THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

On intelligent automation systems

Methods for preparation, control, and testing

Endre Erős

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2024

On intelligent automation systems

Methods for preparation, control, and testing

Endre Erős

ISBN: 978-91-8103-003-7
Series number: 5461
ISSN: 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000
www.chalmers.se

Printed by Chalmers Reproservice
Gothenburg, Sweden, January 2024

Great relationships don't just happen, they are built.

On intelligent automation systems

Methods for preparation, control, and testing

Endre Erős
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Developing automation systems that are capable of handling dynamic and unpredictable situations is a challenging task, as it requires adapting to a changing environment and managing potentially unforeseen action outcomes. In contrast to traditional automation, where control code is explicitly programmed, a model-based approach might be a more appropriate solution for automating such systems. Such an approach allows for integrating planning algorithms, which can enable the generation of control sequences that consider the system's state. This capability is essential in enabling human-robot collaboration and handling error recovery and restart. We refer to such a model-based and goal-oriented approach to automation as Intelligent Automation Systems (IAS). To bridge the gap between research and practical utilization, this thesis aims to facilitate the development of IAS by investigating methods for their preparation, control, and testing.

A framework for preparation and virtual commissioning of IAS is presented, which compiles the necessary methods into a high-level structure, aiming to streamline the IAS development process. As part of the preparation process, an effort to explain the unsolvability of some planning problems by localizing potential faults in behavior models is presented. Furthermore, this thesis investigates planning and SAT solving methods aimed at improving the efficiency of planning, thereby enhancing the responsiveness and adaptability of IAS. A planning and execution framework for IAS is presented, with a focus on handling dynamic and unpredictable systems. Finally, an iterative method for the verification of IAS is presented, where methods such as supervisory control theory, model checking, unit and integration testing, and property-based testing play key roles in ensuring the correct behavior of IAS. Connected to verification, a criterion for assessing the test coverage of IAS is presented.

This research contributes to the field of intelligent automation by providing solutions for the development, control, and verification of systems designed for complex and unpredictable environments, aiming to bridge the gap between theory and practice.

Keywords: Automation, virtual preparation and commissioning, automated planning, modeling, robot operating system, testing and coverability.

List of Publications

This thesis is based on the following publications:

[A] **Endre Erős**, Martin Dahl, Kristofer Bengtsson, Petter Falkman and Knut Åkesson, "Virtual preparation and commissioning of ROS2 based intelligent automation systems". Submitted for possible journal publication.

[B] **Endre Erős**, Kristofer Bengtsson and Knut Åkesson, "Fault localization for intelligent automation systems". IEEE International Conference on Emerging Technologies and Factory Automation vol. 29, pp. 1-8, ETFA 2023.

[C] **Endre Erős**, Martin Dahl, Petter Falkman and Kristofer Bengtsson, "Evaluation of high level methods for efficient planning as satisfiability". IEEE International Conference on Emerging Technologies and Factory Automation vol. 26, pp. 1-8, ETFA 2021.

[D] Martin Dahl, **Endre Erős**, Kristofer Bengtsson, Martin Fabian and Petter Falkman, "Sequence Planner: A Framework for Control of Intelligent Automation Systems". Applied Sciences vol. 12, 12(11):5433. MDPI 2022.

[E] **Endre Erős**, Kristofer Bengtsson and Knut Åkesson, "Structural coverability for intelligent automation systems". IEEE International Conference on Automation Science and Engineering vol. 19, pp. 1-6, CASE 2023.

Other publications by the author, not included in this thesis, are:

[F] **Endre Erős**, Martin Dahl, Atieh Hanna, Per-Lage Götvall, Petter Falkman and Kristofer Bengtsson, "Development of an Industry 4.0 demonstrator using Sequence Planner and ROS2". Robot operating system (ROS): The complete reference, vol. 5, pp. 3-29, Springer 2021.

[G] **Endre Erős**, Martin Dahl, Petter Falkman and Kristofer Bengtsson, "Towards compositional automated planning". IEEE International Conference on Emerging Technologies and Factory Automation vol. 25, pp. 416-423, ETFA 2020.

[H] **Endre Erős**, Martin Dahl, Atieh Hanna, Anton Albo, Petter Falkman and Kristofer Bengtsson, "Integrated virtual commissioning of a ROS2-based

collaborative and intelligent automation system". IEEE International Conference on Emerging Technologies and Factory Automation, vol. 24, pp. 407-413, ETFA 2019.

[I] **Endre Erős**, Martin Dahl, Kristofer Bengtsson, Atieh Hanna and Petter Falkman, "A ROS2 based communication architecture for control in collaborative and intelligent automation systems". Procedia Manufacturing, vol. 38, pp. 349-357, 2019.

[J] Martin Dahl, Christian Larsen, **Endre Erős**, Kristofer Bengtsson, Martin Fabian and Petter Falkman, "Interactive formal specification for efficient preparation of intelligent automation systems". CIRP Journal of Manufacturing Science and Technology, vol. 38, pp. 129-138, 2022.

[K] Martin Dahl, **Endre Erős**, Atieh Hanna, Kristofer Bengtsson, Martin Fabian and Petter Falkman, "Control components for Collaborative and Intelligent Automation Systems". IEEE International Conference on Emerging Technologies and Factory Automation vol. 24, pp. 378-384, ETFA 2019.

[L] Martin Dahl, **Endre Erős**, Atieh Hanna, Kristofer Bengtsson, Petter Falkman, "Sequence Planner - Automated Planning and Control for ROS2-based Collaborative and Intelligent Automation Systems". arXiv cs.RO, 2019.

[M] Atieh Hanna, Kristofer Bengtsson, Martin Dahl, **Endre Erős**, Per-Lage Götvall and Mikael Ekström, "Industrial Challenges when Planning and Preparing Collaborative and Intelligent Automation Systems for Final Assembly Stations". IEEE International Conference on Emerging Technologies and Factory Automation, vol. 24. pp. 400-406, ETFA 2019.

[N] Elinor Jernheden, Carl Lindström, Rickard Persson, Max Wedenmark, **Endre Erős**, Sabino Francesco Roselli and Knut Åkesson, "Comparison of Exact and Approximate methods for the Vehicle Routing Problem with Time Windows". IEEE International Conference on Automation Science and Engineering vol.16, pp. 378-383, CASE 2020.

Acknowledgments

This research is supported by *Chalmers University of Technology*, *Volvo GTO*, *Swedish Research Council* (VR) under the project SyTeC (contract nr. 2016-06204), and *Vinnova* under the projects UNIFICATION (contract nr. 2017-02245), UNICORN (contract nr. 2017-03055), AIToC (contract nr. 2020-01947), and AIHURO (contract nr. 2022-03012).

Over the past five years, I had the pleasure of experiencing different supervision styles from several mentors. Thank you Kristofer Bengtsson for your time and effort, which you have spent plenty of, working and coding together with me. Thank you Knut Åkesson for your patience and dedication. Taking on the supervision of a fourth-year PhD student is no small feat, but you have managed the challenge exceptionally well. Thank you Petter Falkman for your words and sincerity, you always know what to say to encourage and inspire. I am very grateful to all of you.

Thank you Martin Fabian for being a great boss. Due to our DK meetings, study circles, social events, and Hindås days, our Automation group transcended the typical definition of colleagues. Thank you Bengt Lennartson for your courses, and for stopping by my office from time to time for a chat, it was always a pleasure. Thank you Christine Johansson for always being helpful.

Thank you Atieh Hanna and Per-Lage Götvall for our rewarding collaboration over the past seven years, and thank you for accepting me at AB Volvo all those years ago. It has been a turning point in my life.

I am grateful to my colleagues who provided a stimulating academic environment. The coffee breaks, lunches, and afterworks, wouldn't be the same without you Ludvig, Julius, Sabino, Constantin, Zahra, Mattias, Alvin, Ze, Kristian, Remi, Ektor, Ahmet, Ashfaq, Adnan, Ramin, Stefan, Carl-Johan, Nishant, Rita, Gabriel, and Erik.

A special thanks to Martin Dahl, Anton Albo, and Rikard Karlsson, for being such good people and the best colleagues one can hope for. I have learned so much from you over the years, and I still hope to do so in the years to come. Thank you Gabrijela, Anna, Felix, Elin, Lucia, Carlos, and Chiara for all the good times we had together. I am looking forward for more!

To my friends back home, Elena, Mina, Jovana, Aleksandra, Kristina, Jovana, Emil, Stefan, Vladimir, Atila, Aleksandar, Ivan, Miloš, Janko, Miško, Luka, and Milan, I am grateful for having you in my life and for growing up with you. Many of you have been a part of my life for nearly three decades,

and I look forward to treasure our friendship for many more decades to come.

Thank you Erika, Zoltán, Dávid, and Oszkár for your unwavering support and encouragement. It has been a source of strength throughout this journey. I am forever grateful. I extend my appreciation to my extended family, Zita, Siniša, Zoltán, Ana, Andrej, Marta, Alen, Andrej, Barbara, Brigita, Mária, and Rozália, thank you for standing by me with your kindness.

Nastasja, this thesis is a testament to our shared journey. Through its highs and lows, it has solidified into a reminder that our path is now set on a positive trajectory. It invites us to embrace the promising future ahead. We have spent many days together, and I look forward to spending the rest of my days with you, loving you, and growing with you. I am eternally grateful for the love and warmth you have brought into my life, and I am excited about the future chapters that we will write together.

Acronyms

IAS:	Intelligent Automation System
SP:	Sequence Planner
ROS:	Robot Operating System
VC:	Virtual Commissioning
PLC:	Programmable Logic Controller
SAT:	Propositional Satisfiability
SMT:	Satisfiability Modulo Theories
SUT:	System Under Test
DT:	Digital Twin
AMR:	Autonomous Mobile Robot
AGV:	Automated Guided Vehicle
LTL:	Linear Temporal Logic
DES:	Discrete Event System
SCT:	Supervisory Control Theory
MC:	Model Checking
BMC:	Bounded Model Checking
MC/DC:	Modified Condition/Decision Coverage
PBT:	Property Based Testing
PDDL:	Planning Domain Definition Language
CDCL:	Conflict Driven Clause Learning
EFA:	Extended Finite Automata

Contents

Part I

Overview

CHAPTER 1

Introduction

Final assembly and material handling are some of the sectors in discrete production that face challenges such as shorter product lifetimes, just-in-time manufacturing, product customization, increased product complexity, and variation in demand. These challenges can partially be alleviated [1] by introducing Intelligent Automation Systems (IAS), which can change the pace of production, adapt to different products and manufacturing scenarios, react to dynamic environments, and enable safe human-robot interaction.

To practically address these challenges, a solution involves enabling the cooperation of Autonomous Mobile Robots (AMRs), collaborative robots, and human operators within a shared workspace. Within this shared environment, AMRs are responsible for transporting assembly equipment and parts to their designated destinations at the appropriate times, while collaborative robots are assigned to perform assembly and kitting tasks, and are able to work alongside human operators. The operators are tasked with handling intricate tactile operations, complex restart scenarios, and they have the flexibility to move between workstations to balance the workload. Because of the collaborative setting and the nature of tasks that are performed, such systems can be characterized as *dynamic* and *unpredictable*.

In a *dynamic* system, external input can independently change the state of the system. This state change can occur during the execution of tasks, which can make the current sequence of tasks obsolete. For example, an AMR might not be able to pass through an assembly station because of a temporary obstruction, or a human operator can independently perform a task that is not planned by the control system.

In an *unpredictable* system, the tasks performed can have different outcomes. For example, a common task like grasping can often be unsuccessful, resulting in failures such as failing to pick up a part, dropping a carried part, picking up two parts instead of one, colliding on the way to the part, etc.

To successfully manage a *dynamic* and *unpredictable* system, a shift in the automation approach is required. Rather than relying on explicitly programmed control code and manually defined sequences, adopting a model-based approach can be beneficial. Such an approach focuses on declaring *what* a system can do rather than specifying *how* it should be done, contrasting traditional control code programming with *if-then-else* statements. We refer to this model-based and goal-oriented approach as *intelligent automation*, which can be complemented with methods such as automated planning, verification, and virtual commissioning.

Many initiatives, both in academia and industry, have pushed for smarter automation, however, these initiatives often lack clear guidance regarding practical implementation. Developing and deploying such automation systems is a challenging task, and there is often a discrepancy between methods developed in academia and those applied in industrial settings.

The lack of a structured development method for IAS that addresses both the necessary high-level steps and the low-level implementation challenges is identified as an important gap between research and utilization. To address this disparity, this thesis presents a synthesis of methods aimed at facilitating the development of IAS. These methods are inspired by, or are relying on, established methods such as virtual commissioning (VC) [2], automated planning, and verification, and can be grouped into:

1. Preparation and virtual commissioning

2. Planning and execution

3. Verification and coverability

In the following sections, we discuss the state of art, industrial practice, and the research contributions to each of these groups. Since this thesis consists of included papers, we will make references to the included papers below. These included papers are introduced and summarized in Chapter 5.

1.1 Preparation and virtual commissioning

Production *preparation* involves preparing a production system for manufacturing by selecting processes, tools, interfaces, and standards for a specific production scenario [3]. To simplify the preparation process, part of it can be done virtually, by using *simulations* and simulation supported automation engineering called *Virtual Commissioning* (VC) [2].

The purpose of VC is to expose the control system to simulations, in order to enable the control logic to be verified in a virtual environment before physical commissioning. The overall goal of performing VC is to reduce the implementation time and cost by finding potential faults before the system is physically implemented, as seen in Figure 1.1.

For VC to be effective across various projects, it should be applied during all development stages by all relevant personnel, and not be restricted to specialists or niche researchers. To achieve that, the engineering workflow should include the development of a *Digital Twin* (DT), which is essentially a virtual representation of a corresponding physical entity. Beyond VC, such DTs can be used for conceptual development, online diagnostics, virtual sensors, predictive maintenance, and more. In the next section, we present the state of art for VC.

State of the art

VC has the potential to increase production engineering efficiency while also reducing on-site implementation time [2]. Although it forms the foundation for state-of-the-art production systems, it is not yet widely used [4]. The following text provides an overview of current scientific approaches and use cases for VC.

In [5], a methodology based on VC is presented for integrating DTs into manufacturing systems, where the potential of DTs to enhance manufacturing efficiency and competitiveness in dynamic conditions is demonstrated.

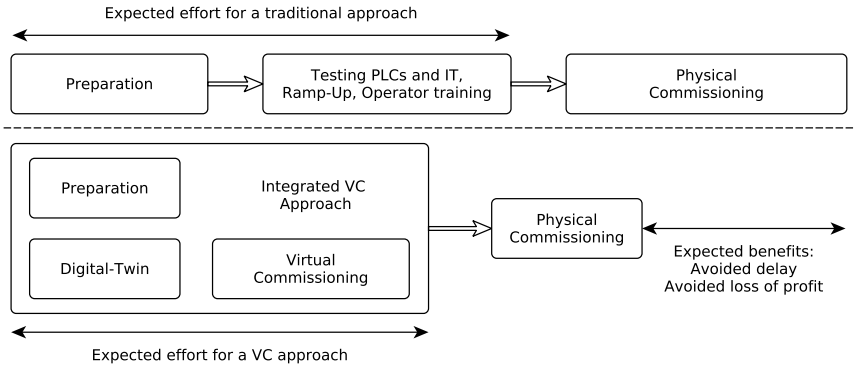

Figure 1.1: A comparison between a traditional implementation approach, and an approach that includes VC. It can be seen that the traditional approach has a higher expected implementation effort and a longer physical commissioning phase compared to an approach that includes VC.

To address the need for comprehensive behavior models for DTs, [6] introduces a real-time co-simulation platform for VC. This platform integrates technology-specific simulation solutions and employs real-time co-simulation techniques, including partitioning, parallelization, synchronization, and data exchange mechanisms.

In [7], challenges and strategies surrounding the implementation of VC in the context of the ongoing industrial revolution are discussed. The paper emphasizes the importance of coordination across organizational layers and technical disciplines, both internally and externally. It highlights the role of standards and common language practices in establishing sustainable business models and reducing lead time, commissioning time, and technical issues.

An approach to enhance production management by focusing on virtual-physical convergence and information integration in manufacturing is presented in [8]. The approach introduces DTs as a key technology to achieve this integration. The study also demonstrates the use of AutomationML through a parts machining cell model.

For implementing DTs in manufacturing, [9] introduces an implementation framework that leverages container technology for self-contained packaging

and uses cloud services to externalize intelligence, achieving scalability and plug-and-play functionality.

In [10], an architecture for DT implementation for real-time monitoring and evaluation of large-scale smart manufacturing systems is presented. [11] utilizes DT technology to establish a virtual prototype of computerized numerical control machine tools. The paper introduces a DT-based VC method that supports dynamic commissioning and kinematic commissioning functions to detect potential errors in the servo system and numerical control programs.

Similarly, [12] presents a DT-driven VC method for computerized numerical control machine tools. This approach constructs a DT model using a multi-domain unified modeling language and a virtual-real mapping strategy to describe response characteristics.

To facilitate model-driven engineering for distributed Cyber-Physical Systems (CPS) using the Robot Operating System (ROS), [13] introduces an approach to automatically generate embedded software, aiding rapid prototyping and system component implementation in complex industrial systems.

[14] explores the use of model-driven engineering to transform production system models into flat files compatible with general-purpose planning tools. This enables the computation of plans and their integration back into the production system model to enhance formalized knowledge.

Moreover, in [15], a model-based approach using Petri-nets [16] is presented to represent control structures in virtual production models. This method aims to enhance VC by providing a clear and interpretable representation of control-related behavior models.

Lastly, [17] discusses the integration of tools such as model checking for off-line verification of controller specifications and VC for on-line functional testing. The paper also highlights the role of tools like PLC Checker and guidelines such as PLCopen in enhancing standardization, readability, reliability, and modularity of PLC code. Efforts like [17] resonate well with the methods presented in this work.

Industrial practice

In the past, VC was primarily accessible to niche experts. However, today's powerful modeling tools and compatible information standards have made VC available to manufacturers of all sizes.

One of the initial challenges in VCs was bridging the gap between different

models and connecting simulation tools from different domains. Traditionally, physical plant simulations occurred on separate platforms from the logic-based PLC design systems. This led to the creation of the Functional Mock-up Interface (FMI) [18] standard, which aimed to enhance the exchange of simulation models between suppliers and original equipment manufacturers (OEMs). The FMI facilitates seamless import and export across various software tools. Advanced tools can now create DTs and export them as Functional Mock-up Units (FMUs) with the required data for automation tools. With widespread support for the FMU standard, engineers can now connect other FMUs with simulation tools and enabling them to connect with PLC code.

This allows virtual models and virtual PLC code to coexist in a unified environment, streamlining testing procedures. Engineers may even consider omitting specific physical sensors in the final product, relying on the DT to provide the necessary data. Some well known discrete even simulation and VC software are, for instance, WinMOD®, Technomatix® by Siemens, and Delmia® by Dassault Systems, however, there are many more tools like this on the market.

Problem and motivation

Based on a performed literature review in Paper A, an identified research gap is the lack of practical implementation insights and specific steps required to develop and deploy IAS. This research gap hinders the effective utilization and implementation of VC for model-based automation systems, and there is a need for comprehensive guidelines and procedures to bridge the gap between research and practical application.

The motivation for this part of the work is the need to streamline the implementation of IAS. Because of the dynamic and unpredictable nature of systems addressed by IAS, explicitly programming control code and performing virtual verification for mostly topologically static and fully automatic robotic cells is not an appropriate method for their preparation and VC. Instead, the preparation and VC of IAS entails the use of behavior modeling, planning, scheduling, control, and testing algorithms, as well as virtual modeling, preparation, commissioning, and verification tools which are not yet fully integrated in the automation systems development tool-chain.

Most of these insights and lessons learned originate from developing industrial demonstrators and solving the surfacing challenges. One of the re-

curring challenges during such implementations is potentially overlooking or incorrectly modeling behavior and constraints. This can result in unsolvable planning problems or plans that are invalid for other reasons. When plans are unsolvable, developers receive no feedback, which makes the model preparation and adjustments a difficult and time-consuming task. In some cases, a fault localization technique can be applied to explain why a problem was unsolvable.

Research question and contribution

Based on the research problem discussed above, we formulate the first research question:

RQ1 *How can the development of IASs be supported by methods such as preparation, virtual commissioning, and fault localization, in order to facilitate the adoption of model-based automation?*

Due to the lack of structured frameworks for preparation and virtual commissioning, there is a lack of integrated methods which can support the development of virtual and behavior models in IAS. This hinders the effective utilization and implementation of VC, and there is a need for comprehensive guidelines and procedures to bridge the gap between research and practical application.

Contribution

A preparation and VC framework for IAS, inspired by the traditional VC methodology is presented. This framework relies on a range of techniques, such as testing, automated planning, formal methods, reachability analysis, code generation, and resource simulations. This framework presents an iterative and structured approach to apply these techniques to ensure efficient implementation of IAS.

However, faults can sometimes be introduced during the preparation of IAS. The types of faults that Paper B focuses on are modeling faults which result in unsolvable planing problems. As these results give no feedback to developers, except of the fact that finding a plan has failed, this paper presents algorithms which can in some cases simplify fault localization in behavior models by identifying and suggesting suspicious variables and operations.

1.2 Planning and execution

Because of the dynamic and unpredictable nature of IAS, many tasks can have several outcomes. Executing sequences can fail, and planned sequences can become obsolete mid-execution. One way to handle this is to model the behavior of IAS using non-deterministic models [19], which can express different outcomes a task can have.

For example, the action of a dice throw can have six possible outcomes, where none of the outcomes can be considered nominal [19]. Such behavior can be captured with non-deterministic models, however, such models are not perfect models of the world. Even if we model the six outcomes of the dice throw, the thrown dice can run off the playing board and end up under the table, or the dice can hit a pen on the table and end up stopping on its edge.

The meaning of this story is that we can never fully capture all the possible outcomes of real-world actions. Instead of trying to describe how an action could fail with non-deterministic models, the models describing the behavior of IAS are *deterministic*. Such models do not express the different possible outcomes an action can have, implying that no randomness is involved in the development of future states of the system. This design choice allows us to avoid models which can become challenging to handle, both conceptually and computationally. However, this design choice strongly depends on the ability of the control system to quickly re-calculate sequences of operations based on constant observations of the current state of the system. Such calculations can be made by automated planning algorithms.

Automated *planning* is a deliberative decision making process which yields sequences of actions that drive state change towards a goal [19], and it is the deliberative *core* of IAS. Using planning in conjunction with acting, the control system is able to automatically adapt and re-plan based on the current state, virtually embedding the ability to restart. A control system based on automated planning is model-based and goal-oriented, always trying to calculate the necessary sequence of actions to reach the goal.

Such sequences are *executed* by an execution algorithm, which communicates with the system's resources, receives constant sensor data, evaluates guards, and executes actions to follow the planned sequence. Execution also takes care of synchronizing the state with the resources and the DT, initiating re-planning when necessary, and blocking execution when certain safety conditions are not met. The high-level planning and execution architecture,

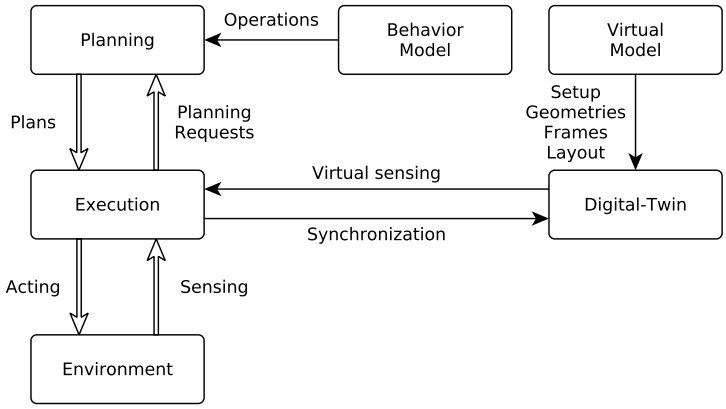

Figure 1.2: The high-level planning and execution architecture of IAS.

which is used in this work is shown in Figure 1.2.

In this work we make use of Robot Operating System 2 (ROS2) [20] to implement the planning and execution framework. This design choice is motivated by its communication protocol, easily distributable nodes, supporting simulation and visualization tools, and years of documented experiences and examples by the ROS community. In the following text, we summarize the current state of the art in planning and execution frameworks for IAS.

State of the art

While ROS can give support with regards to integrating different software libraries and drivers, as well as provide a base layer for communication, challenges remain. One big challenge is the coordination of different devices, including cameras, PLCs, robotic arms, and AGVs, especially when uncertain human behavior needs to be taken into consideration.

In the literature, several frameworks have been proposed that aim to aid in planning and execution of robot actions. One example is ROSPlan [21] that uses PDDL-based models for automated task planning as well as handling plan execution. Another is SkiROS [22] which is based on defining an ontology of skills to help design and execute the high-level skills of an hybrid behavior-deliberative architecture. MaestROB [23] adds natural language processing

and machine learning to teach robots skills that are executed using ontology based planning, CoSTAR [24] is based on Behavior Trees that are used to manually define complex behavior, combined with a way of defining computer perception pipelines, and eTaSL/eTC [25] which defines a task specification language based on constraints. Applications that use planning of robot skills have seen successful experimentation in industrial settings [26], [27].

A Belief-Desire-Intention (BDI) toolkit based on ROS2 is presented in [28], showcasing its effectiveness in a scenario where autonomous robots cooperate to sort and move boxes. This toolkit combines BDI architecture with integrated planning, enabling adaptable agents to collaborate efficiently in dynamic environments. A model-based approach for automatically generating executable C++ code for ROS is presented in [29]. The approach involves three phases: modeling robot behaviors as timed automata, formalizing and verifying safety requirements, and generating executable code.

A plug & produce system using standardized Open Platform Communications Unified Architecture (OPC UA) skills for industrial workcells is introduced in [30]. By extending OPC UA discovery services, the system adapts to component changes, demonstrating reduced changeover times in robot-based assembly, particularly beneficial for small lot production.

Middleware for Intelligent Automation (MIA), a platform addressing Industry 4.0 challenges is presented in [31]. MIA acts as a bridge between field devices, databases, and Decision Support Systems (DSSs), ensuring real-time data management and interoperability.

A software architecture for autonomous service robots operating in human environments is presented in [32]. It integrates deep learning-based perception, knowledge representation, symbolic task planning, and motion planning, to perform tasks by a real robot without human intervention.

A decentralized framework for multi-robot task and path planning is presented in [33]. It uses Markov Decision Processes (MDPs) and Mixed Observed Markov Decision Processes (MOMDPs) to model tasks and employs the max-sum algorithm for decentralized task allocation.

A skill-based architecture is compared to a traditional component-based hierarchical approach for flexible manufacturing systems in [34]. The evaluation criteria includes execution time, modularity, readability, and reusability. The study finds that the skill-based architecture, though more complex, offers greater flexibility, making it suitable for adaptable production plants.

A framework for integrating collaborative and roaming robots into industrial automation alongside human operators is presented in [35]. It is based on model-based design and control, where interactive formal specification enable efficient preparation of IAS.

A ROS2 based planning system that converts plans into Behavior Trees (BTs) to enhance plan execution in mobile service robots is presented in [36].

Finally, PlanSys2 is presented in [37], which is a symbolic planning framework designed for autonomous robots operating in complex environments. PlanSys2 is built on ROS2, and offers features like BTs for efficient plan execution and a novel action auction protocol for multi-robot planning.

However, these systems listed here are mainly robot-oriented (in contrast to automation-oriented) and often focus on a single robot. It seems that a framework for combining both high-level robotic tasks with more traditional automation tasks (low-level execution and state management of a variety of different devices) is missing.

Industrial practice

Over the last two decades, many methods have emerged with the aim to improve the efficiency of automated planning, and satisfiability (SAT) solving [38]. The frontiers of automated planning and SAT solving are improved and tested on a yearly basis during the international planning [1] and SAT [2] competitions. Additionally, by coupling a SAT solver with theory solvers, for example theories such as linear arithmetic, bit vectors, or arrays, Satisfiability Modulo Theory (SMT) based planning techniques [39] can encode and tackle real-world scenarios and complex application domains [40].

Methods developed in academic context are not always found in industry. However, many tools and methods are becoming available to manufacturers of all sizes. Planning, synthesis, model checking, and optimization tools are increasingly more present in the industry, for example, NuXmv [41], Z3 [42], Supremica [43], Uppaal [44], Kissat [45], Gurobi, Cplex, and Xpress [46], Madagascar [47], and MiniSat [48].

[1] https://www.icaps-conference.org/competitions/
[2] http://www.satcompetition.org/

Problem and motivation

Current trends in automation involve the integration of collaborative robots and AMRs into workflows, often working alongside human operators to create more adaptable automation solutions. However, such an automation system must possess the capability to anticipate and respond to both its environment and the actions of individual subsystems.

Tasks which handle perception and grasping tasks, may not guarantee a 100% success rate [49], [50], [51], leading to an increase in expected unsuccessful operations. Effectively managing these failures as a natural part of automation introduces complexity to the software. To handle such complexity, difficult modeling tasks can be off-loaded to control logic synthesis algorithms and the specifics of execution to an online planning system.

Consequently, real-time algorithms for task planning need to be sufficiently efficient to be applicable in the industry. Since planning efficiency can contribute to making a system adaptive and responsive, investigating planning methods and decision-making choices is necessary to determine the most appropriate methods for specific situations. Such insights can then be used to design or adapt automated planners and solvers for specific problems.

Research question and contribution

Based on the problems raised above, we can formulate the second research question that this thesis will attempt to answer:

RQ2 *How can the planning and execution processes in IASs be implemented, in order to ensure reactive and adaptive systems?*

To deliver a robust solution, the automation system must possess the capability to both anticipate and react to the actions of the environment, which means that online planning algorithms have to be a part of the automation system. Moreover, such planning algorithms have to be able to quickly calculate new plans, allowing the automation system to re-plan and adapt to the changing environment when necessary.

Contribution

In IAS where control sequences are constantly calculated, fast re-planning plays a key role. To assess which algorithms can enhance planning perfor-

mance, paper C delves a bit deeper into planning, in order to assess and compare various high-level modeling and planning methods using a set of standard benchmarks. This paper focuses on planning as satisfiability, which is considered the leading approach for solving challenging planning problems.

In order to have responsive control in an IAS, the execution system has to be able to react quickly. Paper D presents a modeling, planning, and execution system, called Sequence Planner (SP), which is built with a hierarchical architecture, making it easier to handle larger systems in comparison to our previous non-hierarchical attempts. SP enables responsiveness with fast replanning, which is achieved by dividing the planning work into two levels.

1.3 Verification and coverability

Verification is a family of methods whose goal is to assure that systems satisfy the expected requirements. It is a crucial element in the development process of control software and plays a key role in ensuring the reliability, functionality, and performance of automation systems. During different stages of development, parts of the system can undergo formal verification which is used to prove that such parts adhere to the specified requirements. For example, in model checking [52], temporal properties are verified by exploring the state space using a set of initial states and transitions. The goal of model checking is to prove that a user defined temporal specification always holds. If it cannot be proven, a counterexample is produced.

Although formal verification can benefit subsystems in automated systems, the behavior model only captures certain discrete deterministic behavior of the system and does not consider the details of dynamics, robot motions, uncertainties, communication, latency, timeouts, failures, error handling, restarts, etc. Such a lack of formal models makes it generally impractical to use model checking to verify a complete implementation.

Another way to ensure that the system behaves as intended is to apply some form of *testing*, while measuring the coverage of such tests to ensure that an adequate portion of the system has been exercised. Although testing can be costly, it doesn't depend on formal models and it can be applied in real conditions such as on the target hardware or operating system. It also is important to note that formal verification and testing complement each other, and that both methods should be applied when possible.

The development and testing process of IAS can be represented with an interpretation of the V-model from [53], as seen on Figure 1.3. Starting from the left side with the *Concept Development Phase*, the high-level system requirements are defined, which describe the project objectives and guide the development and verification process. In the *Constructive Phase*, concrete software components are developed such as the virtual model, the DT, and various drivers, interfaces, and controllers.

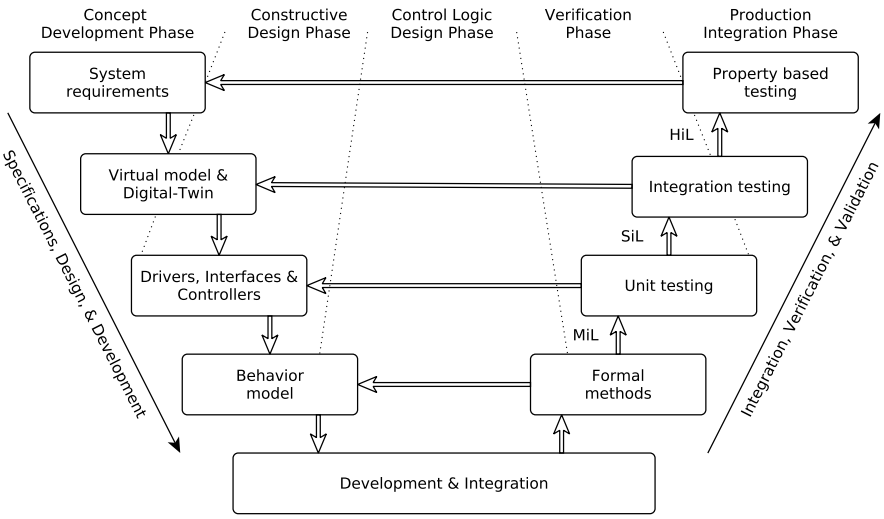

Figure 1.3: A high-level model-based development and verification method for IAS based on the V-model from [53].

The behavior model is developed in the *Control Logic Design Phase*, during which certain parts of the model can be subjected to formal methods. If the formal verification process produces a counterexample which violates a certain specification, the behavior model can be re-iterated. This phase of part of the *Verification Phase* can be sometimes referred to as Model-in-the-Loop (MiL), since the verification if performed directly on the behavior model.

When a software component (*unit*) is created, for example a driver, interface, or controller, the next step is to create unit tests to assert that the component works as expected. After several components have been created, they can be tested in a virtual environment to verify that the interface and

interaction between the components work as expected. This phase is referred to as integration testing. Both unit and integration testing can be performed virtually, using a virtual model or a DT, i.e. Software-in-the-Loop (SiL).

After the developed components have been virtually verified, hardware components can gradually be included in the verification procedure, which can then be referred to as Hardware-in-the-Loop (HiL) verification. In the final stage of the presented model, certain system-level properties can be tested, after which the system is ready for the *Production Integration Phase*.

State of the art

Various studies investigate the challenges of testing in an industrial setting. For example, [54] investigates the challenges of testing robotic systems, while [55] presents the practical insights and lessons learned for automated testing of industrial automation software. Additionally, [56] discusses test case generation approach for industrial automation systems.

A review of simulation-based approaches for verification of embedded control systems is given in [53], with an overview of of traditional and advanced modeling, testing, and verification techniques. Additional state of the art reviews for software engineering and testing in an industrial automation setting can be found in [57] and [58].

For ROS-specific systems, a notable verification framework is the High-Assurance ROS (HAROS) [59]. HAROS uses static analysis to extract models from source code, enabling various analyses like model checking and runtime verification. Other methods can be integrated with HAROS to verify ROS based systems. For instance, a method to automatically generate property-based test (PBT) scripts for ROS configurations is presented in [60]. By applying PBT and the HAROS framework, the method streamlines testing, ensuring dependable robotics software for safety-critical applications.

Another lightweight formal verification technique for assessing system-wide safety properties in ROS-based robotic applications is introduced in [61]. The approach formalizes ROS architecture and node behavior using Electrum, a formal specification language, and integrates seamlessly with HAROS.

Finally, a pattern-based modeling and Uppaal-based verification approach is introduced in [62]. It addresses latency and buffer overflow concerns in distributed robotic systems employing ROS2.

Based on the available verification methods, we are utilizing a four step

verification approach for IAS based on the V-model from [53] as shown on Figure 1.3, which is further discussed in Chapter 4.

Industrial practice

A common practice in the industry is assessing the extent to which the code has been is exercised during testing [63]. One coverage criterion that is widely used in the automotive and aerospace industries, is the Modified Condition/Decision Coverage (MC/DC) [64]. For example, the highest level of safety assurance in the automotive industry, ASIL D in ISO 26262 [65], recommends the use of MC/DC coverage as a criterion for software verification. Naturally, MC/DC has also seen adoption in industrial automation systems [66].

Criteria like MC/DC exist to ensure that the testing provides sufficient coverage of the System Under Test (SUT) [67], otherwise the test has failed to exercise parts of the SUT. High coverage can help minimize the risk of unexpected software failures and ensure that the system performs as expected. Therefore, coverage metrics might be beneficial to assess the adequacy of the testing process and determine if more tests are necessary [63]. The list of available testing tools includes Quickcheck [68], Hypothesis [69], Proptest [70], and Breach [71].

Problem and motivation

The process of testing is a crucial element in the development of control software and plays a key role in ensuring the reliability, functionality, and performance of automation systems [72], [73]. It is essential to implement and utilize a comprehensive testing strategy that can identify software defects before deploying the system [74]. Despite its significance in ensuring system reliability, determining the appropriate level of required testing is a common challenge.

In retrospect, the sprints conducted in the days leading up to the final implementation and demonstration phase underscored the significance of thorough testing and comprehensive test coverage. It became evident that proper testing and assessment of our systems could have mitigated numerous challenges and complications.

Research question and contribution

RQ3 How can IASs be verified with formal methods and testing, and how can the adequacy of such testing be assessed?

Testing is a crucial activity when developing IASs. Combined with a test coverage criteria, it can be ensured that the testing process delivers sufficient coverage of the System Under Test (SUT). Failure to achieve this coverage signifies that the conducted test has not effectively exercised portions of the SUT.

Contribution

Inspired by the modified condition/decision coverage (MC/DC) criterion, paper E introduces an approach to analyze the structural coverability of behavior models for intelligent automation systems. Paired with a testing procedure, this approach allows each test case to influence both the controller and the simulated environment by injecting specific states. Consequently, the proposed coverability criterion can identify segments of the model that require more testing and recommend additional test cases to enhance coverability.

1.4 Research approach

The purpose of research is to generate new knowledge that is valuable both to academia and to current engineering practices. However, deciding *how* to perform research can sometimes be a challenging task, as it depends on the research topic, objectives, scope, available resources, ethical considerations, etc. To address this challenge, a number of research methods have been developed for the engineering design research field. Some of these methods are the Theory of Technical Systems [75], Domain Theory [76] [77], Theory of Inventive Problem Solving [78], Axiomatic Design [79], CK-Theory [80], Function-Behavior-Structure framework [81], and Mathematical Theory of Design [82].

The work presented in this thesis is motivated by industrial challenges, and as such, the presented research is experimental and application oriented. Industrially motivated research typically begins with a real-world problem and gradually evolves to encompass both industrial objectives and research that is academically viable. As such, developing and showcasing demonstrators is

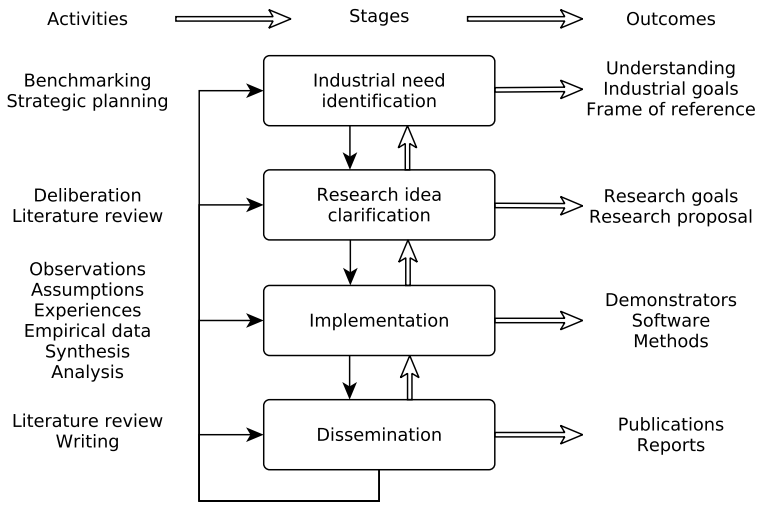

Figure 1.4: Overview of the stages of the research method used.

a crucial activity for bridging the gap between the industry and academia.

The research method used in this work draws inspiration from the Design Research Methodology [83] and it adheres to the Systems Engineering Research Methods [84], as seen on Figure 1.4.

1.5 Industrial use-cases

During the past several years, we have been participating in several projects, out of which the following two are the most relevant to this thesis.

Collaborative engine assembly

This demonstrator is the result of the transformation of an existing truck engine manual assembly station, into a collaborative robot assisted assembly station. The key challenge is on how the tasks are executed, as they can be carried out independently by the robot, the human operator, or collaboratively, see Figure 1.5. Additionally, some of the tools are available to both

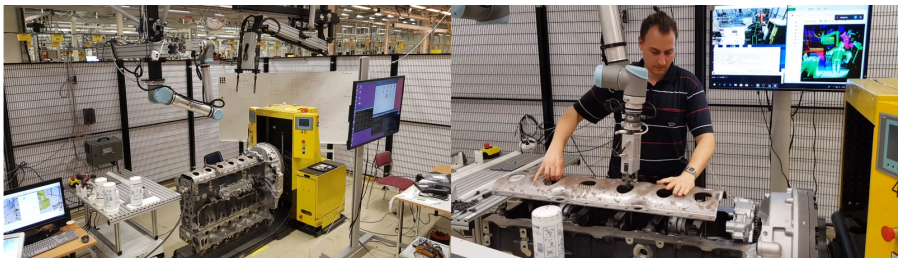

Figure 1.5: The Unification industrial demonstrator.

the robot and operator. To tackle this complex undertaking effectively, the automation system has to calculate a sequence which it can execute together with a human operator, not restricting its movement or compromising safety.

Robot assisted material handling

This demonstrator investigates using a collaborative robot on a gantry to support the operators that use a pick-to-light system to assemble kits onto a trolley, see Figure 1.6. The robot employs can equip itself with a structured light scanner to locate items in blue boxes and selects tools for different tasks. The goal is to enable cooperation between operators and robots in a shared workspace, picking materials and transferring them to an assembly station. Challenges include precise scanning, collision prevention, rapid failure response, and balancing task speed with safety.

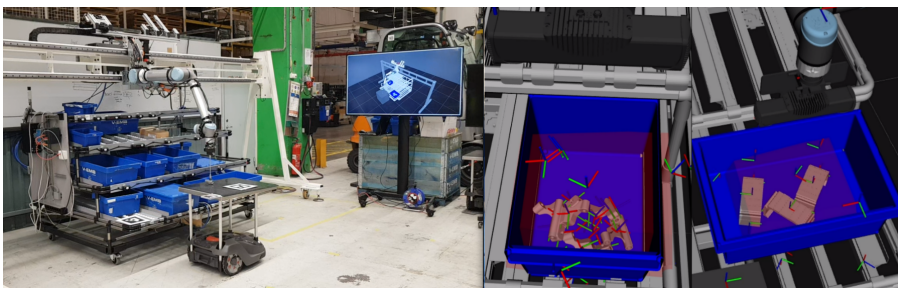

Figure 1.6: The Robot in the air industrial demonstrator.

1.6 Outline

This thesis is comprised of two parts. Part I serves as an introduction to the field, placing the appended papers into context. Part II contains the appended papers. Part I is organized as follows: Chapter 2 contains an introduction to Virtual Commissioning (VC), and a motivation for applying a VC inspired method for developing and deploying IAS. Chapter 3 gives an overview of the modeling, planning, and execution methods for IAS, and discusses planning as satisfiability as a leading method to perform planning. Chapter 4 describes how IAS are verified with formal methods and testing, and how the coverage of such tests can be assessed. The main results of the appended papers, the closing remarks, and directions for future work, are summarized in Chapter 5. Chapter 6 holds a summary of the appended papers.

CHAPTER 2

Preparation and virtual commissioning

The purpose of VC is to enable the control software, which controls and coordinates different devices in a production station, to be tested and validated before physical commissioning. Over the years, the VC community has specified several commissioning configurations [85], [2] that resulted in terms like *Hardware-in-the-Loop*, *Reality-in-the-Loop* and *Constructive Commissioning*, which is also sometimes referred to as *Software-in-the-Loop* or *Model-in-the-Loop* commissioning. As shown in Fig. 2.1, these specifications are defined by combinations of components being real or virtual.

Testing and integrating the physical production system with the real control system has been traditionally referred to as *physical commissioning*. However, in order to reduce the amount of on site man-hours during physical commissioning [86], the real control system is coupled with a simulation model of the production system creating a *Hardware-in-the-Loop* setup. This configuration is commonly known as VC [2].

When designing a new control system, the natural way is to start with offline programming where all components are simulated. This configuration is also known as *Constructive Commissioning* [2].

In cases where debugging the control system is needed, the physical pro-

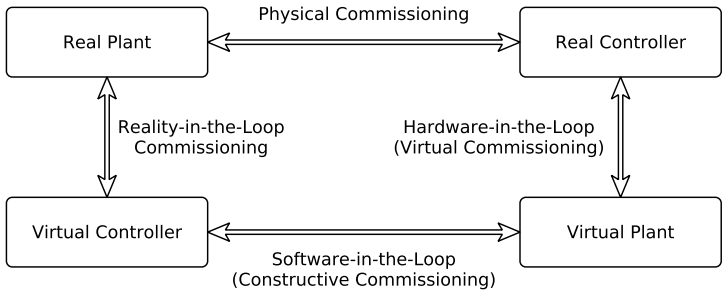

Figure 2.1: Traditional commissioning classification from [2] .

duction system can be controlled by a emulated controller in a setup known as *Reality-in-the-Loop*. In this stage, specific resource behavior can be tested before connecting the actual resources to the controller.

Performing VC can potentially reduce testing and integration time [2], as well as help detect undesired behavior before physical commissioning. However, it is usually the case that creating simulation models requires extensive modelling effort. Because of the cost associated with this effort, it is crucial that the created models provide as much additional value as possible.

Some of the expected benefits of VC are:

1. *Reducing Time-to-Market*: VC allows for the validation and optimization of industrial systems before physical deployment, significantly shortening the time it takes to bring products and systems to market.

2. *Minimizing Costs*: VC helps identify and rectify design flaws, system errors, and operational inefficiencies early in the development cycle. Consequently, it reduces the need for expensive on-site modifications and troubleshooting during physical commissioning.

3. *Enhancing Safety and Reliability*: Through exhaustive testing and simulation, VC contributes to the enhanced safety and reliability of industrial systems, reducing the risk of accidents and downtime once systems are operational.

4. *Bringing what-if scenarios to life*: VC facilitates comprehensive testing scenarios, including normal operation, fault simulations, and emergency situations, ensuring that systems are thoroughly validated before deployment.
5. *Accurate Predictions*: DTs and advanced simulation software provide highly accurate predictions of system behavior, enabling engineers to foresee potential issues and make informed decisions.
6. *Innovation Acceleration*: VC fosters innovation by allowing engineers to experiment with new ideas and configurations without the constraints of physical prototypes, driving the development of new technologies.
7. *Reduced Environmental Impact*: Fine-tuning and optimizing systems in a virtual environment can lead to more energy-efficient operations, reducing an organization's environmental footprint.
8. *Training and Skill Development*: VC serves as an effective platform for training operators and maintenance personnel, enabling them to gain hands-on experience with complex systems in a controlled setting.

Simulation of production systems is a well adopted practice, and using methods for designing control systems and performing VC has become a standard in industry. Because of the lack of practical implementation insights and specific steps required to develop and deploy IAS, the following two sections introduce two contributions that aim to address this gap. A structured development method for IAS which discusses the necessary high-level steps, as well the low-level implementation challenges is presented in paper A, and a fault localization support system for detecting modeling errors during development is presented in paper B.

2.1 Preparation and commissioning of IAS

Here we summarize paper A, which presents a framework for virtual preparation and commissioning of IAS. The framework is based on iteratively developing and using a DT representation of the demonstrator for different activities, such as early simulation, reachability analysis, behavior model development, commissioning, and control. The effectiveness of this framework is exemplified by developing an industrial demonstrator.

The Robot Operating System

Systems based on ROS rely on a set of *nodes* which communicate between themselves in order to achieve a desired collective behavior. Wrapping code into nodes which execute different specific tasks, enables users to maintain a good structure, as well as to distribute code on different machines. For example, if a specific piece of software has to run on a dedicated machine, wrapping it in a ROS node will enable this software to be integrated into the network and used in conjunction with other software and ROS based tools like *tf*[87], *moveit*[88] and *rviz*.

The framework presented in paper A is based on the ROS2 [20], and it is implemented in Rust [89] using the async Rust bindings for ROS2 [90]. In this framework, the control algorithms, resources, and corresponding drivers are distributed across a set of computational *nodes*. In order to achieve control over the system, these nodes must communicate with each other through a network, utilizing message-passing protocols. The framework presented in here is connected to our previous work from [91].

The Framework

Now we introduce a framework, which aims to facilitate effective preparation, implementation, and commissioning of IAS for discrete event control. The framework encompasses several key components:

1. *Virtual Model*: The virtual model represents the system's arrangement, capturing information about resource geometries, frames, topologies, and hierarchies.

2. *Behavior Model*: The behavior model defines the possible system behaviors using variables, transitions, and operations, and can include invariants and specifications to constrain these behaviors.

3. *Planner*: The controller incorporates a planner algorithm that calculates sequences of operations necessary to move the system from the current state to a desired goal state.

4. *Controller*: The controller node communicates with the system's resources, evaluates guards, and executes actions to follow the planned sequence.

5. *Drivers*: Driver nodes act as local controllers for the real and simulated resources, facilitating communication between the main controller and the resources.

6. *Dummies*: Dummy control nodes are used to test specific resource behaviors during the constructive commissioning and reality-in-the-loop stages.

7. *Emulators*: Emulator nodes capture very simple, interfacing behavior of real resources and their drivers, aiding in the initial phases of virtual commissioning.

8. *Simulators*: Simulators are software tools that simulate resources, capturing their internal behavior to some extent. They can be proprietary or developed specifically for virtual commissioning.

The framework, shown in Figure 2.2, has six stages, starting with the virtual preparation stage. In this stage, the virtual model is constructed by collecting or specifying virtual geometries, frames, and layouts. The virtual model is then used in subsequent stages, including system simulation, layout verification, reachability analysis, and post-commissioning activities like path planning, obstacle avoidance, position tracking, and operator safety.

During the constructive commissioning stage, simulators and drivers for the resources are implemented and tested iteratively using dummy control nodes that mimic specific resource control.

Simultaneously, in the emulated commissioning stage, an initial behavior model is prepared by iteratively implementing and testing variables, transitions, operations, and invariants with the controller, using emulated resources.

The next step, known as the software-in-the-loop stage, involves using the real controller to test and refine the model developed in the emulation stage, using the drivers and simulators from the constructive commissioning stage. This stage serves as a crucial transition from testing the initial model to implementing it for controlling the actual physical resources.

Before testing the model with the actual resources, dummy control nodes are once again employed in the reality-in-the-loop commissioning stage to emulate specific resource behavior and further iterate the drivers and the model. Finally, the model undergoes further improvement in the physical commissioning stage, where the controller is used to control the actual resources. Details of this framework are presented in Paper A.

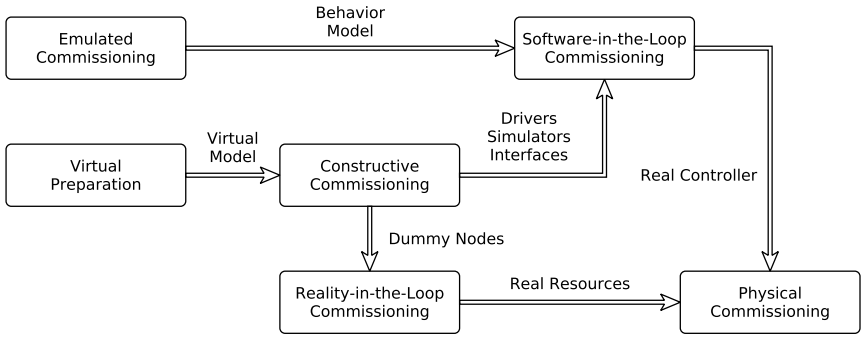

Figure 2.2: A high-level view of the preparation and VC framework from paper A.

2.2 Fault Localization Support

A model-based approach comes with its own set of challenges. While modeling a system, developers may introduce faulty behavior and constraints, or simply forget to specify certain behavior, resulting in unsolvable planning problems or anomalous planning results. Even with a test-driven development (TDD) approach [92], where developers iteratively test the model, it is often hard to identify the root cause of *why* a test failed.

Automated planning and synthesis can be achieved using various approaches. For example, a commonly used method is the planning-as-model-checking approach [93] where the planning problem is transformed into a verification problem. In this approach, the negation of the planning problem's goal is established as a safety property to be verified. If it is possible to violate the safety property, the MC will generate a counterexample that can be used as the plan to reach the goal. Otherwise, if the MC determines that no counterexample exists, it serves as evidence that no feasible plan can be found. However, *explaining* the unsolvability of a planning problem is a challenging task even with this approach, since there is no usable counterexample that can explain why no plan exists.

Related work for explaining that no plan exists includes detecting the unsolvability of a plan [94], [95], generating certificates or proofs of unsolvability [96], [97], and identifying adjustments to the planning problem that could render the problem solvable [98]. In [98] work, the focus is on investigat-

ing the initial state as the reason behind the inability to find a plan. This approach assumes the correctness of the models, thereby making the modification of the initial state a feasible option. In [99], the authors address this issue by capturing the user's expectations through considering abstractions of the given problem. More specifically, the authors use state abstractions to produce potential solutions and sub-goals at higher levels of abstraction.

However, none of the aforementioned research considers the inability to find plans due to modeling mistakes introduced by the model developers themselves. Such faults are inadvertently incorporated during model development. The purpose of iterative testing is to catch and correct such mistakes as early as possible, which can be done by unit testing the model, i.e. evaluating different initial and goal states with a planning algorithm. Such evaluations might reveal that a plan is *valid*, *anomalous*, or *unsolvable*. Passing a test allows the developer to continue refining the model. Anomalous results are not desirable, however they still offer developers valuable feedback. Sometimes, a test can quickly demonstrate that a goal state has been achieved in a manner that violates certain user-defined specifications. This allows a developer to identify potential flaws or shortcomings of the model.

The problem that we are addressing here is to enhance the feedback provided to the developer after failing to generate a plan. This situation typically arises when the developer has inaccurately modeled the system's behavior. Such mistakes are typically easier to detect and handle in the early design phases.

To do so, we took inspiration from software fault localization techniques [100], which have been extensively studied and applied in the field of software engineering. For example, a technique based on combinatorial testing separates input parameters into faulty-possible and healthy-possible to identify minimal failure-inducing combinations of parameters [101], [102]. Moreover, predicate switching [103] is a program fault localization technique that involves altering program states to force execution along different branches during a failed run. If switching a predicate results in a successful program execution, such predicate is identified as critical. Finally, model-checking approaches to fault localization also exist [104], [105], where a MC can provide a counterexample if a program fails to meet its specification.

More specifically, we draw inspiration from delta debugging techniques [106] [107] which identify the cause of software failures by comparing the program states between successful and failed tests. Suspicious variables are identified

by replacing their values from the successful test with their corresponding values from the same point in the failed test and then executing the program again. If the same failure occurs, the variable is considered suspicious, otherwise, it is no longer considered as a potential cause of the failure.

However, we do not directly apply such techniques to behavior models of intelligent automation systems. Instead, we define modeling abstractions called *operations* and *resources*, and test them for *suspiciousness* in a three-step approach. Firstly, we test the failed problems with relaxed versions of the model, where we remove complete resources in an effort to isolate potentially problematic ones. Depending on the outcome of the test, this might give an indication of which resources are incorrectly modeled. Next, we iteratively test the relaxed versions of the model by removing and adding back variables from suspicious resources, which provides us with a list of suspicious variables. However, this does not indicate *where* the problem in the model might be, so in the next step we identify operations that update the suspicious variables. Finally, we provide a list of achievable initial-goal state combinations to iteratively reduce the list of operations that were not triggered during random testing [108].

The main contribution of this effort is a fault localization method for aiding the development of intelligent automation systems, Figure 2.3. This method can be used to identify certain faults in the behavior model during testing and TDD. The effectiveness of this approach is exemplified by a use-case consisting of a robotic manipulator, a gantry, a structured light scanner, and a gripper. Details of this approach, can be found in Paper B.

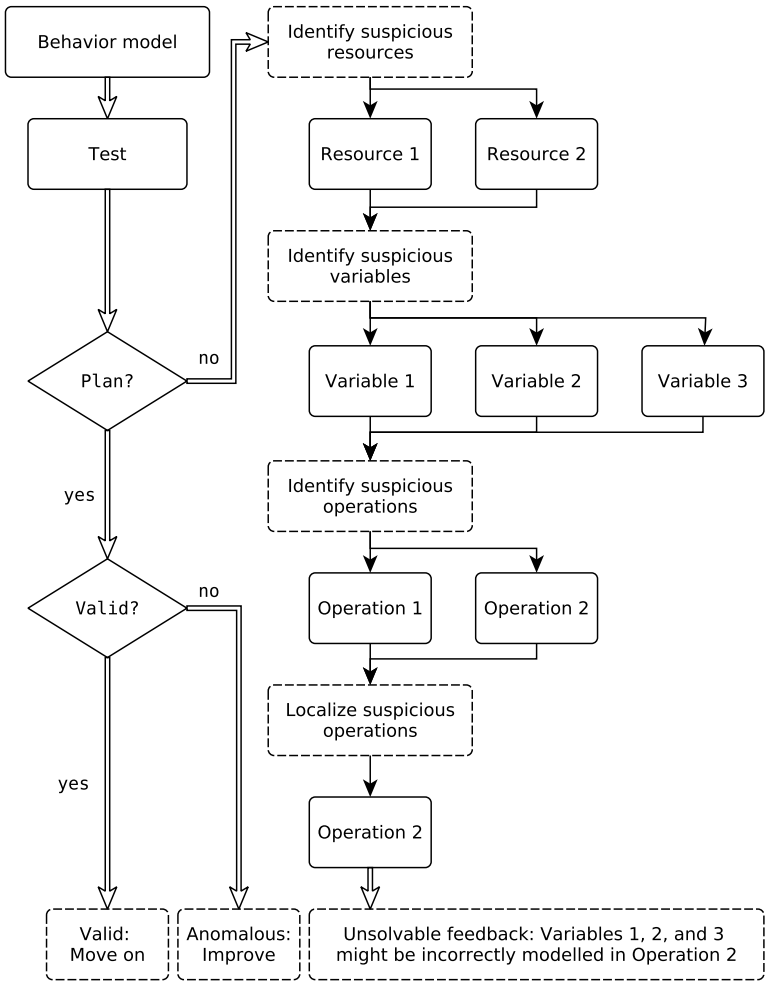

Figure 2.3: The fault localization procedure presented in Paper B.

CHAPTER 3

Planning and execution

The function of planning is to find a sequence of actions that will move the system from an initial state into a desired goal state, while satisfying various given constraints. In this context, execution refers to the ability to handle the execution of a plan, while continuously evaluating the current state and reacting to unexpected state changes.

3.1 Modeling

Modeling a planning and control system depends on design choices and specific use-cases. The representation of the planning domain, be it through state-space graphs or other logical formalisms, is a design choice that attempts to balance expressiveness and efficiency. This choice also depends on the specific application, whether it's for robotics automation, healthcare, transportation, manufacturing systems, or other domains. A common way to model planning problems is by using the Planning Domain Definition Language (PDDL), which can be seen as an effort to establish standardized languages for AI planning, building upon the concepts of the Stanford Research Institute Problem Solver language (STRIPS) and the Action Description Language (ADL).

In this work, we utilize a more automata-centric approach for modeling automation systems. The main reasons for such an approach include a more explicit behavior modeling approach compared to PDDL, as well as the possibility to extend transitions with non-formal guards and actions which are considered only during execution. The motivations and consequences of this decision choice are listed below:

1. A finite state space.
2. A dynamic environment, which assumes the changes occurring as a response to control actions, as well as unexpected external input.
3. No concurrency during planning, meaning that a plan is a strict sequence of actions, where a previous action ends before the next starts. In some designs though, it is practical to allow non-interfering actions to *pre-start* (during execution) before the previous have finished.
4. Deterministic models, which assume with certainty which state will be produced if an action is taken. This design choice allows us to avoid using non-deterministic models and their associated conceptual and computational complexities. However, this design choice considers that uncertainties and errors are handled by constant re-planning.

Instead of trying to describe how an action could fail with nondeterministic models, the models describing the behavior of IAS are deterministic, and do not express the different possible outcomes an action can have. The definitions below outline the components of modeling and execution of such systems:

d1: A *variable* v is a named unit of data that can be assigned a value x from a finite domain V.

d2: A *state* S is a set of tuples $S = \{\langle v_i, x_i \rangle\}$, where v_i is a variable with domain V_i and $x_i \in V_i$ is a value.

d3: A *predicate* is an *equality logic formula* F that evaluates to either *true* or *false*.

d4: An *equality logic formula* F is defined with the following grammar:

$$F : F \wedge F \mid F \vee F \mid \neg F \mid atom$$
$$atom : term == term \mid true \mid false$$
$$term : variable \mid value$$

d5: A *planning transition* t contains a guard predicate $g : S \to \{false, true\}$, and a set of action functions A, where $\forall a \in A, a : S \to S$ models the updates of the state variables. If the guard predicate evaluates to *true*, the transition can occur, after which the actions of the transition describe how the variables are updated. The notation we use to represent a planning transition is $t : g/A$.

d6: A *running transition* t_r extends the planning transition with an additional *running guard* g_r and additional *running action* A_r. We write running transitions as $t_r : g/g_r/A/A_r$, where g and g_r are both guard predicates and $g \wedge g_r : S \to \{false, true\}$, and A and A_r are both action functions, where $\forall a \in A \cup A_r, a : S \to S$ model the updates of the values of the state variables. While planning, only g and A are considered, i.e. the running transition is evaluated and taken as a planning transition. When the execution engine is running the plan, it is considering all components of t_r, i.e. the running transition guard becomes $g \wedge g_r$ and the set of transition actions becomes $A \cup A_r$.

d7: An *operation* O captures the behavior of tasks that can take some time to complete, and it is a convenient modeling abstraction for both planning and execution. A model of an operation can be in its *initial* (init) or *executing* (exec) state, see Figure 3.1. The *precondition* is a running transition associated with the *start* of the operation, switching it to the *executing* state. The operation will be in its executing state until the guard of the *postcondition* running transition is satisfied. The satisfaction of the postcondition implies that the operation is completed and can return to the *initial* state.

d8: An *automatic transitions* is a running transition that can be taken at any time assuming that the corresponding guard predicate is satisfied. Compared to operations that have to be queued in a plan in order to be taken, automatic transitions can be taken by the runner immediately when their guard evaluates to true. These can be used to trigger some specific behavior, for example triggering safety mechanisms, resets, timeout behavior, etc.

d9: A *behavior model* M is a collection of variables, operations, and automatic transitions that model the behavior of that system.

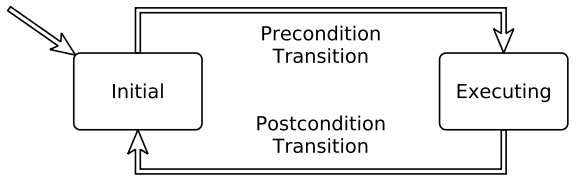

Figure 3.1: A model of an operation.

d10: A *planning problem* Ψ is a 4-tuple $\Psi = \langle S, g, M, p_{max} \rangle$ where S is the current state of the system, g is the goal predicate, M is the behavior model of the system, and p_{max} is a limit on the plan length.

d11: An *operation planner* is an algorithm which given a planning problem Ψ, returns a sequence of operations that takes the system from its current state to a state where the goal predicate is satisfied. While planning, the operation planner is avoiding the running guards g_r and the running actions A_r, treating operation preconditions and postconditions as planning transitions.

d12: A *plan* P is a sequence of operations.

d13: A *runner* is an algorithm which executes the *plan* P based on the model M, the current state of the system S, and a goal predicate g. While running, both the planning and running components of guards and actions of operation pre- and postconditions are evaluated and taken. Moreover, the runner is taking all automatic transitions that are enabled as soon as possible, irregardless of the plan.

3.2 Planning

There exists more than a few methods to calculate plans using deterministic models. Probably the oldest and most used approaches to planning are forwards state-space searches, implemented with uninformed algorithms such as Breadth-First Search (BFS), Depth-First Search (DFS), Iterative Deepening

(IDS), and Dijkstra's, or informed algorithms such as A*, Depth-First Branch and Bound (DFBB), and Greedy Best-First Search (GBFS).

Such methods can be referred to as *explicit*, since they operate on the state-transition level of representation, systematically exploring the search space by explicitly considering individual states. On the other hand, *symbolic* methods, based on Binary Decision Diagrams (BDDs), are based on a symbolic representation of the state by means of propositional variables.

Explicit state-space search is known for the performance in solving problems with a small number of state variables. On the other hand, symbolic methods can efficiently represent very-large state spaces but can sometimes be sensitive to number of variables and the variable ordering. An advantage of SAT-based methods is that they usually excel in solving hard combinatorial planning problems with a relatively high numbers of state variables [109]. Additionally, SAT planners are almost completely based on general purpose SAT solvers, meaning that every improvement in the solver directly improves planning.

Even though SAT-based planning was first proposed by Kautz and Selman already in 1992 [110], the interest of planning researchers in SAT-based planning methods was limited up until relatively recently. One of the main reasons behind this was the performance advantage of explicit state-space search over solving early SAT encodings of planning problems [111]. However, modern planners based on satisfiability now match, and often outperform, planners based on other search paradigms [112].

A commonly used symbolic method is the planning-as-model-checking approach [93] where the planning problem is transformed into a verification problem. In this approach, the negation of the planning problem's goal is established as a safety property to be verified. If it is possible to violate the safety property, the model checking will generate a counterexample that can be used as the plan to reach the goal. Otherwise, if the model checking determines that no counterexample exists, it serves as evidence that no feasible plan exists. Since we are concerned with finite-state cases, planning can be implemented with bounded model checking (BMC), where the model and specifications are defined as a SAT problem with a bounded size. Such a bound defines a limit on how many steps from the initial state to search for counterexamples. At the core of such SAT based planners are SAT solvers, which solve a SAT problem for each time step, until a satisfiable solution is found or the step limit is reached.

3.3 Solving

SAT-based planning relies on SAT solvers to calculate satisfiable assignments for a Boolean expression that encodes a plan. A Boolean expression is satisfiable if there exists a satisfying assignment, which means that variables from the expression have values such that the expression evaluates to true. A naive approach to SAT solving is to enumerate all assignments until a satisfiable assignment is found or no more possible assignments exist; however, this is not practically feasible.

In the context of this thesis, we are mainly interested in systematic search methods because they often yield the shortest path to the goal and can determine if the formula is unsatisfiable. Nevertheless, a comprehensive study on stochastic search algorithms for SAT can be found in [113]. Moreover, instead of look-ahead search methods [114], we are mainly interested in the conflict driven (CDCL) [115] evolution of DPLL [116]. The reason for this is that CDCL usually performs better on the family of problems we are interested in, namely planing of industrial problems.

On the other hand, some solvers such as [117] and [118] excel in solving hard random problem instances. Even if such solvers are usually not competitive on structured instances generated from real applications [119], some research indicates that there might be a way to utilize the strengths of both solver flavours, namely to use look-ahead search as a means to guide a CDCL solver [120]. Current modern solvers are based on CDCL as they can often solve hard industrial instances with a large number of variables.

The CDCL algorithm and low-level methods for SAT planning

The CDCL algorithm can be implemented in various ways, where some design choices can influence the planning speed and plan quality based on the type of the problem being solved. For example, a solver can be *specialized* to perform better on a certain type of problems, which is addressed in [121] with a focus on planning specific decision heuristics. Generally, the CDCL algorithm consists of the following procedures, as shown of Figure 3.2:

1. *Transformation to CNF:* SAT solvers accept the input formula in the Conjunctive Normal Form (CNF), so it is usually the case that the encoded problem has to be transformed into CNF. Every propositional formula can be transformed into an equisatisfiable CNF formula, and

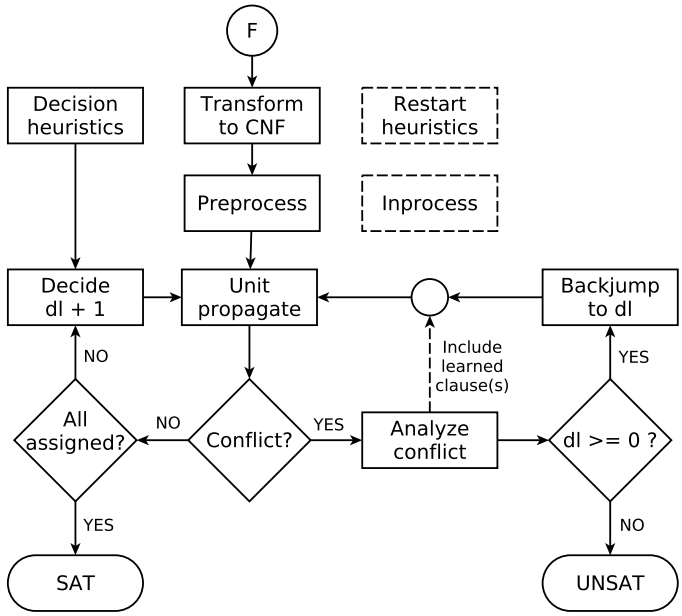

Figure 3.2: CDCL: An overview of the conflict-driven clause learning algorithm.

there are number of different methods to do this [38]. For example, a naive approach is to repeatedly apply De Morgan's laws and the distributive property, however, this can potentially lead to an exponential growth in the size of the formula.

A better way to do this is using Tseitin's encoding [122], which yields an equisatisfiable CNF formula with only a linear increase in size. The expense of Tseitin's transformation is the generation of n new Boolean variables, where n is the number of logical gates in the original formula. As an optimization, the number of clauses generated by Tseitin's encoding can be significantly reduced [123] if the input formula is in Negation Normal Form - NNF [124]. However, due to the added overhead from the need to initially transform the input formula to NNF, empirical data shows that modern solvers benefit very little from this reduction [125].

Further optimizations involve generating CNF formulas that are optimal

with respect to the number of clauses [126], [127], a linear time CNF generation algorithm [128], translation to CNF from Boolean circuits [129], [130], [131], and introducing new data structures for representing logic formulas [132], [133].

2. *Preprocessing and inprocessing:* After CNF transformation, a preprocessing step can be applied to potentially simplify the input formula [134]. Even if there is no theoretical connection between solving time and input formula size [128], it is often the case that simplified formulas take less time to solve, especially when the compared formulas come from the same set of problems [135]. The formula simplification step that is performed before the actual solving is referred to as *preprocessing*.

   On the other hand, *inprocessing* interleaves preprocessing and search, whilst allowing a limit to be set on the inprocessing time. This feature is important to reduce the inprocessing time relative to search time so that it is not very costly when compared. Interleaving preprocessing and search allows the use costly preprocessors without significantly increasing run-time, and makes the learned unit clauses during the search available to preprocessing.

   Preprocessing techniques can involve *subsumption* to eliminate logically redundant clauses from the CNF formula [136], *self-subsuming resolution* to resolve two clauses in order to find a resolvent subsuming both input clauses [135], *bounded variable elimination* which uses the Davis-Putnam procedure [137] to remove variables from the formula [135], [138], *blocked clause elimination* that removes all blocked clauses from the formula [139], *unhiding* that performs Depth First Search (DFS) on the binary implication graph to find all strongly connected components which can be removed without affecting the satisfiability of the formula [140], *hidden literal and tautology elimination* as a result of the previously performed DFS, *clause vivification* which strengthens (vivifies) the redundant clauses from the original formula [141], [142], and other techniques such as *lazy* and *hyper binary resolution* [143].

3. *Unit propagation:* The solver spends most of the time in a procedure called Unit Propagation (UP) which is also sometimes called Boolean Constraint Propagation. Unit propagation [144] repeatedly applies the unit clause rule for all unit clauses, either until all implications are ex-

hausted or a conflict occurs. If no conflict is encountered, the solver checks whether all variables are assigned. If so, the algorithm can terminate and return SAT.

To perform this procedure quicker, unit propagation can benefit from an efficient data structure. For example, there are different versions of *adjacency lists* where variables keep references to a number of clauses that contain them [145], *head-tail lists* which associate two references with each clause (the head and the tail) [146], a counter based method [115], and two-watched literals [147].

4. *Branching and decision heuristics*: If a conflict doesn't occur during unit propagation and not all variables are assigned, a decision heuristic is applied to choose a variable on which the search will branch and the decision level dl is incremented.

   For example, one approach is to simply randomly choose the next branching variable, however, such a heuristic usually only works well for small problems. Empirically, one of the most important features of state-of-the-art solvers is the Variable State Independent Decaying Sum (VSIDS) [147] heuristic, and its exponential (EVSIDS) [148] and normalized variants. However, there are other heuristics which might produce a similar performance, such as Variable Move To Front (VMTF) [149].

   Some of the other heuristics include the Dynamic Largest Individual Sum (DLIS), the Exponential Recency Weighted Average, the Jeroslow-Wong and 2-Sided Jeroslow-Wong heusristics, and Rintanen's planning specific heuristic [150].

5. *Conflict analysis*: If a conflict does occur during unit propagation, it is analyzed in order to produce a learned clause. This learned clause is appended to the original formula to aid the search by narrowing the search space. The conflict analysis algorithm decides the decision level where the CDCL algorithm has to backjump to. If a conflict was detected at the 0-th level, conflict analysis sets $dl = -1$ and the CDCL algorithm returns UNSAT in the next step. Otherwise, the algorithm backjumps to the level dl, or in other words, the assignments made after this level are unset, after which unit propagation is applied again.

6. *Search restart and heuristics*: To prevent being trapped in a single search space corner, modern SAT solvers initiate a periodic *restart* midway through the search process. This involves resetting certain search parameters and starting the search again from the top-level. Specific heuristics have been created to determine when and how frequently to execute this restart procedure during the search, for instance [151], [152], and [153].

High-level methods for SAT planning

We refer to the methods we have listed while exploring the CDCL algorithm as *low-level* methods, since they affect the performance of the solver itself. In Paper C however, we investigate and compare several methods for planning as satisfiability, without getting into the detailed tuning of these solvers. Instead, we focus on what we call *high-level* methods to improve the planning performance, which for example includes the structure of the model and how to call the solvers. The methods that we investigate are incremental planning [154], adding planning invariants [155], modeling with equality logic [39] and solving with SMT solvers [156], skipping planning steps [157], and subgoaling.

These methods and their final results are summarized here, however, for a more comprehensive study, refer to Paper C, where each subsection explores some methods and compares their performances on a set of classical planning benchmarks. The benchmarks that are chosen to test the methods are: *gripper* [158], *blocksworld* [159], *rovers* [160], *barman* [161] and *childsnack* [162]. These benchmarks are chosen to test different strengths and weaknesses of the following planning methods:

Incremental planning

The downside of the usual sequential approach to SAT planning is that for every iteration where an assignment is not found, a new context has to be created. Gocht and Balyo showed in 2017 [154] that it is possible to achieve a significant speed-up by using an incremental SAT-solver. Instead of throwing away the context with all the accumulated data from previous results, the same context is used and constraints are just added on top. An incremental planning algorithm utilizes an incremental solver which makes it possible to add a *backtracking point* in each step, so that the solver can choose which part of the context to save, and thus learn from previous attempts. In summary,

the advantages of an incremental base solver are that learnt clauses are kept, heuristic data is gathered, and that the overhead from asserting the same clauses is reduced.

Adding planning invariants

Invariants can be considered both as a modeling aid as well as a performance increasing method in automated planning. Invariants are specifications that must hold in every step of the calculated plan. In essence, invariants prune states from the state space. They can be added to the model to forbid some undesired behavior, or to enforce tighter constraints and thus derive a more compact state space representation. In each case, the state space gets reduced and thus planning is more efficient. For an existing problem, invariants can sometimes be synthesized to speed up planning [155]. Invariants can also often be a convenient tool for modelling different problems, but can in some cases be quite hard to use.

Equality logic and SMT solvers

Coupling a SAT solver with theory solvers, for example theories such as linear arithmetic, bit vectors, or arrays, SMT-based planning techniques [39] can encode and tackle real-world scenarios and complex application domains [40]. By using different *first-order* logic theories to increase expressiveness, some problems can be modeled in a much more convenient way compared to using pure propositional logic. In paper C, only equality logic is investigated since it is appropriate for modelling most classical planning problems. Since both propositional and equality logic are NP-complete [163], [164], it means that they can model the same decision problems with not more than a polynomial difference in variables [125]. Certain problems are more conveniently modeled in equality logic compared to propositional logic, and for some other problems the opposite is true. As for efficiency, the high level structure of the input equality logic formula can potentially be used to make the decision procedure work faster. This information may be lost if the problem is modelled directly in propositional logic [125].

Skipping steps

The planning algorithms increment the plan length by one when an assignment is not found, after which the encoding for that length is tested for satisfiability. This is a good method when the yielded plan should be of minimal length. However, calculating the shortest length plan can sometimes be very slow, as Rintanen showed in [165]. The evaluation cost of an unsatisfiable formula can be much higher than the evaluation cost of a satisfiable one, even if the latter is not of shortest length. Especially, when considering the sum of evaluation costs for all unsatisfiable formulas, the planner can spend a lot of time trying to find a satisfiable assignment. This is because the cost of evaluating the unsatisfiable formulas usually increases exponentially as the plan length increases [165]. When finding the first satisfiable solution which yields the plan with the minimal length is not a strict requirement, an improvement in the planning time can sometimes be achieved. Rather than increasing the plan length by *one* after an assignment is not found, this improvement can be realized by incrementing the plan length by a larger value. This allows us to skip some hard unsatisfiable instances which take a long time to evaluate.

Subgoaling

When a goal is defined as a conjunction of several predicates, such predicates can be looked at as *subgoals*. Instead of asking the planner to reach the monolithic goal in one planning task, multiple planning tasks can be instantiated to plan for each subgoal. Having a simpler goal shortens the plan length and thus the planning time. As mentioned before, the cost of evaluating unsatisfiable formulas increases exponentially, thus it is usually faster to search for a large number of shorter plans than the other way around. However, if the subgoal order is not correct, finding a plan can sometimes be slower or even impossible. This depends quite much on the nature of the problem itself as planning problems often exhibit symmetry properties that could be exploited to speed up their solving.

The benchmarks were ran on an Optiplex 9020 desktop PC with 8GB of RAM and an Intel Core i7-4790 CPU clocked at 3.60GHz. All algorithms used in this study were implemented using Z3's [156] quantifier free finite domain (QF_FD) theory, which supports propositional logic, bit-vector theories, pseudo-Boolean constraints, and enumeration data types. Fig. 3.3 shows

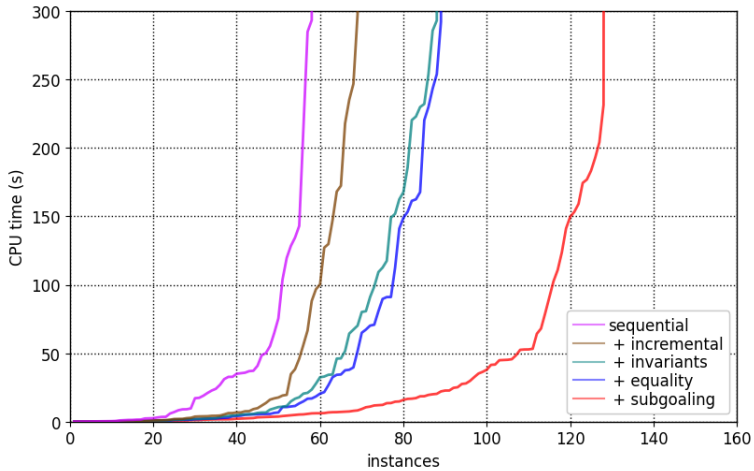

Figure 3.3: High-level planning methods contribution to performance.

how the studied methods contributed to increased planning performance.

3.4 Encoding

In order use solvers to calculate this plan for us, we have to encode the planning problem as a satisfiability problem. To do that, we use time steps to mark the variables, which essentially produces a new set of Boolean variables for each time step. A simple encoding for a plan of length i looks like this:

$$F_i = I(0) \wedge (\bigwedge_{k=0}^{i-1} \delta(T_{k,k+1})) \wedge G(i)$$

where $I(0)$ are clauses that encode the initial step, $G(i)$ are clauses that encode the goal step, and $\delta(T_{k,k+1})$ are clauses that encode the transitions and the rules that declare how those transitions can be taken in each step. A simple encoding $\delta(T_{k,k+1})$ encodes that exactly one transition is to be taken in one step. A single transition t for a step i is encoded like this:

$$t_i = t.name_i \wedge t.g_i \wedge t.a_{i+1}$$

where $t.name_i$ keeps track of the name of the transition that is taken in a step, $t.g_i$ encodes the guard of the transitions for the current step, and $t.a_{i+1}$ encodes the effect of the taken transition *actions* for the next step. Starting from $i = 0$, each unsatisfiable encoding results in incrementing the plan length and encoding a new problem of length $i = i + 1$. If at one point the solver returns SAT, the yielded result is parsed to provide a plan. Otherwise, if the solver can't find assignments for variables for each step i until the limit is reached, the planning problem is deemed *unsolvable*. The following listing shows the first three steps of incremental SAT planning and the manipulation of the assertion stack.

```
s0: add I0   -> Add the intital state assertions for step 0
    push bp0 -> Create a local scope (backtracking point)
    add G0   -> Add the goal state assertions for step 0
    check if SAT return else
s1: pop bp0  -> Revert the local scope (backtrack to bp0)
    add T01  -> Add the transition assertions for steps 0 and 1
    push bp1 -> Create a local scope (backtracking point)
    add G1   -> Add the goal state assertions for step 1
    check if SAT return else
s2: pop bp1  -> Revert the local scope (backtrack to bp1)
    add T12  -> Add the transition assertions for steps 1 and 2
    push bp2 -> Create a local scope (backtracking point)
    add G2   -> Add the goal state assertions for step 2
    ...
```

3.5 Execution

We control IAS with a number asynchronous tasks, which are independent non-blocking units of executions. Such tasks are similar to threads, but rather than being managed by the operating system, they are managed by an asynchronous runtime called Tokio [1]. The advantage of designing a control system this way is the possibility to decouple processes like planning, high-level execution, sensing, and low-level acting, which makes sure that such processes don't block each other. In this design, a crucial element which ensures that data

[1] https://tokio.rs/

can be exchanged between such tasks is the *shared state*. The shared state is a data structure, in this case a hashmap, which is wrapped in a mutual exclusion lock in order to protect data from simultaneous access.

Figure 3.4 shows an overview of how IAS are controlled. An *Execution* task (*runner*) is evaluating the state with a certain frequency, and executing the planned operations and the automatic transitions. This is done by updating the variable values in the shared state. The runner can also initiate re-planning by deleting existing plans and setting planning flags in the shared state. When new plans are calculated by the planning task, they are written into the shared state which are then again accessed by the runner.

The shared state is accessed by the resource handler tasks, which communicate using ROS2 with the resource interfaces. The handler tasks are also asynchronous, meaning that they can independently read from and write to the shared state which is kept safe by a mutex lock. Figure 3.4 illustrates that the ROS2 communication with the resource interfaces can be implemented with a publisher-subscriber, or a request-response mechanism.

On the low-level side, a resource interface is a task that is responsible for driving the actual resource, which can be actual hardware, a resource simulation, an algorithm, etc. In a distributed system, such resource interface tasks can be run on the machine in the field, making it part of the environment.

Additionally, a Digital Twin task maintains a virtual copy of the environment in order to keep track of the positions and orientations of digital geometries. This Digital Twin is updated with a certain frequency, which synchronizes it with the real environment. Such a Digital Twin provides input for task planning, path planning, online collision avoidance, operator safety, etc. More details about planning and execution can be found in Paper D, which presents a hierarchical planning and execution approach.

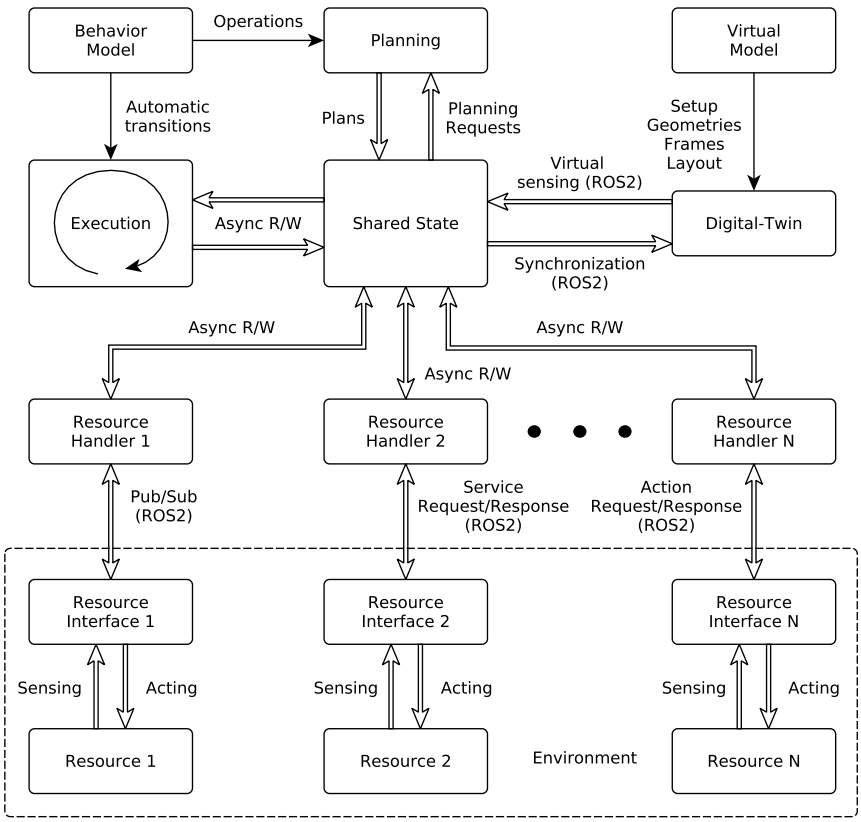

Figure 3.4: The high-level view of the execution system.

CHAPTER 4

Verification and coverability

Verification is a family of methods whose goal is to assure that a system satisfies some user specified requirements. The process of verification is a crucial element in the development of control software and plays a key role in ensuring the reliability, functionality, and performance of automation systems [72], [73]. Verification can be *formal*, which includes methods such as model checking [166] and SCT [167], or non-formal which includes methods such as testing [168].

Formal verification is performed on a model level and not on the system level, which means that a model of the system has to be built. The model of a system is created using a formal language, and doing so usually involves some level of abstraction, as it is seldom efficient (or even possible) to build a model that fully represents the behavior of the system which it models. Instead, a practical trade-off is to model and formally verify certain aspects of a system, and to apply testing to verify larger parts or complete systems [169].

Contrary to formal methods such as model checking, testing is seldom exhaustive, as it is usually quite costly to cover a reasonable amount of the input space. This means that it is not possible prove the absence of errors, however, we can still increase our confidence in the system. With appropriate testing

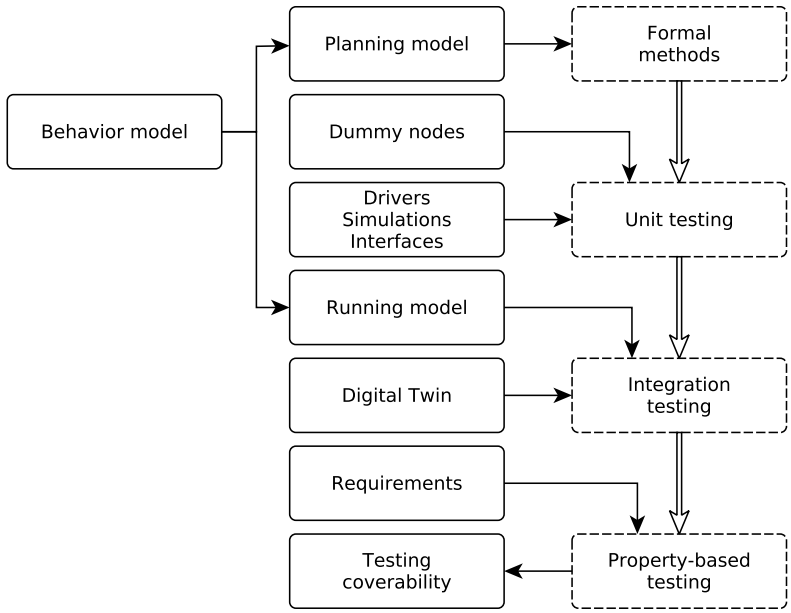

Figure 4.1: A high-level view of the verification activities of IAS.

methods accompanied by coverability criteria, testing can be scalable and usable for complex industrial systems [170]. Moreover, testing can be applied to a program in the conditions in which it is supposed to be run, for example on the target hardware or with the target operating system and drivers [171]. Figure 4.1 shows the testing-focused view of the V-model discussed in Chapter 1.

4.1 Verification of IAS

Verification of IAS is a process that ensures specific components or subsystems meet their design requirements. To verify IAS, both formal methods and testing can be applied. As it was discussed in Chapter 3, the behavior model is built from automatic transitions and operations consisting of a *formal* planning part, and a *non-formal* running part. This design choice allows us

to apply formal methods on the planning model, and test the running model, drivers, interfaces, simulations, etc. An example box scanning operation is shown below:

```
operation: scan_box
  deadline: 10 seconds
  pre: start_scan_box                  -> precondition
    g:  scan_req_state == initial &&   -> planning guard
        scan_req_trigger == false &&
        box_is_scanned == false
    gr: true                           -> running guard
    a:  [scan_req_trigger <- true]     -> planning actions
    ar: []                             -> running actions
  post: complete_scan_box              -> postcondition
    g:  true                           -> planning guard
    gr: scan_req_state == succeeded    -> running guard
    a:  [scan_req_state <- initial,    -> planning actions
        scan_req_trigger <- false,
        box_is_scanned <- true]
    ar: []                             -> running actions
```

Formal methods

Some of the most important families of properties that we would like to verify are *safety* properties, and *liveness* properties. Safety properties declare that something bad can never happen, for example, simultaneous access to a shared zone, a wrong assembly order, or reaching a forbidden state. On the other hand, liveness properties declare that something good will happen eventually, like always being able to reach a desired goal state [172] without getting stuck in livelock. Note that safety specifications have finite-length counterexamples, while liveness specifications have infinitely long counterexamples.

Instead of verifying that a model adheres to a specification, a synthesis approach can be applied to automatically calculate control rules that are correct by construction. For developing IAS, SCT [167] can be utilized to directly compute additional constraints for a system using a guard extraction technique [173], which ensures that a generated plan does not visit any undesired states. Using guard extraction, the calculated rules are represented as addi-

tional guards on the planning transitions. In other words, this technique is used to to refine the planning model based on high-level specifications.

Another formal method that can be applied on the planing model is model checking, which is a technique used to answer to the following question: Does the model of a system satisfy the given properties? In model checking [52], temporal properties are verified by exploring the state space using a set of initial states and transitions. Temporal properties are expressed using extensions of propositional logic, such as Linear Temporal Logic (LTL) [174]. LTL includes temporal operators like "next state" ($\bigcirc$), "always" ($\Box$), "eventually" ($\Diamond$), and "until" (U). For instance, the formula $\Box(x \rightarrow \bigcirc y)$ expresses that for all reachable states it is always true that if x holds in the current state, y will hold in the next state.

In order to take advantage of the SAT solving methods discussed in Section 3, the model and the specifications can be formulated as a SAT problem with a bounded size. On such a formulation, a BMC [175] method can be applied, where the *bound* defines in how many steps from the initial state to look for counterexamples. Similarly to the iterative encoding of the transitions discussed in Section 3, the negated LTL specifications are encoded up to the bound which is represented by the current iteration. If the problem is satisfiable, a specification has been violated, and the resulting assignment can be used to reconstruct a counterexample.

For example, a property that can be checked on the planning model is: *Always, after the scanner gets a command to scan and the box is not scanned, the box should eventually be scanned.* This is written in temporal logic as:

$$\Box\ ((\texttt{scan\_req\_trigger} \wedge \neg\ \texttt{box\_is\_scanned}) \rightarrow \Diamond\ \texttt{box\_is\_scanned})$$

A weakness of modeling systems by separating planning and running behavior is that it does not provide a general way to avoid deadlock situations during execution. While it is possible to perform guard extraction and model checking on the planning model, verifying the running model, as well as the communication, interfaces, drivers, etc., depends on testing activities.

Unit testing

To be able to verify that certain code will behave the way that it was intended by the programmer, a common practice is to test such code with manually written *unit* tests [176]. Such tests are meant to verify or falsify code for

chosen and specific known edge case input values, as well as to test previously known problematic inputs that caused bugs in the past.

In the Test Driven Development (TDD) methodology, unit tests are usually written before the actual code, which means that the tests keep failing until the developers implement the code correctly. Each *unit* is tested independently in an isolated environment to ensure a lack of dependencies in the code.

Unit testing focuses exclusively on the aspects that are essential to the unit under examination. This approach promotes a developer's ability to make alterations to the source code without affecting other units or the overall program functionality. For example, the things that we would like to unit test while developing IAS are the following:

1. *Simulations:* The simulation receives commands and simulates the resource behavior as expected.

2. *Functions:* The functions and algorithms driving the IAS are working as expected.

3. *Communication*: The resource handlers and their corresponding interfaces and hardware are exchanging information as expected, using the correct message types, handling all the message fields, processing commands and responding in time, etc.

4. *Drivers*: The resource drivers are able to drive the hardware as expected.

To unit test the behavior of the simulators, interfaces, and drivers during the development of IAS, we propose to create and utilize simple *dummy* nodes. These nodes allow us to test specific commands and verify the expected responses from the simulators, interfaces, and drivers. If the behavior aligns with the anticipated outcomes for the tested inputs, we can proceed. Otherwise, we iterate the components until we achieve the desired performance.

Integration testing

When every unit within a system has been tested to a satisfying amount, we can move on to assess larger sections of a system and the interaction of components through integration testing. Integration testing is done incrementally, where the components are gradually integrated and then tested as a group.

In IAS, integration testing is done during virtual commissioning, where a Digital Twin provides a common ground to test the communication, controllers, simulators and drivers, for all resources. Once such units have been verified independently, their interaction and collective functionality is tested together by integrating the controller and the behavior model, and testing specific scenarios.

Finally, the complete model is included in the test, which includes the running model and the automatic transitions. Such integration tests will potentially reveal if something is behaving incorrectly for a specific user-defined case.

Property-based testing

Contrary to unit testing which is checking the system for singular test cases, and integration testing which is checking the interaction of such units for singular scenario cases, property-based testing (PBT) [177] tries to verify or falsify systems by checking if certain properties of its output or behaviour hold for a large amount of automatically generated input values. When an input that violates the property is found, techniques such as *shrinking* can be used to automatically reduce it to a minimal counterexample. In practice, such a counterexample is very useful as it directly shows the reason why and how the property is violated, giving developers precise information on how to modify the program.

The nature of PBT is usually random [68], and without knowledge about the previously failed test examples, or *seeds*, such testing can still miss specific failing edge cases. Moreover, testing observes only a finite set of finite program executions, as it is usually the case that it is very costly, or even impossible to test code for all possible input values. Thus, PBT is best used to complement traditional unit testing.

To property test IAS, we can start by defining certain properties that must hold while executing such tests, for example:

1. If the goal is that the object is to be scanned, the object should eventually be scanned.

2. If scanning fails three times in a row, the scanning should be aborted, otherwise the object should be re-scanned.

3. If scanning fails five times in total, the scanning should be aborted, otherwise the object should be re-scanned.

4. If scanning times-out two times in a row, the goal should be aborted, otherwise the object should be re-scanned.

To test IAS for such requirements, a procedure that involves random test case generation, coverability analysis, and creating finishing test cases is utilized. The testing procedure consists of the following steps, but is shown in greater detail in Paper E:

1. Specifying rules on how to generate random test cases.

2. Specifying requirements that must be met while achieving these goals.

3. Generating a set of test cases.

4. Executing each test case.

5. Collecting data from runs to determine the coverability of the model.

6. If the coverability is deemed unsatisfactory, identifying parts that have not been sufficiently covered by the test cases.

7. Creating additional targeted test cases that address the missed areas, with the aim to increase coverability.

Over the years, a lot of tools for PBT have emerged as it has become a standard procedure in the industry. A pioneering testing tool that stood out is QuickCheck [68], which has later inspired other tools such as Hypothesis [69] and Proptest [70]. In this work, Proptest is used.

Test Coverability

Measuring structural coverability involves quantifying the adequacy of the testing process and providing insights into the completeness of the test suite. This can be achieved by defining a set of coverage measures that indicate the degree to which the system has been exercised during testing. For example, during the process of testing a system for a given set of requirements, it is possible to monitor and quantify the frequency and extent to which the behavior model is exercised. This evaluation can offer valuable insights into how to

optimize the initial test set and improve the overall coverability. By using this feedback to refine testing, we can ensure that the system is comprehensively tested to meet the required standards.

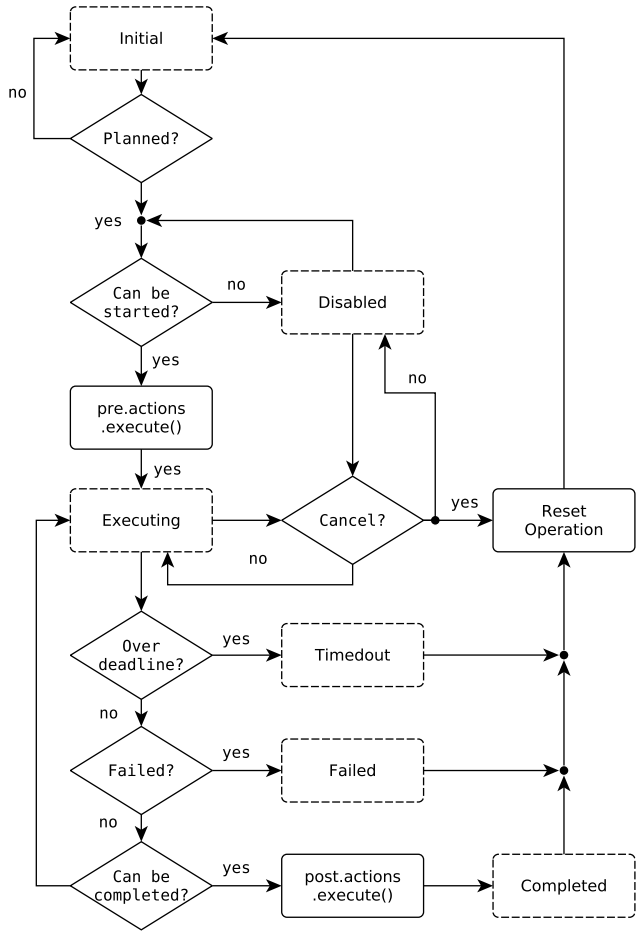

Figure 4.2: An operation and its states while being executed by the runner.

We take inspiration from the MC/DC [64] criterion, however, this criterion cannot be directly applied to evaluate the coverage of behavior models.

Therefore, we focus on the operation runner and compile an overview of the various running states of an operation:

- *Initial:* The operation is not the next one in the plan.
- *Disabled:* The operation is the next one in the plan for execution, but its precondition guard is not yet enabled.
- *Executing:* The precondition guard is enabled and the actions of the precondition are taken.
- *Timedout:* The operation was in the executing state for more time than its deadline allows.
- *Failed:* The operations has failed due to an error.
- *Completed:* The postcondition guard is enabled and the actions of the postcondition are taken. The operation is successfully completed.

Using this overview, we can start defining a criterion for structural coverability of IAS for a test set as follows:

- Every planned operation in the behavior model has visited its Disabled, Executing, Timedout, Failed, and Completed, state at least once.
- Every operation has been included in a plan at least once.
- Every automatic transition has been taken at least once.

Fulfilling these criteria for a set of requirements ensures that the behavior model has been exercised and can be used as a sanity check. Just as with MC/DC, meeting the requirements of this criterion does not guarantee that no defects remain. However, failing to meet this criterion indicates that certain portions of the model have not been sufficiently exercised, highlighting potential areas of concern. Similarly to MC/DC, this structural coverability is given as a percentage.

Improving the coverability is an iterative process. When the criteria are not met, it is necessary to go back to the model and identify the missed parts, then create specific test cases to cover them. Alternatively, the tester can be allowed to influence the simulation nodes to cover the missed aspects faster. After additional tests, the coverability can be reevaluated, and the process repeated until the desired level of coverage is achieved.

CHAPTER 5

Summary of included papers

The presented papers collectively contribute to advancing intelligent automation systems. Paper A highlights the limitations of traditional virtual commissioning methods and proposes the Integrated Virtual Preparation and Commissioning (IVPC) framework for flexible and collaborative systems. Complementing this, Paper B introduces a fault localization support system, supporting the model development process. Paper C investigates high-level planning methods, providing insights for satisfiability-based planners. Paper D introduces the Sequence Planner (SP) framework, addressing modeling and control challenges in collaborative robotics and autonomous machines, while Paper E emphasizes the crucial role of testing in ensuring reliability and functionality. A coverability criterion proposed in Paper E enhances test coverability for Intelligent Automation Systems. Together, these papers offer a comprehensive view of the development process, covering flexible system design, fault detection, planning methods, control framework implementation, and testing strategies for intelligent automation systems.

5.1 Paper A

Endre Erős, Martin Dahl, Kristofer Bengtsson, Petter Falkman and Knut Åkesson
Virtual preparation and commissioning of ROS2 based intelligent automation systems
Submitted for possible journal publication .

This paper discusses the limitations of applying traditional virtual commissioning methods for modern production systems which require flexibility and collaboration between humans and robots. Intelligent automation offers a solution by simplifying control logic synthesis and modeling tasks while delegating execution specifics to an online planning system. This approach uses models based on variables, states, transitions, and operations, benefiting from tools for planning, synthesis, and verification, to ensure system robustness and safety. This paper bridges the gap between research and application by introducing a bottom-up approach for creating ROS2-based intelligent automation systems. It proposes a framework called Integrated Virtual Preparation and Commissioning (IVPC) for Intelligent Automation Systems, extending traditional virtual commissioning methodologies. This framework incorporates additional guidelines and steps for developing flexible, collaborative, and distributed systems, focusing on simultaneous development of control logic and virtual plant. The paper's key contribution is this IVPC framework, which integrates techniques like testing, automated planning, formal methods, and resource simulations in a structured, iterative manner. This approach is exemplified through an industrial material handling case, demonstrating the practical application of the framework.

5.2 Paper B

Endre Erős, Kristofer Bengtsson and Knut Åkesson
Fault localization for intelligent automation systems
Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) vol. 29, pp. 1-8, 2023.

DOI: 10.1109/ETFA54631.2023.10275551 .

This paper focuses on a fault localization support system to assist the development of intelligent automation systems. The behavior of these systems is modeled using variables, states, transitions, and operations, facilitating the application of planning, synthesis, and verification tools to enhance safety and flexibility. A key challenge in developing such systems is the potential introduction of faulty behavior or constraints by developers, leading to planning problems that are either unsolvable or produce anomalous results. Traditional testing methods often struggle to pinpoint the exact reasons for such failures. This paper draws from software fault localization techniques to address this issue. The paper proposes a three-step approach to fault localization in the behavior models of intelligent automation systems. This method involves testing with relaxed versions of the model to identify suspicious resources and variables, and then pinpointing operations modelled by these suspicious variables. This contribution enhances the ability of developers to detect and correct faults during the model development phase. The efficacy of this approach is demonstrated through a use-case, highlighting its potential in improving the development process of intelligent automation systems.

5.3 Paper C

Endre Erős, Martin Dahl, Petter Falkman and Kristofer Bengtsson
Evaluation of high level methods for efficient planning as satisfiability
Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)
vol. 26, pp. 1-8, 2021.

DOI: 10.1109/ETFA45728.2021.9613254 .

The focus of this paper is to investigate and compare various high-level methods for planning as satisfiability. Instead of delving into the detailed tuning of solvers, the study emphasizes high-level approaches to improving planning performance. This includes considerations such as the structure of the model and how solvers are invoked. For example, the paper compares and investigates methods such as sequential and incremental solving, using invariants, modeling with equality and propositional logic, skipping planning steps, and subgoaling. The paper aims to provide insight for those aiming to implement systems that rely on satisfiability-based planners.

5.4 Paper D

Martin Dahl, **Endre Erős**, Kristofer Bengtsson, Martin Fabian and Petter Falkman
Sequence Planner: A Framework for Control of Intelligent Automation Systems
Published in the Applied Sciences Journal (MDPI) vol. 12, 12(11):5433, 2022.

DOI: 10.3390/app12115433 .

This paper introduces a framework called Sequence Planner (SP) designed to address challenges in developing automation systems with collaborative robotics and autonomous machines. These systems rely on online algorithms for sensing and acting to achieve high flexibility. SP offers a control framework for both traditional automation equipment and machines with autonomy. The framework leverages control logic synthesis and online planning algorithms, implemented with plug-in support for the Robot Operating System (ROS). The authors present findings from applying SP to an industrial demonstrator, demonstrating its suitability for highly flexible single-station systems. SP handles complex modeling tasks through control logic synthesis and online planning, introducing new abstractions like operations and intentions for hierarchical modeling. Experimental results from applying SP to control various hardware and software components, including real-time motion control algorithms, are presented, showcasing its planning performance and functionality. The paper builds upon previous publications and contributes new abstractions to facilitate the handling of larger systems.

5.5 Paper E

Endre Erős, Kristofer Bengtsson and Knut Åkesson
Structural coverability for intelligent automation systems
Published in conference proceedings of IEEE International Conference on Automation Science and Engineering (CASE) vol. 19, pp. 1-6, 2023.

DOI: 10.1109/CASE56687.2023.10260498 .

This paper addresses the role of testing in the development process of control software for automation systems, emphasizing its importance in ensuring system reliability, functionality, and performance. Given the impracticality of using model checking for verifying entire automated systems, the paper advocates for a comprehensive testing strategy to identify software defects before system deployment. The paper introduces a testing coverability criterion for Intelligent Automation Systems, inspired by the MC/DC criterion. This approach aims to enhance the test coverability of the System Under Test (SUT), which in this context is the behavior model used by the planner. The paper also references the significance of testing in ROS-based systems and the use of resource simulations during development to iteratively test and refine the behavior model. The coverability criterion is central to the proposed testing procedure allowing state injections into the controller and simulated resources. This method helps define additional test cases and identify modeling errors, illustrated through a use-case involving a robot manipulator, a structured light scanner, and a gripper.

CHAPTER 6

Concluding remarks and future work

Developing, controlling, and verifying, dynamic and unpredictable systems is a challenging task. One part of the challenge is that traditional automation methods might not be adequate for preparation and control of such systems. Instead, a model-based automation approach, where the behavior of the system is described with a model, might be more appropriate. In a model-based approach, behavior is modelled instead of explicitly programmed, capturing *what* a system can do rather than programming *how* to do it.

The second part of the challenge is that the tools, methods, frameworks, and examples, to develop model-based automation systems are missing or not mature yet. Finally, the necessary knowledge, lessons learned, and perspectives, within the industry are not yet established to a level where such methods can be seamlessly applied.

To address these challenges, we have discussed one way to develop, control, and verify IAS. Based on our research, we provide the following answers to the research questions introduced in Section 1:

RQ1 How can the development of IASs be supported by methods such as preparation, virtual commissioning, and fault localization, in order to facilitate the adoption of model-based automation?

Preparation and VC are mature industrial methods, which aim to increase production engineering efficiency while also decreasing on-site implementation time. It would appear that applying such methods could be beneficial to develop IASs.

However, in order to apply preparation and VC on IASs, such methods have to be adapted to be appropriate for systems which are model-based, flexible, and collaborative. This is addressed in paper A, where a framework for preparation and VC of IAS is presented. This framework has been successfully utilized in the development process of several IAS demonstrators.

In the development process of model-based systems, the engineering work is flipped from writing explicit and tractable control code, to writing behavior models with transitions, operations, and specifications. In this process, mistakes can be introduced in the model, resulting in unsolvable planning problems when testing during iterative development.

Addressing such unsolvable planning problems is discussed in paper B, where a fault localization inspired method is used to find potential mistakes in the model, providing some feedback to the developer. Compared to the result of an unsolvable planning problem where no feedback is given, this method can provide some help by pointing out suspicious variables and operations.

RQ2 How can the planning and execution processes in IASs be implemented, in order to ensure reactive and adaptive systems?

To ensure that IAS are reactive and adaptive while handling dynamic and unpredictable systems, the planning and execution in IAS must be able to sense the environment, calculate plans, and act, both quickly and reliably. To do so, planning can be implemented by using an incremental satisfiability approach, relying on the strengths of modern SAT solvers.

However, the performance of planning as satisfiability depends on low-level solver specific methods such as preprocessing and branching heuristics, and high-level modeling and planning specific methods such as

incremental planning and subgoaling. In paper C, several high-level methods have been evaluated on a set of standard planning benchmarks, which has provided an insight into how to model the problems and how to call the solvers, in order to increase the planning performance of IAS.

IAS need to be able to adapt to the environment and be able to find multiple ways to reach the goals. In such a goal-oriented automation approach, goals can be cancelled or changed, even during an execution of a process.

However, such an approach demands an execution and planning system which is fast enough, in order to allow a system that is reactive and adaptive. In paper D, a hierarchical modeling and planning system is introduced, which aims to make it easier to handle larger systems by separating the modeling and planning planning processes into two levels. This system has been successfully utilized for implementation and control of several IAS demonstrators.

RQ3 How can IASs be verified with formal methods and testing, and how can the adequacy of such testing be assessed?

Systems that are addressed by IAS can be unpredictable, which means that actions can have different outcomes. One way to handle this is to model all the different outcomes of actions, however, as discussed in Section 3, this is not always possible. Instead, a design choice for developing IASs was to model behavior with deterministic models, and separate the behavior into a *formal* planning part, and a *non-formal* running part.

However, this design choice makes it impossible to formally verify the complete behavior of the system. Instead, a practical trade-off was to model and formally verify certain aspects of a system, and to apply testing to verify larger parts or complete systems.

Connected to the testing process of IAS, paper E presents a coverability criteria to assess the extent to which the model has been exercised while testing. This criteria can be used as an indicator to decide if more tests are needed in order increase the confidence in IASs.

Future work

An important future endeavour would be to apply the methods presented in this paper to more use-cases. While such methods were established and implemented on several demonstrators, their usefulness on a wider range on use-cases is yet to be evaluated.

Another aspect connected to use-cases which needs investigation is the scalability of IAS. While the proposed methods presented in this thesis showed to be useful on several small to medium scale demonstrators, such demonstrators were relatively small compared to actual industrial production systems.

Modeling behavior for IAS is a challenging task, and while this thesis has presented methods to assist this task, other methods could prove valuable for model development. For example, instead of only suggesting suspicious resources, variables, and operations, a model synthesis approach could be investigated to automatically synthesize parts of the model based on the existing model and some user defined requirements.

Furthermore, an effort that could prove useful would be to try to learn certain parts of the model. For example, after developing and verifying the formal *planning* subset of the model, parts of the non-formal running model could be learnt by interacting with the simulated environment while satisfying some requirements. It might not be possible to learn the complete model, however, and investigation could reveal that it might be possible to correctly learn some parts.

Another improvement that is due is the modeling API and quality of life. To truly facilitate the adoption of IAS by other users and engineers, effort has to be spent for improving the accessibility of the modeling, programming, and graphical interfaces of IAS. Ideally, such interfaces could abstract certain details for high-level general usability, while allowing complete access to more experienced users. Moreover, proper tutorials, documentation, and examples, have to be developed to aid learning engineers.

References

[1] G. I. Fragapane, D. A. Ivanov, M. Peron, F. Sgarbossa, and J. O. Strandhagen, "Increasing flexibility and productivity in industry 4.0 production networks with autonomous mobile robots and smart intralogistics," *Annals of Operations Research*, pp. 1–19, 2020.

[2] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, no. 3, pp. 213–222, 2014, ISSN: 2288-4300.

[3] P. J. Van Laarhoven and W. H. Zijm, "Production preparation and numerical control in pcb assembly," *International Journal of Flexible Manufacturing Systems*, vol. 5, no. 3, pp. 187–207, 1993.

[4] T. Lechler, E. Fischer, M. Metzner, A. Mayr, and J. Franke, "Virtual commissioning–scientific review and exploratory use cases in advanced production systems," *Procedia CIRP*, vol. 81, pp. 1125–1130, 2019.

[5] G. Barbieri, A. Bertuzzi, A. Capriotti, *et al.*, "A virtual commissioning based methodology to integrate digital twins into manufacturing systems," *Production Engineering*, vol. 15, pp. 397–412, 2021.

[6] C. Scheifele, A. Verl, and O. Riedel, "Real-time co-simulation for the virtual commissioning of production systems," *Procedia CIRP*, vol. 79, pp. 397–402, 2019.

[7] A. Albo and P. Falkman, "A standardization approach to virtual commissioning strategies in complex production environments," *Procedia Manufacturing*, vol. 51, pp. 1251–1258, 2020.

[8] J. Bao, D. Guo, J. Li, and J. Zhang, "The modelling and operations for the digital twin in the context of manufacturing," *Enterprise Information Systems*, vol. 13, no. 4, pp. 534–556, 2019.

[9] M.-H. Hung, Y.-C. Lin, H.-C. Hsiao, *et al.*, "A novel implementation framework of digital twins for intelligent manufacturing based on container technology and cloud manufacturing services," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1614–1630, 2022.

[10] Y. Qamsane, C.-Y. Chen, E. C. Balta, *et al.*, "A unified digital twin framework for real-time monitoring and evaluation of smart manufacturing systems," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 2019, pp. 1394–1401.

[11] W. Shen, T. Hu, Y. Yin, J. He, F. Tao, and A. Nee, "Digital twin based virtual commissioning for computerized numerical control machine tools," in *Digital Twin Driven Smart Design*, F. Tao, A. Liu, T. Hu, and A. Nee, Eds., Academic Press, 2020, pp. 289–307, ISBN: 978-0-12-818918-4.

[12] J. Wang, X. Niu, R. X. Gao, Z. Huang, and R. Xue, "Digital twin-driven virtual commissioning of machine tool," *Robotics and Computer-Integrated Manufacturing*, vol. 81, p. 102 499, 2023, ISSN: 0736-5845.

[13] M. A. Wehrmeister, "Generating ros-based software for industrial cyber-physical systems from uml/marte," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 313–320.

[14] B. Wally, J. Vyskočil, P. Novák, *et al.*, "Flexible production systems: Automated generation of operations plans based on isa-95 and pddl," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4062–4069, 2019.

[15] B. Illmer, M. Karkowski, and M. Vielhaber, "Petri net controlled virtual commissioning – a virtual design-loop approach," *Procedia CIRP*, vol. 91, pp. 152–157, 2020, Enhancing design through the 4th Industrial Revolution Thinking, ISSN: 2212-8271.

[16] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[17] A. Philippot, B. Riera, V. Kunreddy, and S. Debernard, "Advanced tools for the control engineer in industry 4.0," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, 2018, pp. 555–560.

[18] T. Blochwitz, M. Otter, M. Arnold, *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th international Modelica conference*, Linköping University Press, 2011, pp. 105–114.

[19] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*, 1st. USA: Cambridge University Press, 2016, ISBN: 1107037271.

[20] *ROS 2*, `https://index.ros.org/doc/ros2/`, [Online; accessed 25-Feb-2019], 2019.

[21] M. Cashmore, M. Fox, D. Long, *et al.*, "Rosplan: Planning in the robot operating system," in *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'15, Jerusalem, Israel: AAAI Press, 2015, pp. 333–341, ISBN: 978-1-57735-731-5.

[22] F. Rovida, M. Crosby, D. Holz, *et al.*, "Skiros—a skill-based robot control platform on top of ros," in *Robot Operating System (ROS): The Complete Reference (Volume 2)*, A. Koubaa, Ed. Cham: Springer International Publishing, 2017, pp. 121–160, ISBN: 978-3-319-54927-9.

[23] A. Munawar, G. De Magistris, T. Pham, *et al.*, "Maestrob: A robotics framework for integrated orchestration of low-level control and high-level reasoning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 527–534.

[24] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 564–571.

[25] E. Aertbeliën and J. De Schutter, "Etasl/etc: A constraint-based task specification language and robot controller using expression graphs," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 1540–1546.

[26] C. Schou, R. S. Andersen, D. Chrysostomou, S. Bøgh, and O. Madsen, "Skill-based instruction of collaborative robots in industrial settings," *Robotics and Computer-Integrated Manufacturing*, vol. 53, pp. 72–80, 2018.

[27] V. Krueger, F. Rovida, B. Grossmann, *et al.*, "Testing the vertical and cyber-physical integration of cognitive robots in manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 57, pp. 213–229, 2019.

[28] D. Dal Moro, M. Robol, M. Roveri, and P. Giorgini, "A demonstration of bdi-based robotic systems with ros2," in *International Conference on Practical Applications of Agents and Multi-Agent Systems*, Springer, 2022, pp. 473–479.

[29] R. Wang, Y. Guan, H. Song, *et al.*, "A formal model-based design method for robotic systems," *IEEE Systems Journal*, vol. 13, no. 1, pp. 1096–1107, 2019.

[30] S. Profanter, A. Perzylo, M. Rickert, and A. Knoll, "A generic plug & produce system composed of semantic opc ua skills," *IEEE Open Journal of the Industrial Electronics Society*, vol. 2, pp. 128–141, 2021.

[31] T. Coito, M. S. Martins, J. L. Viegas, *et al.*, "A middleware platform for intelligent automation: An industrial prototype implementation," *Computers in industry*, vol. 123, p. 103 329, 2020.

[32] C. Nam, S. Lee, J. Lee, *et al.*, "A software architecture for service robots manipulating objects in human environments," *IEEE Access*, vol. 8, pp. 117 900–117 920, 2020.

[33] Y. Chen, U. Rosolia, and A. D. Ames, "Decentralized task and path planning for multi-robot systems," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4337–4344, 2021.

[34] K. Dorofeev and M. Wenger, "Evaluating skill-based control architecture for flexible automation systems," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2019, pp. 1077–1084.

[35] M. Dahl, C. Larsen, E. Eros, K. Bengtsson, M. Fabian, and P. Falkman, "Interactive formal specification for efficient preparation of intelligent automation systems," *CIRP Journal of Manufacturing Science and Technology*, vol. 38, pp. 129–138, 2022, ISSN: 1755-5817.

[36] F. Martin, M. Morelli, H. Espinoza, F. J. Lera, and V. Matellan, "Optimized execution of pddl plans using behavior trees," *arXiv preprint arXiv:2101.01964*, 2021.

[37] F. Martin, J. G. Clavero, V. Matellan, and F. J. Rodriguez, "Plansys2: A planning system framework for ros2," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 9742–9749.

[38] S. Alouneh, S. Abed, M. H. A. Shayeji, and R. Mesleh, "A comprehensive study and analysis on sat-solvers: Advances, usages and achievements," *Artificial Intelligence Review*, pp. 1–27, 2018.

[39] J. E. Arxer, "Smt techniques for planning problems," 2018.

[40] A. Bit-Monnot, F. Leofante, L. Pulina, and A. Tacchella, "Smt-based planning for robots in smart factories," in *Advances and Trends in Artificial Intelligence. From Theory to Practice*, F. Wotawa, G. Friedrich, I. Pill, R. Koitz-Hristov, and M. Ali, Eds., Cham: Springer International Publishing, 2019, pp. 674–686, ISBN: 978-3-030-22999-3.

[41] R. Cavada, A. Cimatti, M. Dorigatti, *et al.*, "The nuxmv symbolic model checker," in *Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings 26*, Springer, 2014, pp. 334–342.

[42] L. De Moura and 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.

[43] K. Akesson, M. Fabian, H. Flordal, and R. Malik, "Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems," in *2006 8th International Workshop on Discrete Event Systems*, IEEE, 2006, pp. 384–385.

[44] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," *Formal methods for the design of real-time systems*, pp. 200–236, 2004.

[45] A. Fleury and M. Heisinger, "Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020," *SAT COMPETITION*, vol. 2020, p. 50, 2020.

[46] R. Anand, D. Aggarwal, and V. Kumar, "A comparative analysis of optimization solvers," *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 623–635, 2017.

[47] J. Rintanen, "Madagascar: Scalable planning with sat," *Proceedings of the 8th International Planning Competition (IPC-2014)*, vol. 21, pp. 1–5, 2014.

[48] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, no. 53, pp. 1–2, 2005.

[49] E. Solowjow, I. Ugalde, Y. Shahapurkar, *et al.*, "Industrial robot grasping with deep learning using a programmable logic controller (plc)," *arXiv preprint arXiv:2004.10251*, 2020.

[50] D. Morrison, P. Corke, and J. Leitner, "Learning robust, real-time, reactive robotic grasping," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 183–201, 2020.

[51] S. James, P. Wohlhart, M. Kalakrishnan, *et al.*, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.

[52] O. Grumberg, E. Clarke, and D. Peled, *Model checking*, 1999.

[53] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 45–64, 2016.

[54] A. Afzal, C. L. Goues, M. Hilton, and C. S. Timperley, "A study on challenges of testing robotic systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 2020, pp. 96–107.

[55] R. Ramler, W. Putschögl, and D. Winkler, "Automated testing of industrial automation software: Practical receipts and lessons learned," in *Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation*, 2014, pp. 7–16.

[56] R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler, and A. Zoitl, "Test case generation approach for industrial automation systems," in *The 5th international conference on automation, robotics and applications*, IEEE, 2011, pp. 57–62.

[57] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

[58] P. Helle, W. Schamai, and C. Strobel, "Testing of autonomous systems–challenges and current state-of-the-art," in *INCOSE international symposium*, Wiley Online Library, vol. 26, 2016, pp. 571–584.

[59] A. Santos, A. Cunha, and N. Macedo, "The high-assurance ROS framework," in *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*, 2021, pp. 37–40.

[60] A. Santos, A. Cunha, and N. Macedo, "Property-based testing for the robot operating system," in *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, ser. A-TEST 2018, Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, pp. 56–62, ISBN: 9781450360531.

[61] R. Carvalho, A. Cunha, N. Macedo, and A. Santos, "Verification of system-wide safety properties of ros applications," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 7249–7254.

[62] L. Dust, R. Gu, C. Seceleanu, M. Ekström, and S. Mubeen, "Pattern-based verification of ros 2 nodes using uppaal," in *International Conference on Formal Methods for Industrial Critical Systems*, Springer, 2023, pp. 57–75.

[63] X. Cai and M. R. Lyu, "The effect of code coverage on fault detection under different testing profiles," in *Proceedings of the 1st International Workshop on Advances in Model-Based Testing*, ser. A-MOST '05, St. Louis, Missouri: Association for Computing Machinery, 2005, pp. 1–7, ISBN: 1595931155.

[64] K. Hayhurst and D. Veerhusen, "A practical approach to modified condition/decision coverage," in *20th DASC. 20th Digital Avionics Systems Conference (Cat. No.01CH37219)*, vol. 1, 2001, 1B2/1–1B2/10 vol.1.

[65] M. Born, J. Favaro, and O. Kath, "Application of iso dis 26262 in practice," in *Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness and Safety*, ser. CARS '10, Valencia, Spain: Association for Computing Machinery, 2010, pp. 3–6, ISBN: 9781605589152.

[66] K. Maruchi, H. Shin, and M. Sakai, "MC/DC-Like structural coverage criteria for function block diagrams," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, 2014, pp. 253–259.

[67] H. Hemmati, "How effective are code coverage criteria?" In *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 151–156.

[68] K. Claessen and J. Hughes, "Quickcheck: A lightweight tool for random testing of haskell programs," *SIGPLAN Not.*, vol. 35, no. 9, pp. 268–279, Sep. 2000, ISSN: 0362-1340.

[69] D. R. MacIver, *Hypothesis 4.24*, `https://github.com/HypothesisWorks/hypothesis`, 2018.

[70] *Proptest*, `https://docs.rs/proptest/latest/proptest/`, 2022.

[71] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, Springer, 2010, pp. 167–170.

[72] R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler, and A. Zoitl, "Test case generation approach for industrial automation systems," in *The 5th International Conference on Automation, Robotics and Applications*, 2011, pp. 57–62.

[73] D. Winkler, R. Hametner, T. Östreicher, and S. Biffl, "A framework for automated testing of automation systems," in *2010 IEEE 15th Conference on Emerging Technologies and Factory Automation (ETFA 2010)*, 2010, pp. 1–4.

[74] S. Sebastian, S. Magnus, M. Thron, *et al.*, "Test methodology for virtual commissioning based on behaviour simulation of production systems," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–9.

[75] V. Hubka and W. E. Eder, *Theory of technical systems: a total concept theory for engineering design*. Springer Science & Business Media, 2012.

[76] M. M. Andreasen, T. J. Howard, and H. P. L. Bruun, "Domain theory, its models and concepts," *An anthology of theories and models of design: philosophy, approaches and empirical explorations*, pp. 173–195, 2014.

[77] M. M. Andreasen, "Design methodology," *Journal of Engineering Design*, vol. 2, no. 4, pp. 321–335, 1991.

[78] G. S. Altshuller, *The innovation algorithm: TRIZ, systematic innovation and technical creativity*. Technical innovation center, Inc., 1999.

[79] N. P. Suh and N. P. Suh, *Axiomatic design: advances and applications*. Oxford university press New York, 2001, vol. 4.

[80] A. Hatchuel and B. Weil, "A new approach of innovative design: An introduction to ck theory.," in *DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm*, 2003.

[81] J. S. Gero and U. Kannengiesser, "A function–behavior–structure ontology of processes," *Ai Edam*, vol. 21, no. 4, pp. 379–391, 2007.

[82] D. Braha and O. Maimon, *A mathematical theory of design: foundations, algorithms and applications*. Springer Science & Business Media, 2013, vol. 17.

[83] L. T. Blessing and A. Chakrabarti, *DRM: A design reseach methodology*. Springer, 2009.

[84] G. Muller, "Systems engineering research methods," *Procedia Computer Science*, vol. 16, pp. 1092–1101, 2013.

[85] F. Auinger, M. Vorderwinkler, and G. Buchtela, “Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop",” in *WSC’99. 1999 Winter Simulation Conference Proceedings. ’Simulation - A Bridge to the Future’ (Cat. No.99CH37038)*, vol. 1, 1999, 798–805 vol.1.

[86] N. Shahim and C. Møller, “Economic justification of virtual commissioning in automation industry,” in *2016 Winter Simulation Conference (WSC)*, 2016, pp. 2430–2441.

[87] T. Foote, “Tf: The transform library,” in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013, pp. 1–6.

[88] S. Chitta, I. Sucan, and S. Cousins, “Moveit![ros topics],” *IEEE Robotics and Automation Magazine - IEEE ROBOT AUTOMAT*, vol. 19, pp. 18–19, Mar. 2012.

[89] *Rust: A language empowering everyone to build reliable and efficient software.* `https://www.rust-lang.org/`, Accessed: 2023-03-17.

[90] *R2r - easy to use, runtime-agnostic, async rust bindings for ROS2.* `https://github.com/sequenceplanner/r2r`, Accessed: 2023-03-17.

[91] M. Dahl, E. Erős, K. Bengtsson, M. Fabian, and P. Falkman, “Sequence planner: A framework for control of intelligent automation systems,” *Applied Sciences*, vol. 12, no. 11, 2022, ISSN: 2076-3417.

[92] D. Astels, *Test Driven Development: A Practical Guide*. Prentice Hall Professional Technical Reference, 2003, ISBN: 0131016490.

[93] F. Giunchiglia and P. Traverso, “Planning as model checking,” in *Recent Advances in AI Planning: 5th European Conference on Planning, ECP’99, Durham, UK, September 8-10, 1999. Proceedings 5*, Springer, 2000, pp. 1–20.

[94] C. Bäckström, P. Jonsson, and S. Ståhlberg, “Fast detection of unsolvable planning instances using local consistency,” in *Proceedings of the International Symposium on Combinatorial Search*, vol. 4, 2013, pp. 29–37.

[95] J. Hoffmann, P. Kissmann, and A. Torralba, “" distance"? who cares? tailoring merge-and-shrink heuristics to detect unsolvability.,” in *ECAI*, 2014, pp. 441–446.

[96] S. Eriksson, G. Röger, and M. Helmert, "Unsolvability certificates for classical planning," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, no. 1, pp. 88–97, Jun. 2017.

[97] S. Eriksson, G. Röger, and M. Helmert, "A proof system for unsolvable planning tasks," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, no. 1, pp. 65–73, Jun. 2018.

[98] "Coming up with good excuses: What to do when no plan can be found," vol. 20,

[99] S. Sreedharan, S. Srivastava, D. Smith, and S. Kambhampati, "Why can't you do that hal? explaining unsolvability of planning tasks," in *International Joint Conference on Artificial Intelligence*, 2019.

[100] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[101] C. Nie and H. Leung, "The minimal failure-causing schema of combinatorial testing," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, Sep. 2011, ISSN: 1049-331X.

[102] X. Niu, C. Nie, Y. Lei, and A. T. Chan, "Identifying failure-inducing combinations using tuple relationship," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, 2013, pp. 271–280.

[103] X. Zhang, N. Gupta, and R. Gupta, "Locating faults through automated predicate switching," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06, Shanghai, China: Association for Computing Machinery, 2006, pp. 272–281.

[104] A. Griesmayer, S. Staber, and R. Bloem, "Fault localization using a model checker," *Softw. Test. Verif. Reliab.*, vol. 20, no. 2, pp. 149–173, Jun. 2010, ISSN: 0960-0833.

[105] A. Griesmayer, S. Staber, and R. Bloem, "Automated fault localization for c programs," *Electronic Notes in Theoretical Computer Science*, vol. 174, no. 4, pp. 95–111, 2007, Proceedings of the Workshop on Verification and Debugging (V&D 2006), ISSN: 1571-0661.

[106] A. Zeller, "Isolating cause-effect chains from computer programs," in *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, ser. SIGSOFT '02/FSE-10, Charleston, South Carolina, USA: Association for Computing Machinery, 2002, pp. 1–10, ISBN: 1581135149.

[107] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 183–200, 2002.

[108] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[109] J. Rintanen, "Search methods for classical and temporal planning," *Tutorials of the 21th European Conference on Artificial Intelligence (ECAI 2014)*, vol. 21, 2014.

[110] H. Kautz and B. Selman, "Planning as satisfiability," in *Proceedings of the 10th European Conference on Artificial Intelligence*, ser. ECAI '92, Vienna, Austria: John Wiley & Sons, Inc., 1992, pp. 359–363, ISBN: 0471936081.

[111] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence*, vol. 193, pp. 45–86, 2012, ISSN: 0004-3702.

[112] J. Rintanen, "Madagascar: Scalable planning with sat," *Proceedings of the 8th International Planning Competition (IPC-2014)*, vol. 21, 2014.

[113] H. Hoos and T. Stützle, "Local search algorithms for sat: An empirical evaluation," *Journal of Automated Reasoning*, vol. 24, pp. 421–481, Jan. 2000.

[114] M. Heule and H. van Maaren, "Look-ahead based sat solvers.," *Handbook of satisfiability*, vol. 185, pp. 155–184, 2009.

[115] J. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[116] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962, ISSN: 0001-0782.

[117] A. Anbulagan and C.-M. Li, "Heuristics based on unit propagation for satisfiability problems," Jul. 2000.

[118] O. Dubois and G. Dequen, "A backbone-search heuristic for efficient solving of hard 3-sat formulae," Jan. 2001, pp. 248–253.

[119] L. Zhang and S. Malik, "The quest for efficient boolean satisfiability solvers," in *Proceedings of the 14th International Conference on Computer Aided Verification*, ser. CAV '02, Berlin, Heidelberg: Springer-Verlag, 2002, pp. 17–36, ISBN: 3540439978.

[120] M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and conquer: Guiding cdcl sat solvers by lookaheads," in *Hardware and Software: Verification and Testing*, K. Eder, J. Lourenço, and O. Shehory, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 50–65, ISBN: 978-3-642-34188-5.

[121] J. Rintanen, "Planning with specialized sat solvers," in *AAAI*, 2011.

[122] A. A. Semenov, "About tseitin's transformation in logical equations," *Prikladnaya Diskretnaya Matematika*, no. 10, pp. 12–13, 2009.

[123] D. A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *J. Symb. Comput.*, vol. 2, no. 3, pp. 293–304, Sep. 1986, ISSN: 0747-7171.

[124] A. Nonnengart and C. Weidenbach, "Computing small clause normal forms," in *Handbook of Automated Reasoning*, ser. Handbook of Automated Reasoning, A. Robinson and A. Voronkov, Eds., Amsterdam: North-Holland, 2001, pp. 335–367, ISBN: 978-0-444-50813-3.

[125] D. Kroening and O. Strichman, *Decision Procedures: An Algorithmic Point of View*, 2nd ed. Springer Publishing Company, Incorporated, 2016, ISBN: 978-3-662-50497-0.

[126] T. Boy de la Tour, "An optimality result for clause form translation," *J. Symb. Comput.*, vol. 14, no. 4, pp. 283–301, Oct. 1992, ISSN: 0747-7171.

[127] A. Nonnengart, G. Rock, and C. Weidenbach, "On generating small clause normal forms," in *Automated Deduction — CADE-15*, C. Kirchner and H. Kirchner, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 397–411, ISBN: 978-3-540-69110-5.

[128] D. Sheridan, "The optimality of a fast cnf conversion and its use with sat," in *In SAT [SAT04*, 2004.

[129] E. Kushilevitz and N. Nisan, *Communication Complexity*. Cambridge University Press, 1996.

[130] M. N. Velev, "Efficient translation of boolean formulas to cnf in formal verification of microprocessors," in *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, 2004, pp. 310–315.

[131] P. Jackson and D. Sheridan, "Clause form conversions for boolean circuits," in *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, ser. SAT'04, Vancouver, BC, Canada: Springer-Verlag, 2004, pp. 183–198, ISBN: 354027829X.

[132] P. Manolios and D. Vroon, "Efficient circuit to cnf conversion," in *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing*, ser. SAT'07, Lisbon, Portugal: Springer-Verlag, 2007, pp. 4–9, ISBN: 9783540727873.

[133] B. Chambers, P. Manolios, and D. Vroon, "Faster sat solving with better cnf generation," in *2009 Design, Automation Test in Europe Conference Exhibition*, 2009, pp. 1590–1595.

[134] R. Arora and M. Hsiao, "Cnf formula simplification using implication reasoning," Dec. 2004, pp. 129–134, ISBN: 0-7803-8714-7.

[135] N. Eén and A. Biere, "Effective preprocessing in sat through variable and clause elimination," in *Theory and Applications of Satisfiability Testing*, F. Bacchus and T. Walsh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 61–75, ISBN: 978-3-540-31679-4.

[136] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. NLD: IOS Press, 2009, ISBN: 1586039296.

[137] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960, ISSN: 0004-5411.

[138] S. Subbarayan and D. K. Pradhan, "Niver: Non-increasing variable elimination resolution for preprocessing sat instances," in *Theory and Applications of Satisfiability Testing*, H. H. Hoos and D. G. Mitchell, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 276–291, ISBN: 978-3-540-31580-3.

[139] M. Järvisalo, A. Biere, and M. Heule, "Blocked clause elimination," Mar. 2010, pp. 129–144, ISBN: 978-3-642-12001-5.

[140] M. J. H. Heule, M. Järvisalo, and A. Biere, "Efficient cnf simplification based on binary implication graphs," in *SAT*, 2011.

[141] C. Piette, Y. Hamadi, and L. Saıs, "Vivifying propositional clausal formulae," in *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, NLD: IOS Press, 2008, pp. 525–529, ISBN: 9781586038915.

[142] C.-M. Li, F. Xiao, M. Luo, F. Manyà, Z. Lu, and Y. Li, "Clause vivification by unit propagation in cdcl sat solvers," *Artificial Intelligence*, vol. 279, p. 103 197, 2020, ISSN: 0004-3702.

[143] M. J. H. Heule, M. Järvisalo, and A. Biere, "Revisiting hyper binary resolution," in *CPAIOR*, 2013.

[144] K. R. Apt, "Some remarks on boolean constraint propagation," in *New Trends in Constraints*, K. R. Apt, E. Monfroy, A. C. Kakas, and F. Rossi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 91–107, ISBN: 978-3-540-44654-5.

[145] I. Lynce and J. Marques-Silva, "Efficient data structures for backtrack search sat solvers," *Annals of Mathematics and Artificial Intelligence*, vol. 43, no. 1–4, pp. 137–152, Jan. 2005, ISSN: 1012-2443.

[146] H. Zhang, "Sato: An efficient propositional prover," in *CADE*, 1997.

[147] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001 , pp. 530–535.

[148] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518, ISBN: 978-3-540-24605-3.

[149] L. Ryan, "Efficient algorithms for clause-learning sat solvers," 2004.

[150] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence*, vol. 193, pp. 45–86, 2012, ISSN: 0004-3702.

[151] J. Huang, "The effect of restarts on the efficiency of clause learning," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07, Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2318–2323.

[152] M. Luby, A. Sinclair, and D. Zuckerman, "Optimal speedup of las vegas algorithms," *Information Processing Letters*, vol. 47, no. 4, pp. 173–180, 1993, ISSN: 0020-0190.

[153] Y. Guo, B. Zhang, and C. Zhang, "A heuristic restart strategy to speed up the solving of satisfiability problem," in *2012 Fifth International Symposium on Computational Intelligence and Design*, vol. 2, 2012, pp. 423–426.

[154] S. Gocht and T. Balyo, "Accelerating sat based planning with incremental sat solving," in *ICAPS*, 2017.

[155] J. Rintanen, "An iterative algorithm for synthesizing invariants," in *AAAI/IAAI*, 2000.

[156] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340, ISBN: 978-3-540-78800-3.

[157] J. Rintanen, "Evaluation strategies for planning as satisfiability," *Proceedings of the 16th European Conference on Artificial Intelligence*, pp. 682–687, 2004.

[158] D. M. McDermott, "The 1998 ai planning systems competition," *AI Magazine*, vol. 21, no. 2, p. 35, Jun. 2000.

[159] F. Bacchus, "Aips 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems," *AI Magazine*, vol. 22, no. 3, p. 47, Sep. 2001.

[160] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis.," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 1–59, Dec. 2003.

[161] A. Coles, A. Coles, A. Olaya, *et al.*, "A survey of the seventh international planning competition," *Ai Magazine*, vol. 33, pp. 83–88, Mar. 2012.

[162] M. Vallati, L. Chrpa, M. Grześ, *et al.*, "The 2014 international planning competition: Progress and trends," *AI Magazine*, vol. 36, no. 3, pp. 90–98, Sep. 2015.

[163] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71, Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, pp. 151–158, ISBN: 9781450374644.

[164] D. Kozen, "Positive first-order logic is np-complete," *IBM Journal of Research and Development*, vol. 25, no. 4, pp. 327–332, 1981.

[165] J. Rintanen, "Evaluation strategies for planning as satisfiability," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ser.
ECAI'04, Valencia, Spain: IOS Press, 2004, 682–686
, ISBN: 9781586034528.

[166] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: Algorithmic verification and debugging," *Communications of the ACM*, vol. 52, no. 11, pp. 74–84, 2009.

[167] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[168] A. Orso and G. Rothermel, "Software testing: A research travelogue (2000–2014)," in *Future of Software Engineering Proceedings*, 2014, pp. 117–132.

[169] I. Buzhinsky, C. Pang, and V. Vyatkin, "Formal modeling of testing software for cyber-physical automation systems," in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 3, 2015, pp. 301–306.

[170] J. Zander, I. Schieferdecker, and P. J. Mosterman, *Model-based testing for embedded systems*. CRC press, 2017.

[171] X. Rival and K. Yi, *Introduction to static analysis: an abstract interpretation perspective*. Mit Press, 2020.

[172] A. Sistla, "Safety, liveness and fairness in temporal logic," *Formal Aspects of Computing*, vol. 6, Sep. 1999.

[173] M. Dahl, K. Bengtsson, M. Fabian, and P. Falkman, "Guard extraction for modeling and control of a collaborative assembly station," *IFAC-Papers On Line*, vol. 53, no. 4, pp. 223–228, 2020, 15th IFAC Workshop on Discrete Event Systems WODES 2020 — Rio de Janeiro, Brazil, 11-13 November 2020, ISSN: 2405-8963.

[174] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, IEEE, 1977, pp. 46–57.

[175] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *International conference on tools and algorithms for the construction and analysis of systems*, Springer, 1999, pp. 193–207.

[176] P. Runeson, "A survey of unit testing practices," *IEEE software*, vol. 23, no. 4, pp. 22–29, 2006.

[177] G. Fink and M. Bishop, "Property-based testing: A new approach to testing for assurance," *SIGSOFT Softw. Eng. Notes*, vol. 22, no. 4, pp. 74–80, Jul. 1997, ISSN: 0163-5948.