid distinct\ xs\}$ of the base type $\alpha$ *list*.[3] In Isabelle/HOL, these axioms are actually packed into a single axiom:

$$type\_definition\ Rep_{dlist}\ Abs_{dlist}\ \{xs : \alpha\ list \mid distinct\ xs\}$$

where the (polymorphic) predicate $type\_definition : (\beta \to \alpha) \to (\alpha \to \beta) \to (\alpha\ set) \to bool$ is defined as follows: $type\_definition\ r\ a\ A \equiv (\forall x.\ r\ x \in A) \wedge (\forall y : \beta.\ a\ (r\ y) = y) \wedge (\forall x : \alpha.\ x \in A \longrightarrow r\ (a\ x) = x)$.

Thus, in this example $T$ is $\alpha$ *list*, $I$ is $\{xs : \alpha\ list \mid distinct\ xs\}$, and $S$ is $\alpha$ *dlist*. □

**Example 2.** ***(Discrete Distributions)*** A (discrete) distribution is a positive function to the real numbers of countable support such that its values sum up to to 1. The polymorphic type $\alpha$ *distrib* of distributions on $\alpha$ is introduced as a typedef having base type $\alpha \to real$ (the type of functions from $\alpha$ to *real* ):

$$\text{typedef}\ \ \alpha\ distrib = \{f : \alpha \to real \mid dist\ xs\}$$

where the predicate $dist : (\alpha \to real) \to bool$ is defined by $dist\ f \equiv (\forall x : \alpha.\ f\ x \geq 0) \wedge countable\ \{x : \alpha \mid f\ x \neq 0\} \wedge \sum_{x:\alpha} f\ x = 1$.

As before, the above command introduces a new type $\alpha$ *distrib* , an abstraction-representation pair of constants ($Rep_{dlist}$, $Abs_{dlist}$) with ($Rep_{distrib}$, $Abs_{distrib}$) with $Rep_{distrib} : \alpha\ distrib \to \alpha\ list$, and the axiom $type\_definition\ Rep_{distrib}\ Abs_{distrib}\ \{f : \alpha \to real \mid dist\ f\}$ saying that $Abs_{distrib}$ and $Rep_{distrib}$ are mutually inverse bijections between $\alpha$ *distrib* and $\{f : \alpha \to real \mid distrib\ xs\}$.

Thus, in this example $T$ is $\alpha \to real$, $I$ is $\{f : \alpha \to real \mid dist\ xs\}$, and $S$ is $\alpha$ *distrib*. □

In a formal development that follows the above scheme, one usually distinguishes between:

- developing the "internal" mathematical theory, which usually proceeds without defining $S$, but instead working with $T$ and stating the theorems relativized to $I$—for example, proving facts of the form $\forall t : T.\ t \in I \longrightarrow \ldots$
- at the end, defining $S$ and "sealing" the library for export by transferring from $T$ to $S$ all the constants and all the main (exportable) facts that have been proved proved relative to $I$—for example, turning facts of the form $\forall t : T.\ t \in I \longrightarrow \ldots$ into facts of the form $\forall t' : S. \ldots$

The process of "isomorphically" transferring $I$-relativized constants and results on $T$ to corresponding constants and results on $S$, while seemingly conceptually straightforward, turns out to be quite subtle in the presence of higher-order constants. It requires infrastructure for lifting relations along type constructors (known as *relators*), which allows the automated proofs of the transferred theorems from the original ones—this is facilitated in Isabelle/HOL by various dedicated tools [8, 9, 10].

In this short work-in-progress paper, we study a fairly common pattern: the isomorphic transfer in the presence of nested type constructors. We start with motivation in terms of a standard construction applied to our running examples (§2), which leads to formulating the wider scope of the problem. We then work out the solution to the problem in an ad hoc manner on the running examples §3. After that, we are ready to describe our main result: some abstract general structure and conditions that enable this pattern, in that they allow constructing a back-and-forth bijection for transfer (§4), and show how it instantiates to our examples. We conclude with related work and future plans, notably the planned integration of our work into Isabelle's Lifting and Transfer package (§5). Our concepts and results apply to Higher-Order Logic, hence are in principle relevant to any HOL-based provers. We have formalized them in the Isabelle/HOL theorem prover, and the formal scripts are publicly available [11].

[3] These are sometimes called an "embedding-projection pair"; here we prefer terminology that is closer to HOL, referring to a representation function (indicating how elements of the new type are represented/implemented in terms of those of the old one) and an opposite abstraction function.

## 2. The Concrete Problem

Consider the problem of proving that the type constructors (polymorphic types) $\alpha$ *dlist* and $\alpha$ *distrib* constitute monads [12, 13] or at least monad-like structures—which are very useful properties to have for any (data)type, whenever possible.

According to the above pattern, one wishes to first prove the properties for the underlying (representing types) $\alpha$ *list* and $\alpha \to$ *real* relative to the defining predicates *distinct* and *dist*, and then transfer them to the defined types $\alpha$ *dlist* and $\alpha$ *distrib*. (Not only is this good practice, but in some sense it is the only way to proceed, at least initially when "bootstrapping" a theory for the defined types, given that initially our only means to prove a property on the defined type is to trace it back to a property on the underlying type.)

Some of the monadic structure and properties involve nesting the application of type constructors, for example we want to have a map (functorial-action) operator $map_{distrib} : (\alpha \to \beta) \to (\alpha\ distrib \to \beta\ distrib)$ and join (counit) operator $join_{distrib} : (\alpha\ distrib)\ distrib \to \alpha\ distrib$ satisfying (among others) the associativity law $join_{distrib} \circ (map_{distrib}\ join_{distrib}) = join_{distrib} \circ join_{distrib}$.

How to define such structure and prove such properties on the defined types? Let us start with a simple example that does not involve nesting, namely defining the map operator on $(\alpha \to \beta) \to (\alpha\ distrib \to \beta\ distrib)$ and proving that it preserves identities, in that $\forall d : \alpha\ distrib.\ map_{distrib}\ (id : \alpha \to \alpha)\ d = d$ where *id* denotes the identity function. We define $map_{distrib}$ on any $g : \alpha \to \beta$ and $d : \alpha\ distrib$ by $map_{distrib}\ g\ d = Abs_{distrib}\ (\lambda b : \beta.\ \sum_{a \in g^{-1}\ b} Rep_{distrib}\ d\ a)$. Notice that the definition requires a back-and-forth application of the abstraction and representation functions $Abs_{distrib}$ and $Rep_{distrib}$, with some specific manipulation of items of the underlying type $\alpha \to real$. (In this case, the specific manipulation happens to involve taking the sum of a function in $\alpha \to real$ over all the elements of the $g$-preimage of $g$, but the exact nature of such manipulations is not important here.) Now, to prove the desired fact, fix $d : \alpha\ distrib$. In order to show $map_{distrib}\ id\ d = d$, by the injectivity of $Rep_{distrib}$ it suffixes to show $Rep_{distrib}\ (map_{distrib}\ id\ d) = Rep_{distrib}\ d$. Using the definition of $map_{distrib}$ and the fact that $Rep_{distrib}$ is left-inverse to $Abs_{distrib}$, the above is equivalent to $\lambda b : \beta.\ \sum_{a \in id^{-1}\ b} Rep_{distrib}\ d\ a = Rep_{distrib}\ d$, i.e., to $\lambda b : \beta.\ \sum_{a=b} Rep_{distrib}\ d\ a = Rep_{distrib}\ d$, which follows from the properties of sums and function extensionality.

We thus have the following pattern: *To define constants on the defined type and prove properties for them, we need to move back and forth via the abstraction-representation pair and use consequences of the associated type_definition axiom.*

But how to do this in the presence of nested defined type constructors (where the abstraction-representation pairs stemming from type definitions no longer work out of the box)? In the next section, we discuss in an ad hoc manner how to tackle the nested isomorphic transfer problem in the presence of nested types for our two running examples. After that, we will introduce an abstract solution, which covers these two cases and many others.

## 3. The Solution for Two Concrete Instances

Now let us come back to the original more complex problem, of defining $join_{distrib} : (\alpha\ distrib)\ distrib \to \alpha\ distrib$ (in addition to $map_{distrib}$ which we have already defined) and proving the associativity law.

In order to define $join_{distrib}$, which operates on the nested defined type $(\alpha\ distrib)\ distrib$, we need an understanding of how a counterpart of this operator should act on the (nested application of) the underlying type, i.e., on $(\alpha \to real)real$. To be more exact, we don't need to consider the behavior of such a counterpart on all functions $F : (\alpha \to real) \to real$, but seemingly only on functions that act like distributions, i.e., satisfy $distrib\ F$ (i.e., are positive, have countable support and sum to 1). In fact, upon a closer look we see that the functions of interest are not really distributions on the entire type $\alpha \to real$, but on the subset of $\alpha \to real$ that consists of distributions only, i.e., distributions on the set $\{f : \alpha \to real \mid distrib\ f\}$. In other words, we need a relativized version of the predicate *distrib*, let us

denote it by $distOn : \alpha\ set \to \alpha\ distrib \to bool$, defined by

$$distOn\ A\ f \equiv (\forall x : \alpha.\ x \in A \longrightarrow f\ x \geq 0) \wedge countable\ \{x : \alpha \mid x \in A \wedge f\ x \neq 0\} \wedge \sum_{x \in A} f\ x = 1$$

where we have highlighted the difference from the original predicate *dist*—in that the conditions are not applied to the entire type $\alpha$, but to a parameter subset $A : \alpha\ set$. Note that we can recover the original predicate as $dist = distOn\ UNIV$, where *UNIV* is the "universal" set covering the entire type.

With this relativized predicate at hand, the picture becomes clear: Given a function $F : (\alpha \to real) \to real$ that acts like a distribution on distributions on $\alpha$, i.e., such that $distOn\ \{f : \alpha \to real \mid dist\ f\}\ F$ holds, we have the formal means to turn it into a "flat" distribution, let us call it *join F*, on $\alpha \to real$, namely by summation (applying of course a well-known mathematical construction): $join\ F\ x \equiv \sum_{f \in \{f \mid dist\ f\}} F\ f\ x$. Now, to define $join_{distrib}$ from *join*, we need to able to move back and forth between $\alpha\ distrib\ distrib$ and and $(\alpha \to real) \to real$, ideally using a similar infrastructure as the abstraction-representation pair $(Abs_{distrib}, Rep_{distrib})$ and predicate *distrib* that allowed us to move between $\alpha\ distrib$ and $\alpha \to real$. And indeed, this is possible:

- We define $dist_2 : ((\alpha \to real) \to real) \to bool$ to be $distOn\ \{f : \alpha \to real \mid dist\ f\}$.
- We define $Abs_{distrib,2} : ((\alpha \to real) \to real) \to (\alpha\ distrib)\ distrib$ by $Abs_{distrib,2}\ F = Abs_{distrib}\ (\lambda\ d : \alpha\ distrib.\ F\ (Rep_{distrib}\ d))$.
- We define $Rep_{distrib,2} : (\alpha\ distrib)\ distrib \to ((\alpha \to real) \to real)$ by $Rep_{distrib,2}\ D = \lambda f : \alpha \to real.\ Rep_{distrib}\ D\ (Abs_{distrib}\ f)$.

Note that we defined $distrib_2$ in line with the above analysis, and defined $Abs_{distrib,2}$ and $Rep_{distrib,2}$ with the aim of achieving a bijective correspondence between $(\alpha\ distrib)\ distrib$ and the elements of $(\alpha \to real) \to real$ satisfyig $dist_2$. Therefore, we can prove that $type\_definition\ Abs_{distrib,2}\ Rep_{distrib,2}\ \{D \mid dist_2\ D\}$ holds.

It now remains to prove the associativity of $join_{distrib}$, which we can rephrase as

$$\forall D : ((\alpha\ distrib)\ distrib)\ distrib.\ join_{distrib}\ (map_{distrib}\ join_{distrib}\ D) = join_{distrib}\ (join_{distrib}\ D)\ .$$

This involves further level of nesting of *distrib*; to this end, by essentially iterating one more step the above construction, we obtain $dist_3 : (((\alpha \to real) \to real) \to real) \to bool$ , $Abs_{distrib,3} : (((\alpha \to real) \to real) \to real) \to ((\alpha\ distrib)\ distrib)\ distrib$ and $Rep_{distrib,3} : ((\alpha\ distrib)\ distrib)\ distrib \to (((\alpha \to real) \to real) \to real)$ such that $type\_definition\ Abs_{distrib,3}\ Rep_{distrib,3}\ \{D \mid dist_3\ D\}$ holds. Now we can easily prove associativity similarly to how we proceeded in the non-nested case, but using the appropriate back and forth infrastructure in each case, depending on the level of nesting.

A somewhat similar discussion applies to distinct lists, though the details differ:

- We define $distinct_2 : (\alpha\ list)\ list \to bool$ by $distinct_2\ xss \equiv distinct\ xss \wedge (\forall i < length\ xss.\ distinct\ (xss!i))$.
- We define $Abs_{dlist,2} : (\alpha\ list)\ list \to (\alpha\ dlist)\ dlist$ by $Abs_{list,2}\ xss = Abs_{list}\ (map\ Abs_{list}\ xss)$.
- We define $Rep_{dlist,2} : (\alpha\ dlist)\ dlist \to (\alpha\ list)\ list$ by $Rep_{list,2}\ xss = Rep_{list}\ (map\ Rep_{list}\ xss)$.

where *map* is the standard mapping operator for lists. Again, we have that $type\_definition\ Abs_{dlist,2}\ Rep_{dlist,2}\ \{xss \mid distinct_2\ xss\}$ holds.

With the goal of a general solution in mind, let us note some similarities and commonalities of the above two cases. While for distinct lists the definitions of the one-level-up abstraction and representation functions involve entities of the same kind (namely abstractions for abstractions and representations for representations), in the case of distributions the definitions combine the two, for example the definition of the one-level-up abstraction uses outer abstraction together with inner representation. This is a reflection of lists being a covariant functor and function-space-to-reals being a contravariant functor.

The common pattern of the two is, however, the fact that the one-level-up operators employ composition between an (1) operator and (2) the map function for the given functor applied to an operator. This is manifestly clear for distinct lists, for example the one-level-up abstraction $Abs_{dlist,2}$ is the composition of $Abs_{dlist}$ with $map_{list}\ Rep_{dlist}$; for distributions, this is also seen to be the case, if we note that the map function for the contravariant functor $\alpha \to real$ is $\lambda g : \alpha \to \beta.\ \lambda f : \beta \to real.\ g \circ f$.

Another discrepancy between the two cases is the definition of the one-level-up characteristic predicates ($distrib_2$ versus $distinct_2$); as we will see next, we will be able to uniformly capture both cases under a more general set-lifting operator.

## 4. An Abstract Formulation of the Problem and a General Solution

We formulate the problem abstractly as follows: *How to lift the abstraction-representation properties characteristic of type definitions to nested applications of the defined type constructor?* In technical terms: Given a polymorphic type $\alpha\ T$ and a polymorphic operator $I : (\alpha\ T)\ set$ such that $I \neq \emptyset$ which produce a type definition

$$\text{typedef}\ \ \alpha\ S = \{x : \alpha\ T \mid x \in I\}$$

what structure and properties are in general required for $T$ and $I$ in order to be able to lift the operator $I$ and abstraction-representation pair ($Abs_S$, $Rep_S$), for any $n$, to $n$-level operator $I_n : (\alpha\ T^n)\ set$ and abstraction-representation pair ($Abs_{S,n}$, $Rep_{S,n}$) with $Abs_{S,n} : \alpha\ T^n \to \alpha\ S^n$ and $Rep_{T,n} : \alpha\ T^n \to \alpha\ S^n$ such that $type\_definition\ Abs_{S,n}\ Rep_{T,n}\ \{x : \alpha\ T \mid I\ x\}$ holds? (Above, $\alpha\ T^n$ denotes the $n$'th iteration of the type constructor $T$, in particular $\alpha\ T^1 = \alpha\ T$ and $\alpha\ T^2 = (\alpha\ T)\ T$; and similarly for $S$.) Indeed, this would allow us to seamlessly apply to the nested case the same back and forth techniques as in the non-nested case.

In the previous section, we have discussed solutions to two instances of this problem. The first instance is representayive of a wide class of situations, namely polymorphic inductive datatypes (which are all covariant functors) with invariants; the second also has some cousins in the formalization and specification literature, for example the defined types topological filters and of mutisets, as well as variations such as discrete subdistributions. In what follows, we introduce a generalization that covers all these cases.

**Assumptions.** We assume that the underlying type constructor $\alpha\ T$ comes equipped with an operator $Trel : \alpha\ set \to \beta\ set \to (\alpha \to \beta \to bool) \to (\alpha\ T \to \beta\ T \to bool)$ for lifting relations to $T$ that for bijective relations commutes with composition; namely, letting $bijBetw\ A\ B\ R$ express that the relation $R$ is a bijection between $A$ and $B$:

(A1) $bijBetw\ A\ B\ P$ and $bijBetw\ B\ C\ Q$ implies $Trel\ A\ C\ (P \circ Q) = Trel\ A\ B\ P \circ Trel\ B\ C\ Q$ for all $A, B, C, P, Q$

Moreover, we assume that there exists an operator $Iset : \alpha\ set \to (\alpha\ T)\ set$ for lifting sets to $T$, which is an extension of $I$ in that

(A2) $Iset\ UNIV = I$

and the following hold, where $eqOn\ A\ R$ says that the restriction of $R$ to $A$ is the equality on (i.e., the diagonal of) $A$:

(A3) $bijBetw\ A\ B\ R$ implies $bijBetw\ (Iset\ A)\ (Iset\ B)\ (Trel\ A\ B\ R)$ for all $A, B, R$
(A4) $bijBetw\ A\ A\ R$ and $eqOn\ A\ R$ implies $eqOn\ (Iset\ A)\ (Trel\ A\ A\ R)$ for all $A, R$

$T$ together with the operator $Trel$ forms a relator-like structure [14, 15], similar to those that underlie Isabelle/HOL's transfer tool [8] and datatype specification mechanism [6]. However, this concept comes with an explicit indication of the domain and codomain sets and targets bijections between these sets. In particular, (A1) only requires $Trel$ to commute with (relation) composition when restricted to bijective

relations. While we think of *Trel* as being associated to $T$, we think of *Iset* as being associated to the invariant $I$ (which it generalizes via (A2)). For example, if $T$ is *list*, then it is reasonable to take *Trel* to be the list relator (relating lists position-wise), but we have no reason to commit to *Iset* as being the standard set operator associated to lists (returning the set of all elements of a list)—rather, the choice of *Iset* will depend on what invariant we want to consider on lists. Of course, as the axiom (A2) suggests, the *Iset* parameter of our abstract framework is reminiscent of the relativized version of the predicate *dist* that we employed in the case of distributions.

*Trel* and *Iset* are connected by the assumptions (A3) and (A4). Thus, (A3) states that *Trel* lifts bijections between two sets to bijections between the *Iset*-liftings of these sets, which roughly means that *Iset* partially acts like a subrelator of (*T*, *Trel*). Finally, (A4) is an *Iset*-relativization of the standard property of relators of preserving equalities—namely, here we say *Trel* preserves partial equalities w.r.t. *Iset*.

Let us see how to instantiate this framework to our running examples. To this end, we first note that having chosen our assumptions in terms of relations rather than functions allows us to capture both covariant and contravariant cases. For the case of the distribution type $\alpha$ *distrib*, we take:

- $\alpha\ T$ to be $\alpha \to real$;
- $Trel\ A\ B\ R\ f\ g$ to be $\Rightarrow_{real}\ (\lambda a, b.\ a \in A \wedge b \in B \wedge R\ a\ b)\ f\ g \wedge (\forall a \notin A.\ f\ a = \bot) \wedge (\forall b \notin B.\ g\ b = \bot)$, where $\bot$ is a (polymorphic) fixed "undefined" element (which is available in HOL on each type via Hilbert choice) and $\Rightarrow_{real}$ is the *real*-instance of the function-space relator, defined by $\Rightarrow_{real}\ P\ f\ g \equiv \forall a, b.\ P\ a\ b \longrightarrow f\ a = g\ b$;
- *Iset* to be given by the relativized form of the *distOn* predicate, namely $Iset\ A \equiv \{f \mid distOn\ A\ f\}$.

For the case of the distinct-list type $\alpha$ *dlist*, we take:

- $\alpha\ T$ to be $\alpha$ *list*;
- $Trel\ A\ B$ to be the list relator *list_all*, where *list_all R* relates two lists just in case they have the same length and their elements are position-wise related (thus, the domain and codomain sets are ignored by *Trel*);
- *Iset* to be defined as $Iset\ A \equiv \{xs \mid distinct\ xs \wedge set\ xs \subseteq A\}$, where $set : \alpha\ list \to \alpha\ set$ is the support operator for lists (returning the set of all elements of a list).

Note the two flavors of instantiating the relativized operator *Iset*, depending on whether we deal with a contravariant or covariant functor, namely : (1) either, in the contravariant case, by relativizing the original predicate *dist* to *distOn*; (2) or, in the covariant case, by intersecting the original predicate *distinct* with the adjoint of the support operator (which is usually available for covariant functors, and in particular is available for all container types)—indeed, the righthand side of the definition of *Iset A* can be written as $\{xs \mid distinct\ xs\} \cap \{xs \mid set\ xs \subseteq A\}$, and the operator $L = \lambda A.\ \{xs \mid set\ xs \subseteq A\}$ is the right adjoint[4] of the support operator *set* in the sense that $xs \in L\ A$ iff $set\ xs \subseteq A$. It is relatively easy to check that these instances satisfy our assumptions. For example, (A3) in the case of the distribution instantiation says that a bijection between two sets $A$ and $B$ induces a bijection the sets of distributions on $A$ and $B$ respectively, which are constant $\bot$ outside $A$ and $B$ respectively.

Our main result is that these abstract assumptions are sufficient for solving our problem, thus generalizing the constructions in the above particular cases (and in many other cases, e.g., any datatypes with invariants turned into typedefs).

**Theorem:** Under the assumptions (A1)–(A4) above, we have a solution to our problem for all $n \geq 1$, in that there exist $I_n : (\alpha\ T^n)\ set$, $Abs_{S,n} : \alpha\ T^n \to \alpha\ S^n$ and $Rep_{T,n} : \alpha\ T^n \to \alpha\ S^n$ such that $type\_definition\ Abs_{S,n}\ Rep_{T,n}\ I_n$ holds.

A formal proof in Isabelle/HOL of the core of this theorem can be found in [11]; due to the HOL type system limitations, the formal proof is restricted to the case when $n = 2$, but also indicates how to iterate the argument for arbitrary $n$ and shows the iterations for 3 and 4.

[4]Incidentally, in Isabelle/HOL the operator $L$ is called *lists*—a suggestive name given that this operator is the set-based counterpart of the *list* type constructor: it takes any set $A$ to the set of lists formed with elements of $A$.

*Proof sketch.* We proceed by induction on $n$. For $n = 1$, we simply take $I_n = I$, $Abs_{S,n} = Abs_S$ and $Rep_{S,n} = Rep_S$, so the desired fact holds by our assumptions.

For the induction step, assume that we have $I_n : (\alpha\ T^n)\ set$, $Abs_{S,n} : \alpha\ T^n \to \alpha\ S^n$ and $Rep_{T,n} : \alpha\ T^n \to \alpha\ S^n$ such that $type\_definition\ Abs_{S,n}\ Rep_{T,n}\ I_n$ holds. We define $I_{n+1} : (\alpha\ T^{n+1})\ set$, $Abs_{S,n+1} : \alpha\ T^{n+1} \to \alpha\ S^{n+1}$ and $Rep_{T,n+1} : \alpha\ T^{n+1} \to \alpha\ S^{n+1}$ as follows: $I_{n+1} \equiv Iset\ I_n$; $Abs_{S,n+1} = Abs_S \circ funOf\,(Trel\,(relOf\,Abs_{S,n}))$; $Rep_{S,n+1} = Rep_S \circ funOf\,(Trel\,(relOf\,Rep_{S,n}))$.

Above, the operators $relOf : (\alpha \to \beta) \to (\alpha \to \beta \to bool)$ and $funOf : (\alpha \to \beta \to bool) \to (\alpha \to \beta)$ provide back and forth conversions between bijective relations and (partially) bijective functins. Namely, we have the following properties for them, where $bij\_betw\ A\ B\ f$ says that the restriction of the function $f : \alpha \to \beta$ to $A : \alpha\ set$ is a bijection between $A$ and $B : \beta\ set$: (1) If $bij\_betw\ A\ B\ f$, then $bijBetw\ A\ B\ (relOf\ f)$ and $funOf\,(relOf\ f) = f$; (2) If $bijBetw\ A\ B\ R$, then $bij\_betw\ A\ B\ (funOf\ R)$ and $relOf\,(funOf\ f R = R$.

To prove $type\_definition\ Abs_{S,n+1}\ Rep_{T,n+1}\ I_{n+1}$ compositionally (along the definitions of the abstraction ad represenations operators), we introduce a generalization of *type_definition* that does not assume one of its argument functions (namely the representation function) to operate on the entire domain, but on an additional parameter set. Namely, we define $bij\_pair : (\beta \to \alpha) \to (\alpha \to \beta) \to \beta\ set \to \alpha\ set \to bool$ as follows, where we highlight the differences from *type_definition*: $bij\_pair\ r\ a\ BA \equiv (\forall y.\ y \in B \longrightarrow r\ y \in A) \wedge (\forall x.\ x \in A \longrightarrow a\ x \in B) \wedge (\forall y : \beta.\ y \in B \longrightarrow a\,(r\ y) = y) \wedge (\forall x : \alpha.\ x \in A \longrightarrow r\,(a\ x) = x)$. Thus, $bij\_pair\ r\ a\ B\ A$ says that $a$ and $r$ are mutually inverse bijections between $A$ and $B$; in particular, we have (3) $bij\_pair\ r\ a\ B\ A = type\_definition\ r\ a\ UNIV\ A$.

Next, we define the relational counterpart of *bij_pair*, namely $bijPair : (\beta \to \alpha \to bool) \to (\alpha \to \beta \to bool) \to \beta\ set \to \alpha\ set \to bool$, such that $bijPair\ P\ Q\ B\ A$ says that $P$ and $Q$ are mutually inverse (relational) bijections between $A$ and $B$. The operators *bij_pair* and *bijPair* correspond to each other via the translations between functions and relations: (4) If $bij\_pair\ r\ a\ B\ A$ then $bijPair\ (relOf\ r)\ (relOf\ a)\ B\ A$; (5) If $bijPair\ P\ Q\ B\ A$ then $bij\_pair\ (funOf\ P)\ (funOf\ Q)\ B\ A$. They also commute with relation and function composition: (6) If $bij\_pair\ r\ a\ B\ A$ and $bij\_pair\ r'\ a'\ A\ C$ then $bij\_pair\ (r' \circ r)\ (a' \circ a)\ B\ C$; (7) If $bijPair\ P\ Q\ B\ A$ and $bij\_pair\ P'\ Q'\ A\ C$ then $bijPair\ (P \circ P')\ (Q \circ Q')\ B\ C$.

Finally, using (A1), (A3) and (A4), we can prove the crucial property that *Trel* "lifts" the *bijPair* property relative to *Iset*: (8) If $bijPair\ P\ Q\ A\ B$ then $bijPair\ (Trel\ A\ B\ P)\ (Trel\ B\ A\ Q)\ (Iset\ A)\ (Iset\ B)$. □

Note that the theorem asserts the existence of $n$-level predicates $I_n$ and abstraction-representation pairs $(Abs_n, Rep_n)$ which extend the original ones, $I$ and $(Abs, Rep)$. But the question arises on whether these are the "right" extensions. The answer relies only on the suitability of $I_n$, since once that is decided than *any* projection pair $(Abs_n, Rep_n)$ satisfying $type\_definition\ Abs_{S,n}\ Rep_{T,n}\ I_n$ would do—mirroring the fact that in a type definition (at level 1) only $I$ matters and any $(Abs, Rep)$ satisfying $type\_definition\ Abs_S\ Rep_T\ I$ is as good as any other.

Now, concerning the suitability of $I_n$, we note from the proof that $I_n = Iset^n\ UNIV$; so it all hinges upon whether *Iset* is the "right" way of lifting sets of elements of $\alpha$ to sets of elements of $\alpha\ T$ that respects the intended meaning of the subset $I$ of $\alpha\ T$. In other words, whatever concept $I$ is supposed to represent, we want that *Iset* provides a correct relativization of that concept from the entire type $\alpha$ to a subset $A : \alpha\ set$. Our assumptions (A2)–(A4) are sanity properties for such a relativization, but whether this is the correct relativization needs to be established in each particular case—in other words, it is the responsability of the user of our framework to provide a correct and meaningful *Iset* operator. This is easily seen to be the case in our two examples, where the move from $I$ to *Iset* clearly represents the move from distributions on the whole type to distributions on a given subset, and from distinct lists on the whole type to distinct lists on a given subset, respectively. Moreover, the scheme that worked for distinct lists (of intersecting with the support operator) works for essentially the same reason for any container type.

Finally, a remark on the generality of the theorem: While we have formulated it in reference to the (possibly iterated) nesting of a *single* type constructor, the construction and proof can be straightforwardly (albeit tediously) extended to cope with combining / nesting different type constructors (which can also have more than one argument).

## 5. Related Work and Future Work

Isomorphic transfer is an important topic in formal reasoning, and is one of the main motivations of major recent developments such as Homotopy Type Theory [16]. Transfer along ismorphisms, as well as along quotient projections and refinements, is also well represented in the world of HOL-based provers (and especially Isabelle/HOL), e.g., [17, 18, 19, 8, 20, 21]. In dependent type theory, problems similar to the ones we address here are formulated in the context of (partial) setoids [22, 23], and proof assistants such as Coq [24] (via SSReflect [25]) and Lean [26] offer quasi-automated mechanisms to address them, thus mitigating what is referred to as "setoid hell", i.e., the need to prove over and over again that certain (partial) equivalence relations are preserved by the defined functions. Note that the problem we addressed in this paper, which has to do with relativization to sets, is a particular case of relativization to partial equivalence relations—and in fact dealing with the former at higher-order types quickly escalates to having to deal with the latter, even in the absence of dependent types [10].

Another technical part of our setting concerns the HOL defined types, which are not included in their base types but only injected into them. Other formalisms offer pure subtyping / inclusion mechanisms, notably PVS [27], F$^\star$ [28], and logical frameworks featuring refinement types [29]—where the goal of isomorphic transfer is replaced by corresponding goals concerning automating aspects of type checking.

Next, we will focus on the most closely related work, namely Isabelle's Lifting and Transfer package [30, 8]. From the very beginning, the authors of this tool have been aware of the potential difficulties raised by nested type constructors, and have implemented a solution based on parameterized transfer relations. For example, in the case of *distrib*, the current implementation defines automatically a relation in $(\alpha \to real) \to \beta\ distrib \to bool$ (thus using different type variables $\alpha$ and $\beta$) as the composition between the relator $\Rightarrow_{real}$ and the relational version of $Abs_{distrib}$. This does not guarantee an isomorphic relationship between nested applications of $(\_ \to real)$ and *distrib* like our framework does (exported as a *type_definition* theorem), but offers enough infrastructure in order to allow the user to "lift" the definition of constants from $(\_ \to real)$ to *distrib* even in the presence of nesting. Since the relativized defining set is not provided explicitly (like in our framework, as the operator *Iset*), the proof goals arising when transferring theorems containing such constants are quite intricate and convoluted, in sharp contrast to the non-nested case. On the other hand, at the expense of asking the user to provide more information (not just the relator *Trel* but also the operator *Iset*) and proving some sanity properties, our approach really flattens everything to the "first-order" non-nesting case—with the potential of simplifying the proof goals. Moreover, currently the transfer package relies on a parametricity theorem for the defined type's underlying predicate, which our approach does not. Overall, our approach requires some more initial effort from the user upon introducing a new type, but this has the potential to pay off in the later stages of the developments.

This having been said, our work addresses only a subproblem of the overall lifting and transfer problem, which the Lifting and Transfer package addresses quite comprehensively. So we do not envision our development as an alternative to this mature tool, but as a possible "add-on" that can improve the usability and automation of the tool's handling of nested types. Since what we produce for nested types are *type_definition* theorems, these can in principle be directly integrated into the Lifting and Transfer package (via the "setup_lifting" command), save for one difficulty caused by the fact that any provided abstraction operators are currently required to be registered as dataype-like constructors—addressing this formal engineering problem is ongoing work. (Of course, the integration will involve the fully general case, of different type constructors of possible multiple arguments nested in arbitrary ways.)

Another short-term plan is to provide some generic infrastructure that automates our inferred result for arbitrary type constructors, which can then be instantiated to different cases. This is being tackled (in joint work with Dmitriy Traytel) with the help of a quasi-foundational developement of polymorphic locales, generalizing Isabelle's standard (monomorphic) locales [31, 32] in a manner that does not impair the meta-properties of the logic and definitional mechanisms underlying Isabelle/HOL [33, 34, 35, 36].

**Acknowledgments.** We thank the three anonymous reviewers for their valuable comments and suggestions, which led to the improvement of the presentation and to the discussion of more related work.

## References


[1] The HOL Team, The HOL4 Theorem Prover, 2024. Http://hol.sourceforge.net/.

[2] J. Harrison, HOL light: An overview, in: S. Berghofer, T. Nipkow, C. Urban, M. Wenzel (Eds.), TPHOLs 2009, volume 5674 of *LNCS*, Springer, 2009, pp. 60–66. doi:10.1007/978-3-642-03359-9_4.

[3] T. Nipkow, L. C. Paulson, M. Wenzel, Isabelle/HOL — A Proof Assistant for Higher-Order Logic, volume 2283 of *LNCS*, Springer, 2002. doi:10.1007/3-540-45949-9.

[4] S. Berghofer, M. Wenzel, Inductive datatypes in HOL – lessons learned in formal-logic engineering, in: Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin-Mohring, L. Théry (Eds.), TPHOLs 1999, volume 1690 of *LNCS*, Springer, 1999, pp. 19–36. doi:10.1007/3-540-48256-3_3.

[5] D. Traytel, A. Popescu, J. C. Blanchette, Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving, in: LICS 2012, IEEE Computer Society, 2012, pp. 596–605. doi:10.1109/LICS.2012.75.

[6] J. C. Blanchette, J. Hölzl, A. Lochbihler, L. Panny, A. Popescu, D. Traytel, Truly modular (co)datatypes for isabelle/hol, in: G. Klein, R. Gamboa (Eds.), ITP 2014, volume 8558 of *LNCS*, Springer, 2014, pp. 93–110. doi:10.1007/978-3-319-08970-6_7.

[7] J. C. Blanchette, F. Meier, A. Popescu, D. Traytel, Foundational nonuniform (co)datatypes for higher-order logic, in: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017, IEEE Computer Society, 2017, pp. 1–12. URL: https://doi.org/10.1109/LICS.2017.8005071. doi:10.1109/LICS.2017.8005071.

[8] B. Huffman, O. Kuncar, Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL, in: G. Gonthier, M. Norrish (Eds.), CPP 2013, volume 8307 of *LNCS*, Springer, 2013, pp. 131–146. doi:10.1007/978-3-319-03545-1_9.

[9] O. Kunčar, A. Popescu, From types to sets by local type definition in higher-order logic, J. Autom. Reason. 62 (2019) 237–260. doi:10.1007/s10817-018-9464-6.

[10] A. Popescu, D. Traytel, Admissible types-to-pers relativization in Higher-Order Logic, Proc. ACM Program. Lang. 7 (2023) 1214–1245. URL: https://doi.org/10.1145/3571235. doi:10.1145/3571235.

[11] G. Buday, A. Popescu, Supplementary material associated to this paper, https://www.andreipopescu.uk/suppl/iFM2024.zip, 2024.

[12] R. Godement, Topologie algébrique et théorie des faisceaux, Publications de l'Institut de Mathématique de l'Université de Strasbourg, Hermann, Paris, 1958.

[13] E. Moggi, Notions of computation and monads, Inf. Comput. 93 (1991) 55–92. URL: https://doi.org/10.1016/0890-5401(91)90052-4. doi:10.1016/0890-5401(91)90052-4.

[14] J. C. Reynolds, Types, abstraction and parametric polymorphism, in: R. E. A. Mason (Ed.), IFIP 1983, North-Holland/IFIP, 1983, pp. 513–523.

[15] P. Wadler, Theorems for free!, in: J. E. Stoy (Ed.), FPCA 1989, ACM, 1989, pp. 347–359. doi:10.1145/99370.99404.

[16] T. Univalent Foundations Program, Homotopy Type Theory: Univalent Foundations of Mathematics, https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[17] P. V. Homeier, A design structure for higher order quotients, in: J. Hurd, T. F. Melham (Eds.), Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, volume 3603 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 130–146.

[18] C. Kaliszyk, C. Urban, Quotients revisited for Isabelle/HOL, in: W. C. Chu, W. E. Wong, M. J. Palakal, C. Hung (Eds.), Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011, ACM, 2011, pp. 1639–1644. URL: https://doi.org/10.1145/1982185.1982529. doi:10.1145/1982185.1982529.

[19] P. Lammich, Automatic data refinement, in: S. Blazy, C. Paulin-Mohring, D. Pichardie (Eds.), Interactive Theorem Proving - 4th International Conference, ITP, volume 7998 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 84–99.

[20] A. Schropp, A. Popescu, Nonfree datatypes in Isabelle/HOL - animating a many-sorted metatheory, in: G. Gonthier, M. Norrish (Eds.), Certified Programs and Proofs - Third International Conference,

CPP, volume 8307 of *LNCS*, Springer, 2013, pp. 114–130.

[21] M. Milehins, An extension of the framework types-to-sets for Isabelle/HOL, in: A. Popescu, S. Zdancewic (Eds.), CPP 2022, ACM, 2022, pp. 180–196. doi:10.1145/3497775.3503674.

[22] G. Barthe, V. Capretta, O. Pons, Setoids in type theory, J. Funct. Program. 13 (2003) 261–293. doi:10.1017/S0956796802004501.

[23] T. Altenkirch, S. Boulier, A. Kaposi, N. Tabareau, Setoid type theory - A syntactic translation, in: G. Hutton (Ed.), Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings, volume 11825 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 155–196. URL: https://doi.org/10.1007/978-3-030-33636-3_7. doi:10.1007/978-3-030-33636-3_7.

[24] Y. Bertot, P. Castéran, Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2004. URL: https://doi.org/10.1007/978-3-662-07964-5. doi:10.1007/978-3-662-07964-5.

[25] G. Gonthier, A. Mahboubi, An introduction to small scale reflection in coq, J. Formaliz. Reason. 3 (2010) 95–152. URL: https://doi.org/10.6092/issn.1972-5787/1979. doi:10.6092/ISSN.1972-5787/1979.

[26] L. M. de Moura, S. Kong, J. Avigad, F. van Doorn, J. von Raumer, The lean theorem prover (system description), in: A. P. Felty, A. Middeldorp (Eds.), CADE-25, volume 9195 of *LNCS*, Springer, 2015, pp. 378–388. doi:10.1007/978-3-319-21401-6_26.

[27] J. M. Rushby, S. Owre, N. Shankar, Subtypes for specifications: Predicate subtyping in PVS, IEEE Trans. Software Eng. 24 (1998) 709–720. URL: https://doi.org/10.1109/32.713327. doi:10.1109/32.713327.

[28] N. Swamy, C. Hritcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P. Strub, M. Kohlweiss, J. K. Zinzindohoue, S. Z. Béguelin, Dependent types and multi-monadic effects in F, in: R. Bodík, R. Majumdar (Eds.), Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016, ACM, 2016, pp. 256–270. URL: https://doi.org/10.1145/2837614.2837655. doi:10.1145/2837614.2837655.

[29] W. Lovas, F. Pfenning, Refinement types for logical frameworks and their interpretation as proof irrelevance, Log. Methods Comput. Sci. 6 (2010). URL: https://doi.org/10.2168/LMCS-6(4:5)2010. doi:10.2168/LMCS-6(4:5)2010.

[30] O. Kunčar, Types, Abstraction and Parametric Polymorphism in Higher-Order Logic, Ph.D. thesis, Fakultät für Informatik, Technische Universität München, 2016. URL: http://www21.in.tum.de/~kuncar/documents/kuncar-phdthesis.pdf.

[31] F. Kammüller, M. Wenzel, L. C. Paulson, Locales - A sectioning concept for Isabelle, in: Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin-Mohring, L. Théry (Eds.), Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings, volume 1690 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 149–166. URL: https://doi.org/10.1007/3-540-48256-3_11. doi:10.1007/3-540-48256-3_11.

[32] C. Ballarin, Locales and locale expressions in isabelle/isar, in: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers, volume 3085 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 34–50. URL: https://doi.org/10.1007/978-3-540-24849-1_3. doi:10.1007/978-3-540-24849-1_3.

[33] O. Kuncar, A. Popescu, Comprehending Isabelle/HOL's consistency, in: H. Yang (Ed.), Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, volume 10201 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 724–749. URL: https://doi.org/10.1007/978-3-662-54434-1_27. doi:10.1007/978-3-662-54434-1_27.

[34] O. Kuncar, A. Popescu, A consistent foundation for Isabelle/HOL, J. Autom. Reason. 62 (2019)

531–555. URL: https://doi.org/10.1007/s10817-018-9454-8. doi:10.1007/S10817-018-9454-8.

[35] O. Kuncar, A. Popescu, Safety and conservativity of definitions in HOL and Isabelle/HOL, Proc. ACM Program. Lang. 2 (2018) 24:1–24:26. URL: https://doi.org/10.1145/3158112. doi:10.1145/3158112.

[36] A. Gengelbach, T. Weber, Proof-theoretic conservative extension of HOL with ad-hoc overloading, in: V. K. I. Pun, V. Stolz, A. Simão (Eds.), Theoretical Aspects of Computing - ICTAC 2020 - 17th International Colloquium, Macau, China, November 30 - December 4, 2020, Proceedings, volume 12545 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 23–42. URL: https://doi.org/10.1007/978-3-030-64276-1_2. doi:10.1007/978-3-030-64276-1_2.

# 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

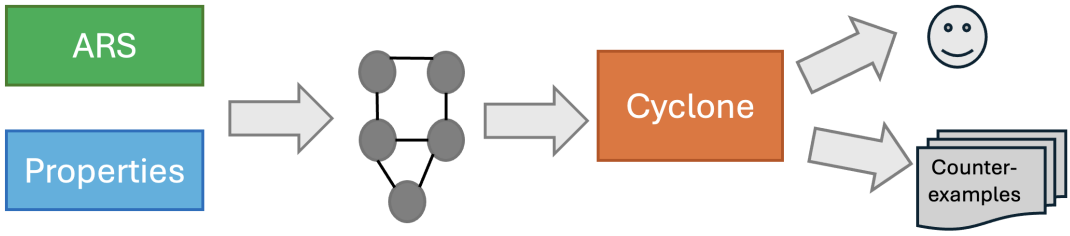



**Figure 1:** The overview of our proposed approach. An ARS along with its properties are represented using a graph that can be mapped to a Cyclone specification for verification.

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.

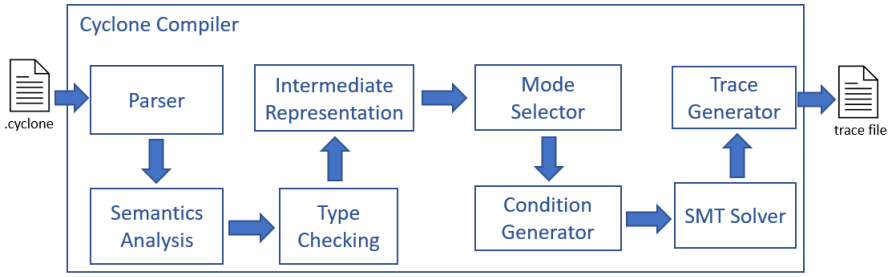



**Figure 2:** Architecture of Cyclone.

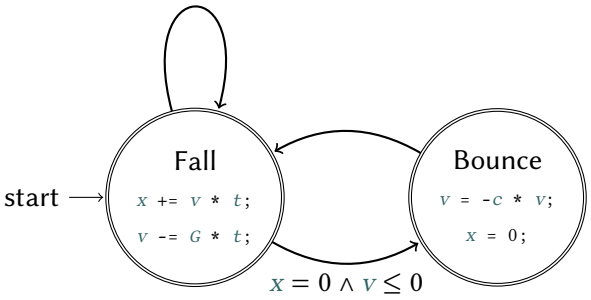



**Figure 3:** A transition graph for a bouncing ball.

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.

# Extended Abstract: Skill-Based Architectures in Autonomous Systems: Lessons Learnt

Pierre Malafosse[1,2], Alexandre Albore[2], Jeremie Guiochet[1] and Charles Lesire[2]

[1]TRUST, LAAS-CNRS, Université de Toulouse, 31000, Toulouse, France
[2]DTIS, ONERA, Université de Toulouse, 31000, Toulouse, France

**Abstract**

The deployment of complex autonomous systems into open environment calls for robust, modular and verifiable software architectures. Skillset models, which encapsulate the capabilities of the autonomous robots into modular Skills, have emerged to address these challenges. Expressed within an intermediary layer of the robot's architecture, Skillsets break down high level actions produced by the Deliberative layer into lower level actions, executed by functional components. The use of a Domain Specific Language (DSL) for Skillset modeling allow formal methods to provide robust code generation and model verification. However, as Skill-based architectures are more and more developed, the true benefits of opting for such an approach have yet to be investigated.

This extended abstract presents an early-stage analysis of recent Skill-based architectures making use of the ONERA Robot Skill Toolchain[1] with a focus on identifying best practices, common pitfalls, and offering guidelines for future developments. The experiment relies on the works of several french laboratories, all making use of the Robot Language DSL formalized by Albore et al.[1]. The findings suggest that while Skill-based architectures present undeniable advantages in the development of autonomous systems, there is currently a need for more standardized frameworks.

**Keywords**

Autonomous systems, robotic architectures, robotics, Skill-based architectures

## 1. Introduction

The development of autonomous systems presents the major challenge of integrating low-level functions (e.g., actuation) and high-level decision-making capabilities (e.g., task scheduling) into robust and reliable software architectures [2, 3, 4]. A common approach involves using layered architectures, especially a three-layer architecture [5, 6] (Functional, Executive, and Deliberative layers). In this architecture, *Skills* are abstractions of the robot's capabilities [7]. A robot has a *Skillset* from which Skills can be executed following plans provided by the Deliberative layer. Skills are formally modeled in the Robot Language DSL [8] to facilitate formal verification [1, 9, 10] and reuse. Their flexibility and adaptability enables non-experts to program and control robots by redesigning missions or adding new Skills [11, 12]. This *Skillset model* is used for code generation creating the robot's Executive layer, the *Skillset Manager*. Code generation reduces the occurrence of faults and ensures the Skills behave as defined in the DSL. Finally, developers define the *Skillset Implementation*, the interface between the Functional layer and the Skillset Manager.

The Robot Language DSL has been evolving over the past few years, improving both on the aspects of robustness and verifiability [1, 9, 10]. Several developers have implemented the Skill-based Executive layer on different platforms and for different use cases. Yet, there is currently no established development process or framework for developing such architectures. The modeling of Skills, the structure of the Skillset implementation and the location of error handling features are left to the user's preference. Design choices and trends must be investigated in the hope of unveiling good practices and common mistakes. This study marks the beginning of a Ph.D. thesis with the aim to improve the confidence

[1]Robot Skills documentation

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

pierre.malafosse@onera.fr (P. Malafosse); alexandre.albore@onera.fr (A. Albore); guiochet@laas.fr (J. Guiochet); charles.lesire@onera.fr (C. Lesire)

level in the three-layer architectures used in autonomous systems by proposing new approaches to specify Skill-level and multi-level recovery strategies, with a focus on a Skill-based Executive layer as defined by Albore et al.[1], and displayed in Figure 1.

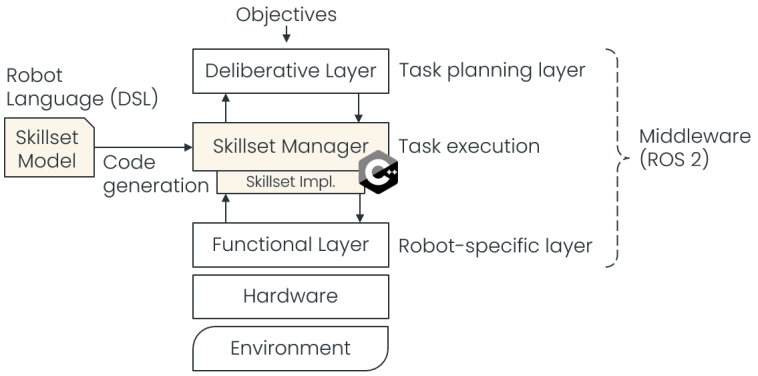



**Figure 1:** 3-layer architecture as defined by Albore et al.[1] with the Skill-based layer implemented as the Executive layer.

As a first step, we present an early-stage analysis of recent Skill-based architectures making use of the ONERA Robot Skill Toolchain[1] with a focus on identifying best practices, common pitfalls, and offering guidelines for future developments. To this end, we focus on three research questions (RQs):

RQ 1. To which extent is the level of abstraction proposed by the Skills suitable for different applications?
Metrics: Assessment of the Reusability and adaptivity of Skills both across missions and across systems.

RQ 2. In what ways does the abstraction level of Skills influence the management of system complexity?
Metrics: Assessment of the Mission-level complexity and architecture-level complexity

RQ 3. Does the use of Skills facilitate the integration of error detection and handling mechanisms?
Metrics: Nature and localization of the error handling mechanisms in the Skill-based architecture.

With these three research questions set, we present the sources and methods used for the experiment in Section 2. Section 3 then summarizes our observations and results before concluding with Section 4.

## 2. Materials and methods

First, lets us point out that the studied Skill-based approach is an open-source research project, with a limited community of developers. Nevertheless, over the past few years, numerous works using this technology have been conducted. Discarding versions form before 2022, the 22 most recent projects involving a Skill-based architecture have been investigated in the present work. Data ranging from interpretations of the code to interviews with the developers has been collected. These projects were conduced by several French laboratories, namely ONERA and LAAS-CNRS in Toulouse as well as the LIRMM in Montpellier. The data is diverse, as are involved a wide variety of robot types: Manipulator Arms, Unmanned Aerial Vehicles (UAVs), Unmanned Marine Vehicles (UMVs), Unmanned Ground Vehicles (UGVs) and Legged Robots, sometimes working all together to achieve the same goal. Each project is treated as an individual case study and each Skill-based architecture is assessed, following the research questions, on how the Skills were designed, reused and adapted. Thirteen individuals (including Ph.D. students and senior researchers) involved in these projects were interviewed. In addition to code analysis, semi-structured interviews were conducted to elucidate the mission contexts, rationale behind design decisions, and practical challenges faced during the implementation of the Skills. The interview protocol was meticulously designed to address the three primary research questions.

[1]Robot Skills documentation

Developers provided valuable insights into the process of implementing Skill-based architectures, specifically focusing on strategies for managing system complexity, Skill correctness, and integrating error-handling mechanisms.

# 3. Results

## 3.1. RQ 1: Suitability of Skill Abstraction

First, we investigated the reusability and genericity of Skills. The analysis of the 22 project codes shows that while Skills are theoretically meant to be reusable and platform-independent, they tend to only be reused within the same robot type (e.g., UAVs, UGVs). Generic Skillset models have been developed for each type and the crossing of a Skill model between different robot types is rare. This is mainly due to diverging functional needs. For example, while UGVs and UAVs may share the same capability of reaching a specific waypoint, UAVs need to process additionnal information such as altitude values, flight status or safety requirements. Therefore, distinct movement-related Skills have been implemented. This observation leads to the matter of adaptivity of the Skills. Due to the lack of modularity and specific project needs, developer teams adopted diverging solutions. Some directly modified generic Skillsets to fit mission-specific needs, while others created new, dedicated Skillsets for a precise robot type. We observe an unbalance between genericity and adaptivity of Skill models: On the one hand, the more a Skill is generic, the more difficult it is to implement for specific needs. On the other hand, highly adaptive Skillsets are bound to their initial robot type and cannot easily be adapted elsewhere. The framework does not currently propose explicit, documented solutions on the matter. However, we observe significant similarities between the existing models and argue that the creation of a truly generic Skillset model is possible. To this end, an unified framework for designing Skill-based architectures is needed.

The key takeaways regarding RQ 1 are:

- The framework does not currently propose an explicit and documented list of generic Skills.
- The framework does not include a built-in extension mechanism (such as inheritance in object-oriented programming) to ease the redefinition and specialization of Skillsets.
- There is currently no training material on generic Skill reuse.

## 3.2. RQ 2: Effect on Complexity Management

Complexity management can be tackled in many ways. First insights from the interviewed participants concerned mission complexity. In this context, the Skill abstraction helps simplify interactions with non-experts and stakeholders by focusing on what the robot does rather than how it operates at a low level. This approach was particularly useful in projects with non-technical stakeholders, such as marine biologists in one of the case studies.

Another perspective is at the architecture level, where Skills may also play a role for managing complexity. According to the interviewed participants, without a Skill-based architecture, the development process would be longer and significantly more complex. Indeed, the abstraction of the robot's capabilities into Skills prevents the developers from making direct connections between high level tasks and low-level functional components: The Deliberative layer only reasons in term of Skill activation while the Skills manage the execution of the lower-level components. This modular approach also simplifies the integration of new functionalities. The use of ROS2 as the middleware distributing the Skill-based architecture may also influence the architecture-level complexity. The Executive layer uses ROS2 to harvest and propagate useful information to the whole system. Furthermore, robots from different projects (e.g., Boston Dynamics®Spot and Kinova®manipulator) were able to communicate and coordinate Skills effectively thanks to the shared middleware.

Key takeways regarding RQ 2 are:

- The abstraction of the robot's capabilities allows for simpler and clearer mission definitions.

- The abstracted Skills simplify the implementation and/or modification of the architecture by allowing for clear connections between modular components.
- Thanks to the ROS2 middleware, the Skillset Manager is able to centralize and spread information internally as well as externally, in multi-agent scenarios.

### 3.3. RQ 3: Effect on Error Detection and Handling

The Skillset manager is C++ code generated from a Skillset model which has been formally verified beforehand [1, 9]. This approach reduces the occurrence of faults within the Executive layer and ensures the Skills behave as intended. Furthermore, Skills provide error detection and handling mechanisms through the definition of preconditions, invariants, and postconditions which prevent a failure from the neighboring layers to propagate further. The Skillset implementation, making the link with the functional layer, is user defined and allows developers to integrate custom error detection and recovery mechanisms. However, what has transpired through both the interviews and the code reviews is the absence of method to identify error conditions and reactions and how to include them into the architecture. A work was done by Medina et al. [13] using Fault Tree Analysis (FTA) to create Skill fault models and help identify the potential causes of failures of the system. Albore et al.[1] made use of these Skill fault models to implement fallback modes onto a Behavior Tree and identified missing detection mechanisms within the Skillset implementation. However, we observed no unified practice for including error detection and handling within the 22 projects, apart from the mandatory formal verification of the Skillset model.

Key takeways regarding RQ 3 are:

- There is a lack of method to bind fault propagation analysis to Skill models
- There is a need for documented good practices to help developers choose the nature and location of the error detection and recovery mechanisms
- There is a need for tools to identify scenarios in which multiple (possibly incompatible) recovery actions are simultaneously triggered.

## 4. Conclusion

The present study has demonstrated, through an examination of 22 projects, the benefits and challenges inherent to the use of Skill-based architectures. Our findings indicate that, while Skills offer a robust mechanism for modular programming and rapid reconfiguration, they have yet to meet all their promises. Indeed, due to a lack of genericity and adaptivity, such implementations are currently not fully accessible to non-experts, because applications to specific robotic types, with specific functional needs, necessitate a process of careful consideration and adaptation. Regarding the genericity and adaptability of Skills, we posit that the development of truly generic Skill models is feasible. While this endeavor is beyond the scope of the current Ph.D. research, future studies should be conducted in this direction.

This study has highlighted a clear benefit of implementing Skill-based architectures into increasingly complex autonomous systems: A list of benefits in term of mission-level and architecture-level complexity management has transpired from the interviews while no significant downsides have been observed. On top of the the previously mentioned aspects, complexity management can therefore be a reason why developers choose to use Skills.

Our study also underscores the need of a clearly defined framework for the development of both the Skillset models and the Skillset implementations. The absence of formal guidelines allows for significant flexibility in the approach taken by the developer. This may result in inconsistencies within the implementations and could eventually compromise the safety and robustness of the system. The same lack of precise guidelines applies for the implementation of error detection and handling mechanisms.

In this context, the Future works related to the Ph.D. will be focusing on Skill-level error handling as well as a multi-level recovery framework with the aim of addressing the need for guidelines to safer, more robust implementations. To this end, we currently investigate the capabilities of BDI agents in term

of failure handling [14]. A BDI-based Deliberative layer coupled with a Skill-based Executive layer could indeed prove interesting in term of failure handling, and multi-level recovery [15]. Furthermore, the formal aspect of both paradigms may allow for the creation of robust formal verification tools [16, 17, 18].

## References


[1] A. Albore, D. Doose, C. Grand, J. Guiochet, C. Lesire, A. Manecy, Skill-based design of dependable robotic architectures, Robotics and Autonomous Systems 160 (2023). doi:10.1016/j.robot.2022.104318.

[2] T. Lozano-Perez, Robot programming, Proceedings of the IEEE 71 (1983) 821–841.

[3] T. Lozano-Pérez, R. A. Brooks, An approach to automatic robot programming, International Conference on Scientific Computing (1986) 61–69. doi:10.1145/324634.325195.

[4] R. A. Brooks, A robust layered control system for a mobile robot, IEEE Journal on Robotics and Automation 2 (1986) 14–23. doi:10.1109/JRA.1986.1087032.

[5] E. Gat, R. P. Bonnasso, R. Murphy, et al., On three-layer architectures, Artificial intelligence and mobile robots 195 (1998) 210.

[6] R. Simmons, D. Apfelbaum, A task description language for robot control, in: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications, 1998.

[7] J. Urzelai, D. Floreano, M. Dorigo, M. Colombetti, Incremental robot shaping, Connection Science 10 (1998) 341–360. doi:10.1080/095400998116486.

[8] C. Lesire, D. Doose, C. Grand, Formalization of robot skills with descriptive and operational models, IEEE International Conference on Intelligent Robots and Systems (2020) 7227–7232. doi:10.1109/IROS45743.2020.9340698.

[9] B. Pelletier, C. Lesire, D. Doose, K. Godary-Dejean, C. Dramé-Maigné, Skinet, a petri net generation tool for the verification of skillset-based autonomous systems, Electronic Proceedings in Theoretical Computer Science, EPTCS 371 (2022) 120–138. doi:10.4204/EPTCS.371.9.

[10] B. Pelletier, C. Lesire, C. Grand, D. Doose, M. Rognant, Predictive runtime verification of skill-based robotic systems using petri nets, in: Proc. of IEEE International Conference on Robotics and Automation, volume 2023-May, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 10580–10586. doi:10.1109/ICRA48891.2023.10160434.

[11] C. Laugier, T. Fraichard, P. Garnier, I. E. Paromtchik, A. Scheuer, Sensor-based control architecture for a car-like vehicle, Autonomous Robots 6 (1999) 165–185. doi:10.1023/A:1008835527875/METRICS.

[12] A. Bjorkelund, L. Edstrom, M. Haage, J. Malec, K. Nilsson, P. Nugues, S. G. Robertz, D. Storkle, A. Blomdell, R. Johansson, M. Linderoth, A. Nilsson, A. Robertsson, A. Stolt, H. Bruyninckx, On the integration of skilled robot motions for productivity in manufacturing, in: Proc. of IEEE International Symposium on Assembly and Manufacturing (ISAM), 2011. doi:10.1109/ISAM.2011.5942366.

[13] G. C. Medina, J. Guiochet, C. Lesire, A. Manecy, A skill fault model for autonomous systems, in: Proceedings of the 4th International Workshop on Robotics Software Engineering, 2022, pp. 55–62.

[14] S. Sardina, L. Padgham, A BDI agent programming language with failure handling, declarative goals, and planning, Autonomous Agents and Multi-Agent Systems 23 (2011) 18–70.

[15] L. A. Dennis, M. Fisher, Actions with durations and failures in BDI languages, in: ECAI 2014, IOS Press, 2014, pp. 995–996.

[16] B. Archibald, M. Calder, M. Sevegnani, M. Xu, Quantitative modelling and analysis of bdi agents, Software and Systems Modeling 23 (2024) 343–367.

[17] M. Xu, T. Rivoalen, B. Archibald, M. Sevegnani, can-verify: A verification tool for bdi agents, in: International Conference on Integrated Formal Methods, Springer, 2023, pp. 364–373.

[18] P. Stringer, R. C. Cardoso, C. Dixon, M. Fisher, L. A. Dennis, Adaptive cognitive agents: Updating

action descriptions and plans, in: European Conference on Multi-Agent Systems, Springer, 2023, pp. 345–362.