HAZARD ANALYSIS

OF COLLABORATIVE HUMAN-ROBOT SYSTEMS

A Simulation- and Agent-based Approach

TOM P. HUCK

License Information

Textual Materials

Figures

Cover design: Timo Wolf Kamp, persoonlijkproefschrift.nl

Important notice: This book deals with the topic of safety and hazard analysis of safety-critical systems.

It is important to emphasize that this is a *research publication*. The purpose of this publication is to propose and discuss novel ideas and concepts related to the aforementioned topics. These ideas and concepts, however, are not yet proven in practice.

Thus, this book is *not* intended as a guideline or as a reference work for designing, implementing, analyzing, or otherwise dealing with safety-critical systems. For such purposes, the reader should consult the national and international standards and legal frameworks which are relevant for their particular region and use-case.

Tom P. Huck

Hazard Analysis of Collaborative Human-Robot Systems: A Simulation- and Agent-based Approach

Second Edition

September 2024

To my parents.

Preface

This Book is based on work I conducted between 2019 to 2023 at Karlsruhe Institute of Technology (KIT) in Germany. The work was originally published as a PhD thesis under the same title. This E-Book is an edited and extended version of the original thesis.

Compared to the original publication, several changes have been made. The section on functional safety and standardization has been extended to describe functional safety basics in more detail and cover additional standardization topics (originally, this section was abbreviated to keep the thesis more concise). A section on generative AI and its impact on hazard analysis has also been added, as this topic has attracted a lot of attention recently.

Furthermore, the book has been revised to correct some of the minor errors that will almost inevitably occur in a first publication. At some points, explanations have been edited to improve clarity. Finally, some third-party figures have been replaced with figures under a creative commons license to facilitate easier re-use of the material (see licensing information at the beginning of this book).

Although I am aware that this book only covers a niche research topic, I sincerely hope that the contents will be useful to some.

Ottersweier, September 2024

Tom P. Huck

Abstract

Hazard Analysis of Collaborative Human-Robot Systems: A Simulation- and Agent-based Approach

With the trend towards human-robot collaboration, robot systems are getting both more complex and more safety-critical. To ensure safety of humans, and to meet normative and legal requirements, safety-critical robot systems need to be analyzed thoroughly for potential safety flaws and hazards before going into operation. In current practice, this hazard analysis is largely based on human reasoning, experience, expert knowledge, and simple tools such as checklists. With increasing system complexity, additional tools for hazard analyses are required.

This book proposes a simulation- and agent-based approach for hazard analysis of collaborative human-robot systems. The proposed approach draws on the concept of agent-based testing. In agent-based testing, the analyzed system, also known as the system under test (SUT), is embedded into a simulation environment where one or multiple *testing agents* are present. The testing agents are entities in the simulation model which interact with the SUT, thereby stimulating the SUT and allowing analysts to test whether the SUT responds in a safe manner. In the context of human-robot collaboration, the agent-based approach allows it to simulate human robot interactions and observe whether the robot interacts with humans in a safe manner. A crucial question in agent-based testing is how to create agent behaviors which are suitably critical to expose safety flaws and hazards in the SUT. To address this question, the book frames hazard analysis as a search problem. Given a simulation model of SUT and Agent, a safety specification, and a search space of possible agent behaviors, the goal of the search problem is to find agent behaviors to which the SUT responds in an unsafe manner. For solving this search problem, the book proposes the concept of *risk-guided search*, which uses domain-specific risk metrics

in conjunction with search- and reinforcement learning algorithms. By attempting to maximize the risk metric, these algorithms learn to create high-risk agent behaviors which lead to unsafe states and thereby expose hazards.

The risk-guided search concept is demonstrated and validated in experiments from the domain of industrial human-robot collaboration. In these experiments, simulation models of the test scenarios are created which contain certain safety-critical design flaws that are deliberately introduced into the models in order to create hazards. The risk-guided search is then deployed to identify these hazards.

While the results are promising, there are certain limitations to the risk-guided search approach. In particular, there is a fundamental trade-off between accuracy of the simulation and exhaustiveness of the search: A high level of detail and complexity leads to a search-space explosion, so that an exhaustive search generally becomes infeasible. To address this limitation, the book proposes a second approach, namely a two-layer analysis which analyzes the system both on a higher abstraction level where an exhaustive exploration is possible, and on a less abstract simulation model which provides a higher level of detail. The performance of risk-guided search and two-level approach, as well as their respective strength and weaknesses, are compared. Although the two-layer approach outperforms the risk-guided search, the experiments also indicate cases where the risk-guided search has certain advantages, indicating that both approaches are merited.

With respect to application in safety-critical use cases, it should be emphasized that the techniques developed in this book should be seen as an addition to augment the existing spectrum of hazard analysis methods, and not as a replacement for existing methods.

Keywords: *Robot Safety, Risk Assessment, Hazard Analysis, Simulation*

Zusammenfassung

Gefahrenanalyse von Kollaborativen Mensch-Roboter Systemen: Ein Simulations- und Agentenbasierter Ansatz

Mit dem Trend zur Mensch-Roboter-Kollaboration werden Robotersysteme sowohl komplexer als auch sicherheitskritischer. Um die Sicherheit von Menschen zu gewährleisten und normative und gesetzliche Anforderungen zu erfüllen, müssen sicherheitskritische Robotersysteme vor der Inbetriebnahme gründlich auf mögliche Gefahren hin analysiert werden. In der aktuellen Praxis basiert diese Gefahrenanalyse weitgehend auf menschlichem Verstand, Erfahrung, Expertenwissen und einfachen Werkzeugen wie Checklisten. Mit zunehmender Systemkomplexität werden zusätzliche Werkzeuge für die Gefahrenanalyse benötigt.

In diesem Buch wird ein simulationsbasierter Ansatz für die Gefahrenanalyse von kollaborativen Robotersystemen vorgeschlagen. Der vorgeschlagene Ansatz stützt sich auf das Konzept des agentenbasierten Testens. Beim agentenbasierten Testen wird das zu analysierende System, auch *System under Test* (SUT) genannt, in eine Simulationsumgebung eingebettet, in der sogenannte Testagenten vorhanden sind. Bei den Testagenten handelt es sich um Entitäten im Simulationsmodell, die mit dem SUT interagieren. Die Testagenten regen das SUT zur Reaktion an, wobei beobachtet werden kann, ob sich das SUT in einer sicheren Weise verhält. Im Kontext der Mensch-Roboter-Kollaboration erlaubt der agentenbasierte Ansatz die Simulation von Mensch-Roboter-Interaktionen. So kann virtuell getestet werden, ob der Roboter auf sichere Weise mit Menschen interagiert.

Eine entscheidende Frage beim agentenbasierten Testen lautet, wie man ein Verhalten der Testagenten erzeugt, welches hinreichend sicherheitskritisch ist, um bestehende Gefahren aufzudecken. Hier wird diese Herausforderung als Suchproblem formalisiert. Ausgehend von einem Simulationsmodell des SUT und des Agenten, einer Sicherheitsspezifikation und

einem Suchraum möglicher Verhaltensweisen des Agenten besteht das Ziel des Suchproblems darin, Verhalten zu finden, auf die das SUT in einer unsicheren Weise reagiert. Zur Lösung dieses Suchproblems wird das Konzept der risikogeleiteten Suche vorgeschlagen. Hierbei werden domänenspezifische Risikometriken in Verbindung mit Such- und Reinforcement-Learning Algorithmen verwendet. Durch die Maximierung der Risikometrik lernen diese Algorithmen, unsichere Zustände zu provozieren und decken somit Gefährdungen auf.

Das Konzept der risikogeleiteten Suche wird in Experimenten aus dem Bereich der industriellen Mensch-Roboter-Kollaboration demonstriert und validiert. In diesen Experimenten werden Simulationsmodelle von kollaborativen Robotersystemen betrachtet. In diese Modelle werden gezielt sicherheitskritische Fehler eingebracht, um potentielle Gefahren zu erzeugen. Anhand dieser gefahrenbehafteten Systeme wird getestet, ob die risikogeleitete Suche zielgerichtet Gefahren identifizieren kann.

Auch wenn die Ergebnisse der Experimente vielversprechend ausfallen, hat der Ansatz der risikogeleiteten Suche gewisse Grenzen. Insbesondere gibt es eine grundlegende Limitation im Hinblick auf die Genauigkeit der Simulation und die Vollständigkeit der Suche: Ein steigender Detailgrad führt zu einer Explosion des Suchraums, sodass eine vollständige Exploration im Allgemeinen nicht mehr durchführbar ist. Um dieser Einschränkung zu begegnen, schlägt diese Buch einen zweiten Ansatz vor. Hierbei handelt es sich um eine zweischichtige Analyse, die das System sowohl auf einer höheren Abstraktionsebene analysiert, auf der eine erschöpfende Suche möglich ist, als auch auf einem weniger abstrakten Ebene mit einem detaillierteren Simulationsmodell. Die Performanz der risikogeleiteten Suche wird mit der des zweistufigen Ansatzes verglichen. Obwohl die Performanz des zweistufigen Ansatzes die der risikogeleiteten Suche übertrifft, zeigen die Experimente auch Fälle, in denen die risikogeleitete Suche gewisse Vorteile hat, sodass beide Ansätze ihre Berechtigung haben.

Im Hinblick auf die Anwendung in sicherheitskritischen Anwendungsfällen ist zu betonen, dass die hier entwickelten Methoden als Ergänzung des bestehenden Spektrums von Gefahrenanalyseverfahren zu sehen sind und nicht als Ersatz für bestehende Verfahren.

Stichwörter: *Robotik, Sicherheit, Risikobeurteilung, Gefahrenanalyse, Simulation*

Contents

1. Introduction

HUMAN-ROBOT COLLABORATION (HRC) is a type of human-robot interaction where humans and robots collaborate to achieve shared goals [24]. Often, this entails human and robot working together in shared workspaces or even in direct physical interaction, which requi-res safe robot systems. To address this need, an increasing number of robot manufacturers offer so-called *cobots*, that is, robots with safety features intended specifically for collaborative use. Although the share of cobots compared to traditional industrial robots is still small, it has been increasing steadily over the last five years [98]. Various HRC use-cases are currently being explored, such as in manufacturing [95], construction [132], service [185], or personal care [38].

Functional safety is a crucial requirement for HRC systems. In the context of HRC, functional safety means that humans who are interacting with a robot shall not be subjected to unacceptable risk of injury [96]. Over recent years, a large number of research works have been published to address this need. Various safety measures for collaborative robots have been developed and standards have been published to detail the safety requirements to which robots must comply [123, 103, 105, 107].

However, endowing robots with safety measures is – on its own – not sufficient to achieve functional safety. One also needs to ensure that the safety measures are appropriate to the use-case and sufficient to address potential hazards that might occur in a given system context. In other words, there needs to be a procedure to analyze if the designed system is actually safe. Such a procedure is called *risk assessment*. A risk assessment typically entails an analysis of the system as well as an identification and estimation of potential risks. On this basis, safety experts decide if the risks are acceptable or if further safety measures are required [101]. In comparison to the rapid development of HRC, risk assessment methods are lagging behind, as most methods are still based on human reasoning, experience, and expert knowledge [92, 86]. Although this approach is viable

for relatively simple HRC systems, it does not scale well with increasing system complexity.

This book explores a novel approach to support the risk assessment of HRC systems. In particular, the book focuses on the *hazard analysis* phase of the risk assessment procedure. The goal of the hazard analysis is to analyze if a system can reach any unsafe states, and how these states manifest themselves in terms of hazardous situations (such as particular human-robot collisions). This is a crucial aspect of the risk assessment, because risks can only be assessed and mitigated if the underlying hazards are identified in the first place. However, hazard analysis is often challenging, because safety is an *emergent property* which results from dynamic interactions between various system components. Experience in safety engineering has shown that analyzing such emergent properties places a high burden on human analysts and frequently leads to hazards being overlooked [131]. Another challenge is the inherent non-determinism and variability of human behavior. In a human-robot system, one cannot assume that humans behave as expected. Human errors or otherwise unforeseen behavior can lead to hazardous situations and therefore also needs to be included in the analysis, adding a further layer of complexity to the problem.

One possible way to mitigate this complexity is to conduct hazard analysis on the basis of *system models*. Different types of models are conceivable for this purpose, including semi-formal models (e.g. UML or flow diagrams), formal models (e.g., automata or petri nets), and simulation models. In this work, simulation models are chosen as the main approach. Simulation has long been used for the development of robot systems. Since simulation is already a part of many development processes, simulation models are readily available in many cases and can be re-purposed for hazard analysis. Furthermore, simulations can provide a relatively high level of detail, especially with respect to spatial and physical aspects which are particularly important in HRC. However, despite these advantages, simulation-based hazard analysis is still relatively sparsely researched in the domain of robotics. It is the goal of this work to demonstrate that simulation can be a beneficial tool for hazard analysis of HRC systems. To achieve this, a number of open questions need to be addressed, as will be discussed in the following section.

1.1. Questions addressed in this Book

Q1: How to analyze increasingly complex robot systems for potential hazards?

Traditionally, safety engineering has been focused on ensuring that individual safety-critical systems, such as sensors or actuators, behave according to their specifications and do not exceed certain failure probabilities [96, 106]. However, complex safety-critical systems consist of a large number of subsystems which interact with each other and with the system environment in a variety of ways. Safety at the system level is therefore an *emergent property* [131]. In other words, even if each subsystem operates reliably and according to its specifications, unsafe behaviors may still occur due to unintended interactions between multiple subsystems or between subsystems and system environment. Emergent properties cannot be sufficiently assessed when considering only subsystems without taking into account interactions between them, and between system and system environment. As a consequence, there may be hazards which only emerge under specific circumstances, e.g. when the system interacts with its system environment in a specific manner. Such hazards can remain undetected in the design phase of a system but then emerge later in operation to cause accidents. Traditional safety engineering methods focus on hazards which arise due to system malfunctions, such as random component failures. Developing methods to identify emergent hazards on a system-level, however, is still an open challenge, in particular with respect to increasing system complexity. This work aims to addresses this challenge in the context of human-robot collaboration. However, the research question is also relevant in a wider context, since similar challenges also exist in other safety-critical domains (e.g., autonomous driving).

Q2: How to formalize and automate the task of hazard analysis?

As mentioned in the introduction, the analysis of safety-critical systems is part of a process known as risk assessment. To date, the practice of risk assessment has been largely informal. While there are general guidelines and simple tools such as checklists, the core tasks still rely mainly on human reasoning [92, 86]. As the complexity of safety-critical systems increases, this approach alone is no longer sufficient. Automated tools can be a valuable complement to human analysts when analyzing complex

systems. However, to enable automated analyses, the informal problem of identifying hazards based on human reasoning must be translated into a formal problem definition. This work addresses this challenge by proposing a framework in which hazard analysis is formalized in such a way that it can be solved algorithmically, thereby enabling the development of automated tools which can support the human reasoning that is applied in current risk assessment procedures.

Q3: How to mitigate search-space complexity in simulation-based testing?

Search space complexity is a crucial challenge in the analysis of safety-critical systems. Robot systems may encounter a wide range of different situations and interactions with their environment. This can lead to a vast set of different conditions under which the behavior of the robot system needs to be analyzed. Furthermore, it my be the case that unsafe system states are difficult to find, especially as they tend to occur seldomly. Efficient search techniques are needed to mitigate search space explosion problems and find conditions that lead to unsafe states. Furthermore, there is a fundamental conflict between the detail and the exhaustiveness of model-based hazard analyses. The more detailed a system is modeled, the more computational cost is associated with evaluation of the model's behavior in a given scenarios. Thus, an increasing level of detail makes it more difficult to achieve an exhaustive analysis. On the other hand, more abstract models with smaller state spaces can enable exhaustive analyses, but cannot represent the real-world system accurately. For effective hazard analyses, a suitable trade-off needs to be found.

1.2. Proposed Solutions

To address these questions, this book proposes a set of possible solutions. These solutions are based on the concept of *Agent-based testing* [13]: Agent-based testing is a technique in which the analyzed system, also referred to system under test (SUT) is embedded in a simulation environment along with so-called *test agents*. The test agents are dynamic entities within the simulation that interact with the SUT. By creating different agent behaviors, the SUT is tested in a variety of different interaction scenarios. Agent-based testing is particularly suitable for the analysis of human-robot interactions. By modeling the human as a testing agent, one can subject the robot system to test cases which capture a wide range of different human-robot interactions. In such a setting, the challenge is to find agent behaviors which cause the SUT to enter an unsafe state. Thus, the hazard identification problem essentially takes the form of a *search problem* where the search space consists of possible agent behaviors. The following techniques are proposed for solving this search problem:

- **Risk-guided search:** A technique where the agents are trained to behave in a way that maximizes a domain-specific risk-metric. By maximizing the risk metric, the agents create critical testing conditions which lead to the discovery of unsafe states.

- **Automata-constrained risk-guided search:** An extension of risk-guided search where the agent is constrained to certain feasible behaviors which are encoded in the form of Extended Finite Automata (EFA). The EFA limits the search space to a particular set of potentially relevant behaviors. This further mitigates search complexity and excludes unrealistic or infeasible behaviors.

- **Two-level hazard analysis:** This method relies on system models on two distinct abstraction levels: An abstract formal model in the form of EFA, and a more detailed simulation model. The formal level is treated as a pre-processing step to the simulation-based analysis. By leveraging synthesis methods for discrete event systems, a set of potentially unsafe agent behaviors is extracted from the automata model. These agent behaviors are then used as input to the simulation model, where the resulting simulation runs are investigated in more detail.

1.3. Scope of this Work

This work has several contact points with other, related research areas. In order to clearly delimit the scope of the work and to avoid misunderstandings, some distinctions from other research fields are drawn in the following.

First, although this book presents simulation-based methods, it is not concerned with how to *create* simulation models of HRC systems. Of course, detailed and accurate simulation models are essential for successful application of these methods in practice. However, the development of simulation models, especially with respect to digital modeling of humans, is a separate field of research with its own research challenges. Here, it is assumed that suitable models are already given, because the focus of this work is on how to leverage existing simulation models effectively for hazard analyses, and not on how to develop the models themselves.

Furthermore, this book is not concerned with developing hazard analysis tools for immediate practical application. Instead, the focus is placed on *conceptual* solutions which can serve as a basis for practical tools in the future. Since the focus here is on conceptual aspects, there will be certain simplifications throughout the book, especially regarding experiments and application examples. The associated limitations and the required future work to address these limitations will be discussed towards the end of this book.

Finally, it should be pointed out that in this work, risk assessment is understood as a *design-time* activity. This distinction is important because there is also a body of work where risk assessment is understood as a *run-time* activity (e.g. [122, 173, 56]). These works investigate how robots can continuously monitor their environment, identify potentially hazardous situations in real time, and react accordingly. While both design-time and run-time risk assessments are important, they are fundamentally different in their respective approaches: Run-time risk assessment is intended to assess the *current* state of a system at a given point of time during the system's operation phase. The goal is to identify hazardous situations as they are about to arise so that the system can trigger a safe reaction. In contrast, design-time risk assessments are conducted *prior* to the system's operation. Here, the goal is to identify and mitigate potential hazards during the system's design phase. In other words, one might say that run-time risk assessments are a part of the system's control strategy, whereas design

time risk assessments are a part of the system's development process. Both problems are challenging in their own way: for runtime risk assessments, a robust real-time perception and interpretation of the robot's environment is necessary. For the design-time risk assessment, this is not needed, since the risk assessment is conducted prior to the system's operation (i.e., "offline"). However, the main challenge here lies in the vast search space of possible system states, and in trying to identify possible unsafe states given incomplete or imperfect information about the system and the conditions it may be subjected to in future operation.

1.4. Concepts from Literature

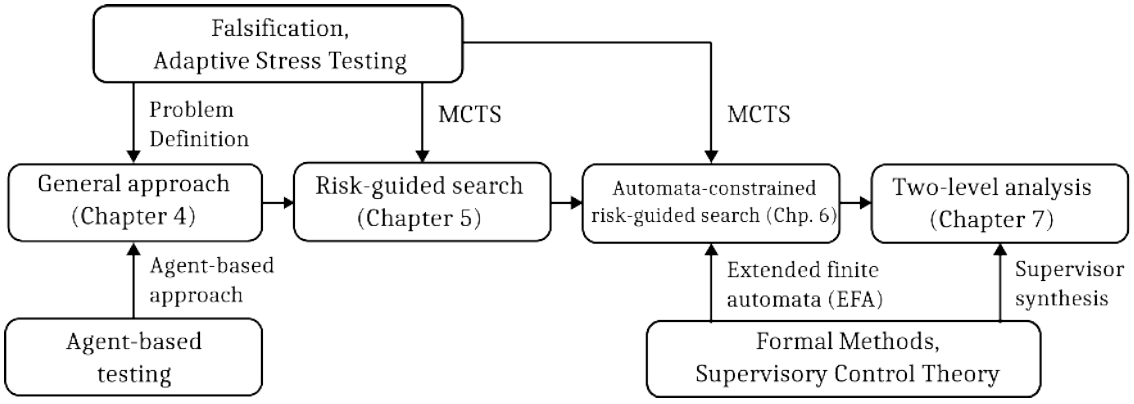


Figure 1.1.: Methods from prior literature used in this book. (Author's own figure.)

This book draws on various concepts from literature. The respective literature sources will be referenced as these concepts appear throughout the book. Additionally, Figure 1.1 additionally gives a brief overview of the main concepts used throughout this book.

The agent-based approach of this work is inspired by the concept of agent-based testing (see section 3.3.1 and [13]). The problem definition is inspired by the field of falsification, in particular by a method known as *Adaptive Stress Testing (AST)* (more in section 3.3.3 and [126]).

For the risk-guided search and automata-constrained risk-guided search, the search algorithm *Monte Carlo Tree Search* (MCTS) is used (more details in section 2.2.2). This algorithm was partly selected because its use for falsification purposes was demonstrated successfully by AST [126].

Finally, this book also draws from a body of work related to formal methods, more specifically *Supervisory Control Theory* (more details in section 2.2.3 and 2.2.4). In particular, the concept of *Extended Finite Automata* (EFA) is used for automata-constrained risk-guided search, and the concept of *Supervisor synthesis* is used for the two-level hazard analysis.

1.5. Overview of Content

Chapter 2 introduces preliminaries. The Chapter is split into two sections. Section 2.1 defines some terminology which is used throughout the book and introduces relevant background information about safety standards and regulatory requirements with regard to safety in HRC. Section 2.2 focuses on theoretical background, especially with respect to modeling formalisms and algorithms that are used later.

Chapter 3 presents the current state-of-the-art regarding hazard analysis methods, covering both HRC and other safety-related systems. Sections 3.1-3.3 present related research works while Section 3.4 discusses current industrial practice. Section 3.5 concludes the chapter by discussing limitations of the current state of the art.

Chapter 4 presents the general outline of the concept proposed in this book. Section 4.1 provides a formal problem definition and Section 4.2 outlines the proposed solutions for the stated problem.

Chapters 5-7 present the core contributions of this work. Each chapter focuses on a specific technique or aspect of the proposed methods:

- **Chapter 5** introduces the concept of risk-guided search, discusses the heuristic risk metric, and describes the selection of suitable search algorithms.
- **Chapter 6** extends the previously introduced risk-guided search by introducing additional formalisms for more detailed agent modeling, especially with respect to constraints on the agent's behavior.
- **Chapter 7** compares the proposed simulation-based techniques with formal verification and introduces the two-level approach as a combination of both.

Chapter 8 discusses the contributions with respect to their potential impact as well as limitations and open challenges and concludes the book with a summary and an outlook on possibilities for future work.

It is recommended to read this book in sequential order. This goes in particular for Chapters 5-7, which build each other. However, each of the Chapters 5-7 contains a brief chapter summary. Readers who are looking for a quick overview, or readers who are only interested in particular aspects of the work, may skip some chapters and read the respective summaries instead.

Colored boxes are used throughout the book to highlight particular aspects:

Blue boxes contain definitions.

Yellow boxes contain examples.

Green boxes reference prior publications of the author on which this book is based.

2. Preliminaries

THIS CHAPTER DISCUSSES PRELIMINARIES which are needed as a basis for further contents of this book. The first section provides some basic understanding of Functional Safety in the context of HRC. This includes terminology, safety standards, and current procedures for risk assessment and hazard analysis. While not all of these topics are directly coupled to the methods proposed in this book, they are important as they provide a general understanding of the context of this work and the associated challenges.

The second section introduces theoretical concepts which are used in this book, such as Markov Decision Processes, Finite and Extended Finite Automata, and Supervisory Control Theory. If these concepts are known, the reader may skip the theoretical part at this point and refer back to it if necessary.

2.1. Functional Safety Basics

2.1.1. Terminology

Before discussing the issue of safety in HRC systems, some terminology needs to be clarified. Terms like *safety*, *hazard*, or *risk* are often used with an intuitive understanding, but without a clear definition. Sometimes, terms are used interchangeably and even the relevant standards differ in some definitions. Within this book, the following definitions are used. They are based on the standards ISO 12100 [101] and IEC 61508 [96], as these are some of the most commonly used safety standards in Europe. Readers should be aware that terms and standards may vary depending on region and responsible standardization body.

Safety

IEC 61508 defines *safety* as the "freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment" [96]. Note that this book deals only with safety in the sense of direct physical injury (e.g., as result of human-robot collisions). Tangentially related aspects, such as ergonomic problems or psychological issues (e.g., due to stressful working conditions), are not considered here. Also, it is necessary to differentiate between safety and security. Security is the protection against intentional threats such as cyber attacks, unauthorized access, or sabotage. Safety and security are often closely related (e.g. a security threat in a critical infrastructure may also become a safety concern). This work, however, focuses explicitly on safety and does not deal with security issues.

Risk

ISO 12100 describes *risk* as a combination of the severity of a damage and its probability of occurrence [101]. This definition, however, is rather vague as it does not specify how exactly risk, severity and probability are linked. There is currently no universally accepted definition of how risk should be be quantified. For instance, some approaches use decision trees or matrices to map different risk criteria to discrete risk levels [104]. Others define risk on the basis of probability distributions [134]. This book defines risk as a metric which assigns a numerical value to a given situation or system configuration which indicates its criticality, based on domain-specific criteria (e.g., human-robot distance, collision forces, etc.). Although this does not fall strictly within the definition of risk given above, it is a pragmatic approach which is sufficient for the purposes of this book. Details on the risk metric follow in Chapter 5.

Hazard and Hazardous Situation

A *hazard* is a potential source of harm [101]. In industrial systems, hazards come in various different types, such as mechanical hazards, electrical hazards, hazards through noise, etc. This book focuses on mechanical hazards posed by robots, namely collisions and clamping/crushing hazards. Note that the presence of a hazard does not necessarily imply that there is

a concrete and immediate danger to a person. Instead, a hazard can result from certain safety-critical system properties which may remain uncritical during most of the operational time and only emerge as a concrete danger in specific situations.

A *hazardous situation* is a situation in which a hazard manifests itself as a concrete danger to a person [101]. For instance, a fast-moving robot is a latent danger (i.e., a *hazard*), but human workers are only concretely in danger if they come near it without sufficient slow-down or safety-stop of the robot (i.e., a *hazardous situation*). This difference is particularly noteworthy as the general approach of this work will be to expose latent system hazards by finding concrete action sequences that result in concrete hazardous situations where the latent hazard becomes manifest and observable (details in Chapter 4).

Risk Assessment

Risk Assessment is a procedure which is conducted during the development of safety-critical systems to ensure that system designers are aware of risks and can plan appropriate safety measures. The exact steps of a risk assessment procedure differ across domains and safety standards. Generally speaking, however, risk assessment includes the determination of system limits, the identification of hazards, the assessment of the risks associated with these hazards, and the identification of risks that need further risk reduction measures [101, 96].

Risk Mitigation

Risk Mitigation is the procedure of planning and implementing safety measures to bring unacceptable risks below an acceptable threshold. The risk mitigation procedure generally follows the risk assessment procedure [101].

Hazard Analysis/Hazard Identification

A *hazard analysis*, also referred to as *hazard identification*, is a sub-task of risk assessment that focuses on identifying hazards of the analyzed system and the concrete hazardous situations in which these hazards emerge [96, 101]. The identification of hazards and hazardous situations is a crucial

part of the risk assessment procedure, since it is a necessary precondition for risk evaluation and mitigation.

2.1.2. Development of Safety-Critical Systems

As hinted to in the introduction to this book, development of safety-critical systems must adhere to rigorous development procedures. These procedures are governed by safety standards. Below, these development procedures will be explained using the example of IEC 61508, a generic safety standard which is the basis of many industry-specific standards [96, 184]. Note that the explanation is far from exhaustive. It is only meant to give the reader a general sense of safety-related development and of the associated cost and time.

Safety Integrity Levels

The Safety Integrity Level (SIL) is a means of characterizing the safety-related performance of a system or function. It ranges from SIL1 to SIL4. Each SIL is associated with a certain probability of dangerous failures that the respective system shall not exceed. These probabilities are either expressed as probability of dangerous failure per hour (PFH_D) for systems that operate continuously, or as failure on demand (PFD) for systems that are only invoked rarely, so that expressing a failure rate per hour would not be meaningful (e.g., airbag system in a car). They decrease as the SIL increases, i.e., a higher SIL is associated with a lower probability of dangerous failure (see Table 2.1). However, these probabilities should be seen as development goals rather than proven system properties, as it is difficult to predict such failure probabilities with great accuracy, especially for complex systems (note that this fact does not diminish the rigor by which these goals should be pursued). Generally, the rigor of the development process and the required fault tolerance will depend on the targeted SIL. [184]

Random and Systematic Faults

One of the most important aspects of safety-related development is to prevent or limit the impact of safety-critical failures. Broadly, there are two types of failures that need to be considered [184, p. 7-8]:

SIL	PFH_D (Failures per hour)	PFD (Failure on Demand)
SIL1	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 10^{-2}$ to $< 10^{-1}$
SIL2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to $< 10^{-2}$
SIL3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to $< 10^{-3}$
SIL4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to $< 10^{-4}$

Table 2.1.: Safety Integrity Levels and associated probabilities of dangerous failure per hour (PFH_D) and on demand (PFD) [184, p. 8].

- Systematic failures originate from errors in the system's design or implementation. Examples are errors in specifications or software bugs.
- Random hardware failures originate from non-deterministic causes. Examples are the failure of components due to ageing or manufacturing defects, or faults in digital memory due to exernal disturbances.

Systematic failures need to be avoided by the appropriate development processes which minimize the chances of errors in system design or implementation. To that end, part 1 of IEC 61508 defines a safety lifecycle approach intended to ensure safety integrity throughout all phases of the development. The prevention of systematic failures in software is handeled separately in part 3 of IEC 61508. Note that the methods presented in this book are mainly geared towards exposing and preventing systematic failures (more details later). Random hardware failures are addressed in part 2 of IEC 61508.

Hardware

Random hardware failures can be neither completely avoided, nor predicted. Thus, their impact needs to be minimized by increasing fault tolerance, i.e., the robustness of a system against random failures. Part 2 of IEC 61508 details how this can be achieved. Examples include:

- Appropriate selection of hardware components (e.g. over-dimensioning, selection of more reliable components)
- Redundancy (multiple components performing the same functions)
- Diversity (multiple components performing the same functions, but in different ways. This helps to reduce so-called "common-cause

failures", i.e., failure of multiple redundant components at the same time due to the same underlying cause)

- Self-tests and diagnostic measures: The system can detect the presence of faults and automatically goes into a safe state (e.g. switch-off).

Furthermore, part 2 of IEC 61508 requires a quantitative calculation to verify that the desired failure rates are met. One important aspect of this quantitative calculation is the so-called safe-failure fraction (SFF):

$$SFF = \frac{\sum \lambda_S + \sum \lambda_{DD}}{\sum \lambda_S + \sum \lambda_{DD} + \sum \lambda_{DU}} \tag{2.1}$$

In this equation, the terms are defined as follows:

- λ_S are safe failure rates, that is, rates of failures that are not safety critical
- λ_{DD} are dangerous, but detected failure rates (i.e., rates of failures that are potentially critical but can be detected trough self-tests and other diagnostic measures, and the system can be put into a safe state after detection)
- λ_{DU} are failure rates of dangeorus, but undetectable failures, that is, failures which cannot be discovered by the systems self-tests and diagnostic measures.

Thus, the SFF expresses the ratio between the rate of safe failures (which are either inherently uncritical or can be detected) to the overall rate of failures. Together with the Hardware Fault Tolerance (HFT), the SFF determines the achievable SIL (IEC 61508-2:2010, Table 2).

Software

Software faults are, by definition, systematic faults (see definitions above). As such, they need to be avoided through managing quality in software development. This may include, for example:

- Structured and modular software design and specification
- Testing (e.g. black- or white-box testing, integration testing)
- Use of formal methods in design and verification

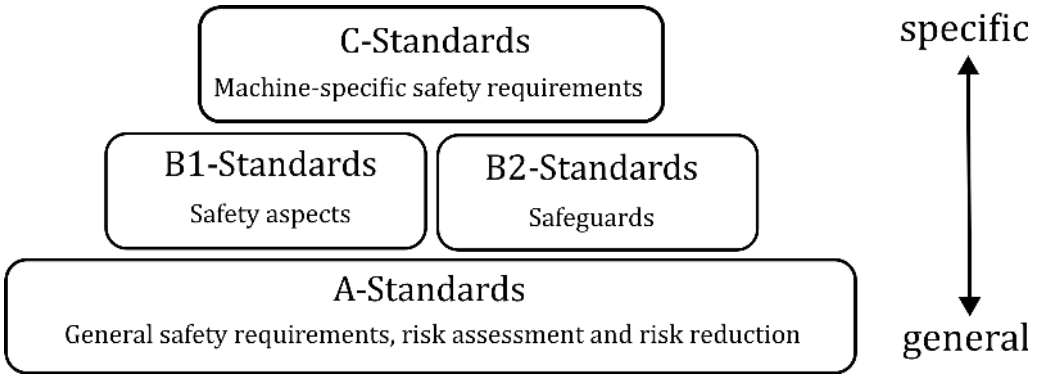


Figure 2.1.: Difference between A-, B-, and C-Standards. (Author's own figure, modeled after [157].)

- Use of appropriate coding guidelines (e.g., MISRA, AUTOSAR, JSF) to avoid unsafe language features and enforce a defensive programming style

Generally, the required rigor of these measures will depend on the respective SIL. Appendix A in part 3 of IEC 61508 provides guidance as to what measures are recommended at each SIL. For example, formal verification is recommended for SIL 3 and 4, but not for SIL 1 and 2, whereas functional testing and black-box testing is highly recommended for all SIL levels (Table A.5, IEC 61508-3:2010).

2.1.3. HRC-related Safety Standards

This Section introduces a number of more specific safety standards which are related to the machinery sector and to HRC systems. This section is structured into A-, B-, and C-standards (Figure 2.1). A-standards are generic standards which define basic guidelines independently of concrete machine types or safety measures. B-standards address more specific safety aspects (e.g., calculation of safety distances) or types of safeguards (e.g., safety fences). C-standards are have the most narrow focus, as they give specifications for a specific group or type of machine [157].

A-Standards

ISO 12100 is the basic ISO standard for risk assessment and risk reduction of machinery [101]. It defines a general procedure for risk assessment and risk reduction as well as a checklist of potential hazards. Since ISO

12100 is an A-Standard, its guidelines are applicable to a wide range of machinery. Thus, the procedures and the hazard checklist are fairly generic and not specifically geared towards robot systems. A more detailed discussion of the procedures defined by ISO 12100 follows in Section 2.1.6.

B-Standards

ISO 13849 specifies requirements for functional safety of control systems, including both hardware and software aspects. Similarly to IEC 61508, ISO 13849 defines safety levels which are called *Performance Level* (PL) and range from PLa to PLe, where PLc is comparable to SIL1, PLd to SIL2, and PLe to SIL3. The achievable performance level is determined on the basis of the system architecture, the hardware reliability, expressed as Mean Time to Dangerous Failure, the diagnostic coverage, and the resilience against common cause failures [106]. Since ISO 13849 is derived from IEC 61508 and the general approach of both standards is similar, it will not be discussed further here.

ISO 13854 considers relative movement between objects which can lead to hazardous situations where human body parts are trapped and potentially crushed. ISO 13854 defines body-region specific safety distances to avoid this. If there is relative movement between objects whose distance is less than the minimum distance, a crushing hazard must be assumed [108]. In the context of HRC, these minimum distances are important because they help to distinguish between transient collision hazards (i.e., collisions in free space), and quasi-static collision hazards (i.e., collisions where the human body is trapped between the robot and another obstacle). Details on these collision types are discussed in Section 2.1.5.

ISO 13855 deals with contactless safety sensors like laser scanners or light curtains, which are often used to monitor the area surrounding the robot for approaching humans [168] (see Fig. 2.2). ISO 13855 defines how these safety sensors should be placed and how their detection areas should be dimensioned, depending on various factors such as the expected approach speed, approach direction, and response time of the safety sensor [102]. The standard therefore has implications for the layout of collaborative robot cells.

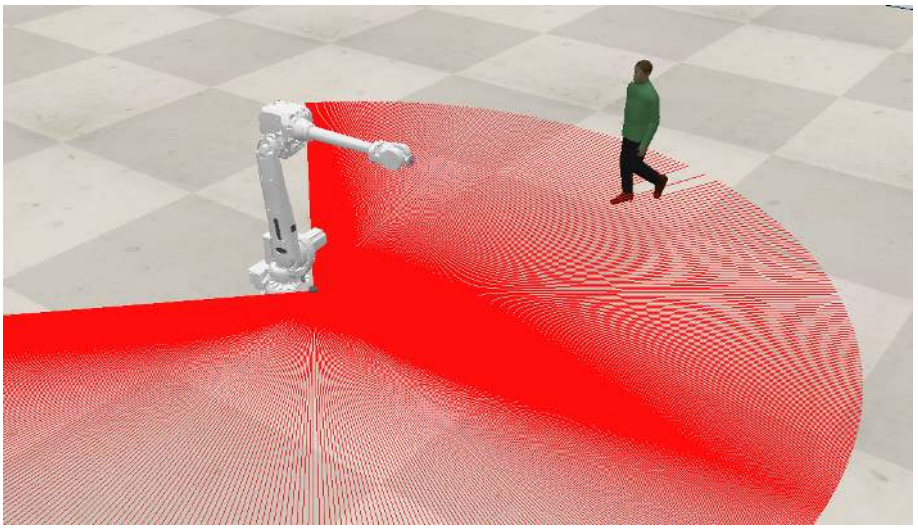


Figure 2.2.: A Laserscanner monitors an area surrounding a robot. The schematic shows the laser beams of the scanner, although these are not visible in reality (Author's own figure).

ISO 13857 specifies minimum distances to avoid the reaching of hazardous areas by human limbs [110]. Despite the close proximity of human and robot, safety distances still play a role in HRC, for example when safety fences are used to close off areas which are not monitored by safety sensors. In such cases, ISO 13857 must be considered to avoid hazards where human workers can reach over or around safety fences and provoke critical collisions.

C-Standards

ISO 10218 is a C-Standard specifically for robots and robot systems. ISO 10218 consists of two parts. Part 1 defines safety requirements for the robots themselves, whereas part 2 defines safety requirements for the robot *system*, which includes not only the robot, but also other components like tools, sensors, fences, etc. Part two explicitly states that any robot system must undergo a risk assessment according to the ISO 12100 procedure. Similarly to ISO 12100, part two of ISO 10218 also provides a hazard checklist, but with hazards that are more specifically geared towards robots systems [103]. The currently valid iteration of this standard is from 2011. As of 2024, a new draft of ISO 10218 is under development.

Technical Specifications and Technical Reports

Apart from A-, B-, and C-standards, there are also technical specifications and technical reports. These are defined as follows [99]:

- Technical specifications (TS) are precursors to standards. They contain information which is expected to be incorporated into a standard in the future. While they can be used immediately, they should be seen as a work in progress towards an actual standard.
- Technical reports (TR) are primarily a means of information sharing. They can contain various information related the state of the art in a particular technical field. While TR may be published in preparation to standardisation activities, they are not as close to being transformed into a standard as a TS would be.

ISO/TS 15066 amends ISO 10218 with further details specifically for collaborative robot systems. ISO/TS 15066 defines four possible safety modes for collaborative operation, namely [107]:

- *Safety-rated monitored stop (SRMS):* In this mode, safety is achieved by monitoring the surrounding area of a robot (e.g. with a laser scanner, Figure 2.2) and stopping the robot as soon as a human enters a certain safety zone. While this operation mode is the easiest to implement, it also is the most restrictive.
- *Hand Guidance:* In this mode, the robot is guided by hand while certain safety restrictions (e.g., on the maximum admissible velocity) are continuously monitored. Hand guidance is often used for teach-in of waypoints in robot programming, but otherwise only plays a limited role.
- *Speed and Separation Monitoring (SSM):* In this mode, human-robot distance and robot velocity are continuously monitored. At any given time, the maximum admissible robot velocity is calculated as a function of the human-robot distance and the robot velocity is limited accordingly. SSM may also be realized with multiple discrete safety zones and step-wise velocity reduction instead of continuous velocity sclaing.
- *Power and Force Limiting (PFL):* In this mode, safety is maintained by ensuring that collision force and pressure remain below certain

body-region specific limits. Thus, collisions are acceptable as long as they are uncritical in terms of the stated limits.

Especially the PFL mode is interesting from a hazard analysis perspective, as it is much more permissive with respect to collisions than SRMS or SSM. In PFL mode, collisions need not necessarily be avoided for a robot application to be safe. Rather, it is only required that certain body-region specific thresholds are not exceeded. This introduces an additional layer of complexity into the hazard analysis and risk assessment procedure as it is not sufficient to only identify whether collisions are possible or not. Instead, possible collision scenarios need to be identified, including the body regions that are affected and the robot velocity at time of collision, in order to determine if the collisions are acceptable.

It is expected that contents of ISO/TS 15066 will be incorporated into the 2024 draft of ISO 10218.

ISO/IEC TR 5469

ISO/IEC TR 5469 deals with functional safety of artificial intelligence (AI) [112]. This document has been published recently as a TR. A related TS, namely ISO/IEC AWI TS 22440 is currently under development [100]. Although ISO/IEC TR 5469 is not directly related to HRC, it is relevant for HRC systems where collaborative robots use AI systems in some capacity (e.g., for tracking or predicting human movement). Furthermore, simulation-based testing as a means to analyze the safety of AI systems is discussed. The technical report categorizes safety-related applications of AI systems according to two criteria [112]:

- The *Usage Level* considers how critical the AI application is and in how far it can influence safety-related behaviors of the system. The range of usage levels comprises six steps from level D, where the AI system has no influence on safety-related system behavior, to level A1, where the AI system can have a direct impact on safety-related decision making.
- The *AI Technology Class* considers how strict the development process of the AI system is, and how well they system is understood and handled in terms of functional safety. This category ranges in three steps from Class I, where the AI system is developed and reviewed according to already existing safety standards, to Class III, where the

> AI system is so complex that it cannot be developed in accordance the existing safety standards. The middle ground is Class II, where the AI system cannot be fully developed in accordance to existing safety standards, but it is possible to identifiy, verify, and validate additional requirements sufficiently [112].

According to the technical report, systems of Class I should be treated similarly to traditional, non AI-based safety systems by applying the relevant safety standards. The same applies for all systems with Usage level D, which are uncritical regardless of the AI technology that is used. Systems of Class III and a usage level above D shall *not* be implemented for safety-critical applications, at least not with the currently known safety methods. For systems of Class II and usage level above D, different safety measures are recommended, whose rigor depends on the respective usage level. These measures are details in Clause 8-11 and Annex B of the report. Interestingly, ISO/IEC TR 5469 also recommends - among other measures - simulation-based testing, a technique which also forms the basis of this book.

2.1.4. Implementation of Safe HRC Systems

In industrial robot applications, safety has conventionally been ensured by physically separating human workers and robots through the use of safety fences. However, more sophisticated approaches are needed to implement the aforementioned collaborative operation modes specified by ISO/TS 15066.

The operation modes SRMS and SSM are usually implemented with perception systems which detect approaching humans and trigger a reactive robot behavior [150]. In industrial use cases, commonly used perception systems are laser scanners, light curtains, and pressure sensitive floor mats. Light curtains and floor mats are suitable for implementing SRMS. They trigger safety stops when human workers are detected. More sophisticated sensors like laser scanners are typically used for SSM. They allow for a step-wise or continuous velocity reduction as the worker approaches the robot. The use of camera systems for the perception of humans is currently being researched. While camera systems for perception of humans are commonplace nowadays, their use in safety-critical applications is still an open challenge, as it is challenging to build perception systems which comply with the strict safety requirements of HRC [174, 175]. Examples of

safety-compliant commercially available camera systems for the detection of human workers are the *PILZ SafetyEye* [155] and the *SICK safeVisionary* [181].

The operation modes SRMS and SSM are not always feasible, especially in cases where human worker and robot need to collaborate in close proximity to perform their task. In such cases, PFL is the preferred operation mode. In this operation mode, a limitation of collision force and pressure is required. This can be achieved through different measures. First, the mechanical design of the robot itself can reduce collision criticality, e.g. by limiting the robot's mass, using compliant joints, or padding the robot links with soft material. Examples of this are seen in *ABB Yumi* robot [3] which is lightweight and has padded surfaces, and in the *Rethink Robotics Sawyer* robot [167], which uses compliant joints with springs to reduce collision forces.
Apart from constructive measures, software functions are also a feasible way to implement the PFL operation mode. This is typically done through a combination of software-based speed limitation and collision detection. The collision detection typically uses joint torque measurements (either measured directly or estimated via motor currents) to sense collisions and stop the robot shortly after a contact is detected, while the speed limitation limits the robot's kinetic energy that is transferred to the human body in case of a collision. Examples of these safety functions can be found in the *Universal Robot URe-Series* [189], the *KUKA LBR iiwa* [120], and the *ABB GoFa* [2] robot. Recently, capacitive sensing has also been proposed to implement PFL. Capacitive sensors are able to detect potential collision objects at a certain distance and are therefore able to initialize a safety stops before the actual impact occurs. While this strategy may not be able to avoid collisions entirely, it reduces collision forces significantly compared to collision detections which are based only on torque measurements [7].

With respect to the further contents of this book, it should be noted that many of the safety measures presented here and below follow relatively simple principles (e.g. access protection or area protection by light curtains or laser scanners). For systems with such safety solutions, current hazard analysis methods are often already sufficient. Therefore, the methods proposed in this book may seem like an overly complicated approach at first glance. However, the comparative simplicity of current safety solutions should not distract from the fact that more sophisticated and complicated safety solutions are expected in the future. By proposing a scalable ap-

proach and first demonstrating it first on relatively simple safety solutions, this book aims to lay the groundwork for analyzing more complex systems in the future.

2.1.5. Human-Robot Collisions

Collisions are the primary hazard in HRC scenarios and have been studied extensively in crash-tests [80, 81, 79]. These studies show that collision criticality depends on several factors including the affected body region, the robot's speed and mass, and whether the collision is unconstrained (also known as *transient*) or constrained. In a constrained collision, the human body is clamped or crushed between the robot and another obstacle (also known as *quasi-static* collision). This is the more critical type of collision because the human body cannot recoil from the collision point and thus more of the robot's kinetic energy is transferred to it. In early stages of HRC, various different metrics have been proposed to quantify collision criticality [78, 141, 45]. Nowadays, however, collision force and pressure are the generally accepted criteria and have been included into the standardization [27, 28, 107] (see also Section 2.1.3).

For the hazard analysis of HRC systems, it is required to assess if potential human-robot collisions are acceptable with respect to the ISO/TS 15066 limits. Thus, collision force and pressure must be determined. If a physical prototype of the HRC system is available, this can be done with specialized collision measurement equipment which mimics the collision behavior of the human body and measures both force and pressure [75, 156]. However, the hazard analysis procedure typically begins in an early development stage where real-world collision measurements are not possible. Thus, model-based collision analyses are required. For that purpose, ISO/TS 15066 proposes a simple two-body collision model which consists of an effective robot mass m_R, an effective mass of the affected human body m_H, and a spring constant k to model the elasticity of the human body. Note that m_H and k are specific to the affected human body part. For transient collisions, m_R and m_H are combined to a reduced effective mass μ (see ISO/TS 15066, Eq. A.3):

$$\mu = \left(\frac{1}{m_R} + \frac{1}{m_H} \right)^{-1} \tag{2.2}$$

For quasi-static collisions where the human body is constrained, the effective human mass approaches infinity, so the effective collision mass equals

the robot mass (i.e., the total kinetic energy of the robot is transferred to the human body):

$$\lim_{m_H \to \infty} \mu = m_R \tag{2.3}$$

The effective robot mass is approximated as follows (compare ISO/TS 15066, Eq. A.2):

$$m_R \approx \frac{M}{2} + m_L \tag{2.4}$$

Here, M is the total mass of the robot's moving parts and m_L is the payload of the robot. The effective mass m_H and spring constant k of the human body are body-region specific values which are defined in Table A.3 of ISO/TS 15066.

From the equivalence between the maximum spring energy and the maximum kinetic energy during the collision, the peak collision force F_{coll} and the peak collision pressure p_{coll} for a collision with relative velocity v_{rel} and contact area A are calculated as follows (compare ISO/TS 15066, Eq. A.2):

$$\frac{F_{coll}^2}{2k} = \frac{1}{2}\mu {v_{rel}}^2 \tag{2.5}$$

$$F_{coll} = v_{rel}\sqrt{k\mu} \tag{2.6}$$

$$p_{coll} = \frac{F_{coll}}{A} \tag{2.7}$$

The ISO/TS 15066 model is strongly simplified. In reality, human-robot collisions are highly complex nonlinear processes. Especially the pressure exerted on the human body is difficult to estimate, because the contact area A depends on the human body deformation. Furthermore, the estimation of m_R (see Eq. 2.4) is simplified. In reality, the effective mass depends on the robots joint angle configuration q [127]:

$$m_R = \left(u^T (J(q)M(q)J^T(q)u\right))^{-1} \tag{2.8}$$

Here, $M(q)$ and $J(q)$ denote the robots inertia and jacobian matrix, respectively, and u denotes the collision direction vector.

The development of more accurate yet efficient collision models is still an open research challenge. Novel approaches include the use of nonlinear spring constants to model body elasticity [127], nonlinear skin deformation models [180], finite-element analysis [152], and regression methods

which are trained on data from collision measurements [119, 187]. In the remainder of this book, however, the simplified collision model will be used because it is computationally inexpensive and all required model parameters are specified by ISO/TS 15066.

2.1.6. Risk Assessment, Risk Mitigation, and Hazard Analysis of HRC systems

As emphasized multiple times throughout this book, safety is an *emergent* property which arises from the interaction of all relevant components in a system [131]. Thus, a set of safe components does not necessarily comprise a safe system. Translated to the context of robotics, this means that robots are not inherently safe. Instead, potential hazards and appropriate safety measures are highly dependent on a number of factors such as task, spatial layout of the robot cell, programming and system environment of the robot, and more. To ensure that the relevant hazards in a system are recognized and the associated risks are sufficiently addressed, a risk assessment and risk mitigation procedure is required. An important step in this procedure is the hazard analysis, in which potential hazardous situations need to be identified [103].

As discussed in Section 2.1.3, risk assessment is a design-time activity, that is, the system is analyzed *before* commissioning. If systems go into operation without a thorough risk assessment, system designers might be unaware of potential hazards. This can lead to serious omissions in the safety measures of a system, and, ultimately, to accidents. In the European Union, performing risk assessments is mandatory from a legal perspective. The EU machinery directive 2006/42/EC mandates that a risk assessment must be performed. Failure to comply with this directive might result in lawsuits and barriers to market entry [61]. Furthermore, an early identification of hazards helps to avoid costly changes or re-designs at later development stages, which reduces development time and costs.

General risk assessment guidelines and procedures are specified by the standard ISO 12100 [101]. These procedures include the following steps:

1. **Determination of limits:** The limits of the analyzed system and its intended use are specified. This includes for example the purpose of the system, its intended use as well as foreseeable misuse, environmental conditions, operator qualifications, and more (Section 5.4 of ISO 12100).

2. **Hazard analysis:** Potential hazards and hazardous situations are identified (Section 5.5 of ISO 12100).
3. **Risk estimation:** The risks associated with the identified hazards and hazardous situations are estimated (Section 5.6 of ISO 12100).
4. **Risk evaluation:** The estimated risks are evaluated and it is decided which of the risks require risk mitigation measures (Section 6 of ISO 12100).

After these four steps are completed, safety measures are chosen and implemented to mitigate any risk that is deemed to be unacceptable (examples of such risk mitigation measures have already been shown in Sec. 2.1.4). If the chosen risk reduction measures are not purely mechanical, but related to a control function (e.g., safety stop or SSM), then the control function is assigned a suitable safety level (i.e., a SIL or PL) that depends on various factors that are to be determined in the risk assessment, such as the criticality of the related hazardous situation and its probability of occurence. To achieve these target safety levels, the related control functions must be developed according to a suitable standard (e.g. ISO 13849 or IEC 61508, see sections 2.1.2 and 2.1.3). If available, a suitable pre-certified safety system from a supplier can also be selected.

After measures for risk reduction have been chosen, the procedure must be repeated, since the risk mitigation measures might change the properties of the system and could introduce new hazards in other places. Thus, risk assessment and mitigation are iterative procedures, as shown in Figure 2.3.

Within the general framework for risk assessment and mitigation, this work focuses on the hazard analysis phase. Hazard analysis is perhaps the most crucial part of the overall risk assessment and mitigation procedure, since risks cannot be estimated, evaluated, or mitigated if system designers are unaware of the underlying hazards.

The following non-exhaustive list includes some exemplary questions which safety engineers need to consider when analyzing the safety of HRC systems:

- Are any human-robot collisions possible in this system?
- If yes, in what situation or at what point of the collaborative task are these collisions likely to happen?
- Which parts of the human body are affected by the potential collisions?

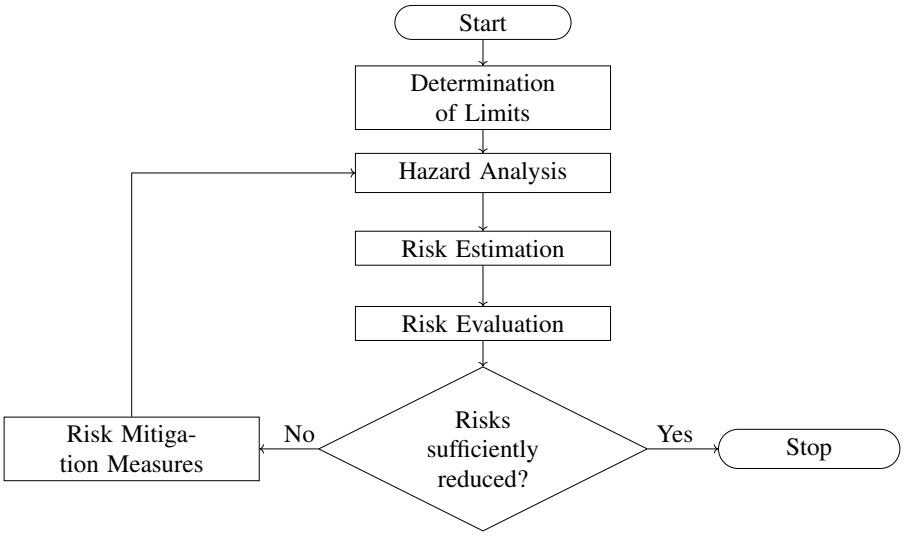


Figure 2.3.: Iterative risk assessment and mitigation procedure according to ISO 12100. (Author's own figure, based on [101]).

- What collision force and pressure are expected for the potential collisions? Are these values acceptable for the affected body parts?
- Are there any foreseeable misuses or possible human errors that can cause critical situations which are not included in the nominal task?
- Is it possible to circumvent safety measures and if yes, how?

Note that - like the whole risk assessment and mitigation procedure - the hazard analysis is a *design-time* activity, meaning that it is carried out before the developed system becomes operational. Usually, the hazard analysis begins even before a physical prototype of the system is available. This means that safety engineers need to answer the aforementioned questions and predict hazardous situations without having a real-world system available that could be used for analysis or measurements. Safety standards such as ISO 12100 generally assume that risk assessment is a largely manual procedure, performed mainly on the basis of human reasoning, expert knowledge, and experience. Simulation- or other model-based hazard analysis methods are not mentioned by the standard [101]. Instead, the standard mostly recommends intuitive techniques such as brainstorming. For additional guidance through the the hazard analysis procedure, generic checklists of commonly occurring hazards and typical hazardous situations are available [101, 103].

The need for risk assessments and hazard analyses is not specific to industrial robot systems. These procedures are also required for a wide range

of other safety-critical systems. This includes all kinds of industrial machinery [61], non-industrial robots (e.g., personal assistance robots) [105] and autonomous vehicles [109, 111].

As shown in Figure 2.4, there are a number of issues which make hazard analysis in the context of HRC particularly challenging (list based on [92], extended with additional content):

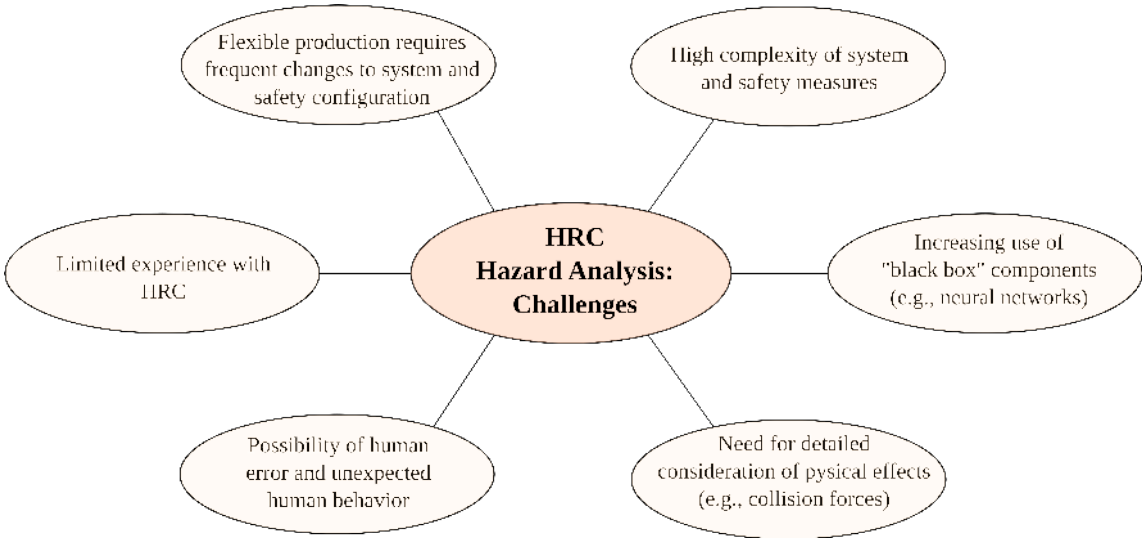


Figure 2.4.: Challenges affecting the hazard analysis of HRC systems. (Author's own figure.)

- *Complexity of system and safety Measures:* safety in HRC depends on a large number of factors, including design choices such as cell layout, choice of robot, and operation mode, etc. as well as dynamic effects, such as robot stopping time, collision forces, sensor response times, etc. [158, 107] All of these factors interact with each other and influence overall system safety in a complex manner. Experience in safety engineering has shown that in such complex systems can only be assessed properly when considering all possible interaction of the subsystems, and that human analysts only have a limited capacity to do this [131].

- *Systems with black-box components:* black-box components are system components whose internal structure and functionality is not fully known to analysts. The behavior of black-box systems is therefore difficult to foresee, which makes it challenging to analyze their safety [126]. In future, it must be expected that an increasing number of robot systems will contain such black box components. One

reason for this is the advance of artificial intelligence and machine learning methods, whose behavior in real-world conditions is often difficult to predict. Furthermore, pre-compiled third party software whose source code is not available must also be considered a black box component.

- *Need for detailed consideration of physical aspects:* in hazard analyses of HRC systems that operate under PFL mode, it not sufficient to simply determine if collisions are possible or not. Instead, it must determined what body regions are potentially affected and the potential collision force and pressure must be estimated. This requires detailed considerations about the movement and physical properties of humans and robots, which adds another layer of complexity to the hazard analysis procedure.
- *Need for consideration of human error/unexpected behavior:* the inherent non-determinism of human behavior, especially the possibility of human error or other unforeseen behaviors, is a general challenge in the design and analysis of human-machine interactive systems [84, 36]. In HRC scenarios, however, this is especially critical since the human is such an integral component of the HRC systems, and unforeseen human behaviors can easily lead to safety-critical situations [149, 19].
- *Limited experience:* HRC is, despite being seen as a major trend, still relatively novel in industrial practice. Thus, compared to other kinds of machinery, there is still a major lack of experience and expert knowledge when it comes to the hazard analysis of HRC applications [25, 1]. This is especially critical since the tool support for HRC hazard analysis is limited and the procedure is still mainly based on human reasoning (as will be discussed in the following Chapter).
- *Flexible production requires frequent changes to system/safety configuration:* since HRC applications are intended for flexible production of small- to medium sized product lots, it is expected that many HRC systems will undergo frequent changes, which may affect safety-relevant aspects of the system configuration. This, in turn requires frequent renewal of the hazard analysis, leading to increased efforts in terms of time and cost [61].

When considering the aforementioned issues, it becomes clear that performing hazard analyses for HRC systems is not a trivial task. As discussed previously, current hazard analysis methods are not sufficient to

address these challenges. Development of assistive hazard analysis tools is therefore an active area of research. An in-depth discussion of the current state of the art and novel hazard analysis methods follows in Chapter 3.

2.2. Theoretical Background

2.2.1. Markov Decision Process

The Markov Decision Process (MDP) is a model for sequential decision making problems [190]:

Definition 1 ***Markov Decision Process***
An MDP is described by the following 5-tuple (adapted from [190]):

$$\langle S, A, T, R, s_0 \rangle$$

Where S is the set of states, A the action space, T the transition function, R the reward, and s_0 the initial state. An agent in an MDP starts in the initial state s_0 and sequentially selects actions $a \in A$. The execution of an action alters the state according to the transition function T and returns a reward according to the reward function R.

Note that the transitions between states in the MDP are usually stochastic. This means that for a given state-action pair s, a, the transition function $T(s, a)$ does not give a deterministic next state, but rather some probabilities by which different resulting states may be reached. Thus, the transition function maps to each state-action pair a vector of probabilities for reaching the various states in S:

$$T : S \times A \to [0, 1]^{|S|} \tag{2.9}$$

where $|S|$ denotes the number of states. However, the transition does not need to be stochastic. Deterministic transitions are also possible. In the case of deterministic transitions, the transition function maps a next state to each state-action pair:

$$T : S \times A \to S \tag{2.10}$$

The reward function R is typically used to describe costs and benefits of taking certain actions in certain states. For any given state-action pair (s, a), the reward function issues a reward $R(s, a)$, that is, the reward for taking action a in state s. The set of values which the reward can take are generally problem-specific. In this work, it is assumed that the reward is a real-valued number:

$$R : S \times A \to \mathbb{R} \tag{2.11}$$

Note that the reward function in itself may also be stochastic, that is, the rewards may differ between different executions of the same action in the same state. An MDP must fulfill the so-called *Markov property*, which requires that for any given state and action, the transition probability $T(s, a)$ and the reward $R(s, a)$ must depend only on the current state and action, and not on the history of previous states and/or actions [190].

A *policy* π is a decision-making strategy which maps to each state an action that is taken by the agent:

$$\pi : S \rightarrow A \tag{2.12}$$

A common problem related to MPDs is to find an *optimal policy* π^* which maximizes the expected sum of rewards over a finite horizon of n decisions:

$$\pi^* = \text{argmax}_\pi \sum_{k=0}^{n} R_k, \quad \gamma \in \mathbb{R},\ 0 < \gamma < 1 \tag{2.13}$$

where R_k is the expected reward incurred in the $k - th$ timestep. Another common problem is to find a policy that maximizes the expected sum of rewards over an infinite horizon:

$$\pi^* = \text{argmax}_\pi \sum_{k=0}^{\infty} \gamma^k R_k \tag{2.14}$$

In the case of infinite horizons, a real-valued discount factor $0 < \gamma < 1$ is introduced to avoid divergence of Eq. (2.14) [190]. The problem of finding optimal policies can be solved by Reinforcement Learning algorithms and heuristic search algorithms. Such an algorithm, namely Monte Carlo Tree Search, is discussed in the following.

2.2.2. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm to approximate optimal decision making strategies in sequential decision making problems such as MDPs [126]. MCTS is a search method which operates on a search tree. Each node of the tree corres-ponds to a state of the decision making process, and each vertex of the tree corresponds to an action. By sampling action sequences and storing information about the incurred rewards, MCTS iteratively builds the search tree and estimates the

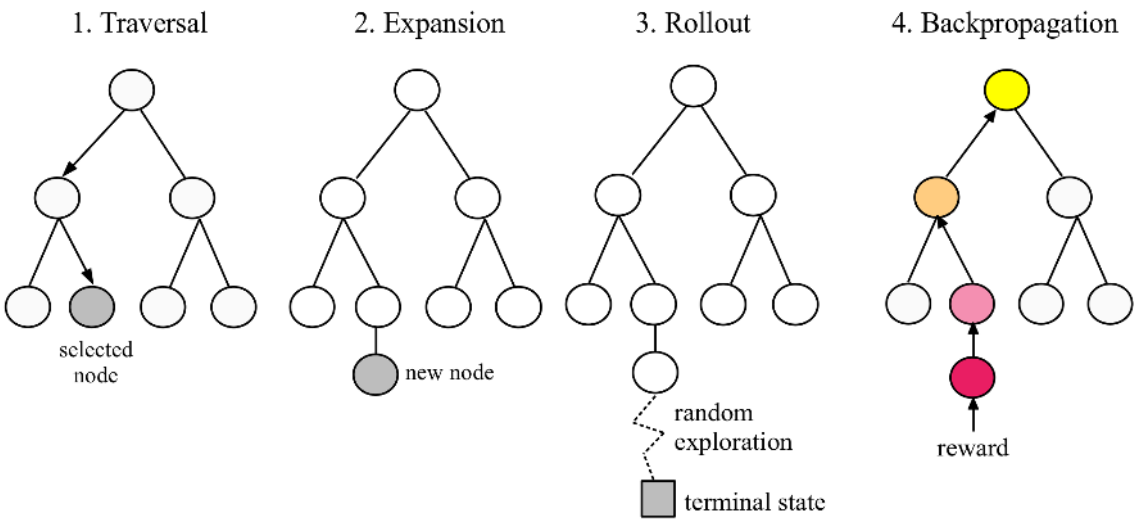


Figure 2.5.: Principle of Monte Carlo Tree Search: (1) Tree is traversed by selecting from existing nodes according to a selection criterion. (2) A new node is added. (3) Random actions are sampled, starting from the newly selected node. (4) The incurred reward is backpropagated. (Authors own figure, modeled after [198])

values of the nodes. For the sampling strategy, MCTS combines selection based on previously incurred rewards with random sampling and thereby balances exploitation and exploration [71]. The principle of the MCTS algorithm is shown in Figure 2.5. Pseudocode is given in Algorithm 1.

MCTS iteratively repeats the following four steps [71, 198]:

- **Traversal**: The traversal starts at the root node. As long as the current node is fully expanded (i.e., all actions in the current state have been tried at least once), the next state is chosen according to some pre-defined selection rule (Algorithm 1, ll. 3-5). A common selection rule is the UCT criterion (**U**pper **C**onfidence Bounds applied to **T**rees). This criterion selects the action which leads to the child node that maximizes the following expression (Algorithm 1, ll. 11-14):

$$\frac{q_i}{n_i} + c \cdot \sqrt{\frac{log(n_p)}{n_i}} \tag{2.15}$$

  where q_i is the value of the $i-th$ child node, n_i is the number of visits to the $i-th$ child node, and n_p is the number of visits to the parent node. The tuning parameter $c \in \mathbb{R}$ is chosen by the user to balance between exploration and exploitation.

Algorithm 1 Monte Carlo Tree Search (Pseudocode based on explanation in [33])

```
1:  while not stoppingCriterion() do
2:      currentNode ← rootNode;
3:      while fullyExpanded(currentNode) do // Traversal
4:          currentNode ← SELECTBESTCHILD(currentNode);
5:      end while
6:      currentNode ← EXPAND(currentNode);
7:      reward ← ROLLOUT()
8:      BACKPROPAGATION(currentNode, reward);
9:  end while
10:
11: function SELECTBESTCHILD(currentNode) // according to UCT criterion
12:     bestChildNode ← argmax ( node.q / node.n + c·sqrt( log(parentNode.n) / childNode.n ) );
13:     return bestChildNode;
14: end function
15:
16: function EXPAND(currentNode)
17:     a ← randomSample(currentNode.untriedActions);
18:     childNode ← createNewNode(parent←currentNode, incomingAction ← a);
19:     currentNode.children.append(childNode);
20:     return childNode;
21: end function
22:
23: function BACKPROPAGATION(currentNode,reward)
24:     currentNode.n ← currentNode.n + 1;
25:     currentNode.q ← currentNode.q + reward;
26:     if currentNode == rootNode then
27:         return;
28:     else
29:         BACKPROPAGATION(currentNode.parent, reward);
30:     end if
31: end function
32:
33: function ROLLOUT( )
34:     while not isTerminal() do
35:         doRandomAction();
36:     end while
37:     reward ← getReward();
38:     return reward;
39: end function
```

- **Expansion**: When the traversal reaches a node that has not yet been fully expanded (i.e., that has some untried actions), a new node is added by randomly sampling and executing one of the untried actions (Algorithm 1, l. 6 and ll. 16-21).

- **Rollout**: From the new node, a so-called *rollout* is performed to obtain an initial estimate of the node's value. In the rollout, actions are sampled according to some pre-defined rollout policy until a termination criterion is fulfilled. A common rollout policy is to sample actions randomly (Algorithm 1, l. 7 and ll. 33-39).
- **Backpropagation**: The reward incurred in the rollout phase is backpropagated along the nodes that have been taken in the traversal and expansion phase. Depending on the implementation of the algorithm, the backpropagation rule may vary. A common approach is to increment each node's q-value by the incurred reward, and the number of visits for each node by one (Algorithm 1, l. 8 and ll. 23-31).

One property of MCTS which is important in the context of this book is that MCTS builds a search tree on the fly (i.e., during runtime) and only requires information about the rewards that have been incurred and the actions that have been taken in a given search iteration. In contrast to other methods for optimal decision making such as Q-learning, which estimate a state-action-value function, MCTS does not require explicit state information. Instead, states are encoded *implicitly* in the search tree (note that each node can be encoded as a the sequence of actions that have led to the respective node without using information about the internal state of the decision making process). This makes the MCTS algorithm suitable for exploring black-box systems whose internal state is either unknown, or too complex to be represented explicitly [126].

2.2.3. Automata

Automata models are frequently used to describe discrete-event systems (DES), that is, systems with discrete states and state transitions [128]. In contrast to MDPs, however, they do not include stochastic transitions or reward functions (at least not in their original form, extensions such as stochastic automata are however known [177]). In this work, several processes in collaborative robotics (e.g., sequential assembly tasks) are abstracted as DES. Therefore, a short introduction to automata theory is given below. In particular, two types of automata are discussed: *Finite Automata* (FA) and *Extended Finite Automata* (EFA), which extend FA by additional modeling formalisms.

Finite State Automata

Definition 2 - *Finite State Automaton*
A finite state automaton (FA) is given by a 4-tuple [128]:

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0 \rangle \tag{2.16}$$

Where Q is a discrete set of states, Σ is the input alphabet, q_0 the initial state and δ the state transition function. The set of states Q is finite. The transition function δ describes the transitions between states. Transitions are triggered by events $e \in \Sigma$.

Note that in literature, the elements of Σ are often called *symbols*. In this work, however, transitions of automata are used to model discrete events. Thus, the term *event* is more fitting. The set of all events is called *input alphabet* Σ. The transition function δ is a function which returns the next state for a given current state and event:

$$Q \times \Sigma \to Q \tag{2.17}$$

Note that δ may not be defined for certain combinations of states and events, since it can be that some events are not feasible certain states. Thus, the structure of the automaton limits the set of feasible event sequences.

Input alphabet and acceptance. The set of events Σ is called the *input alphabet* of the automaton. Let Σ^* denote the set of all sequences of events of arbitrary length that can be constructed from Σ. Then, a sequence of events $(e_0, e_1, e_2...),\ e_i \in \Sigma$ is called a *word* over the input alphabet Σ. Executing the sequence results in a *run* through the automaton, that is, a sequence of states $(q_0, q_1, q_2, ...)$ which are reached by executing the respective events and taking the related transitions in the automaton. Note that for the run to be completed, the next event must always be feasible, that is, $\delta(s_i, e_i)$ must return a valid state for all (e_i, s_i) in the word and the run, respectively. If this is the case, then the word is *accepted*. The set of all accepted words is called the *Language* $L(\mathcal{A})$ of the automaton [128].

Synchronization. In some cases, it is desirable to model systems in a *modular* manner, that is, broken down into subsystems, where each subsystem is represented as a submodel automaton. The system behavior as a whole is then given by the *synchronization* of the automata submodels.

Consider two finite state automata $\mathcal{A}, \mathcal{B}$. Then, the synchronization of the automata is given by [128]:

$$\mathcal{A}||\mathcal{B} = \langle Q_{\mathcal{A}} \times Q_{\mathcal{B}}, \Sigma_{\mathcal{A}} \cup \Sigma_{\mathcal{B}}, \delta, (q_{0,\mathcal{A}}, q_{0,\mathcal{B}}) \rangle \tag{2.18}$$

That is, the state space of the synchronized automaton is the cartesian product of the submodel automata's respective state spaces, and the input alphabet is the union of the submodel automata's respective state spaces. The initial state is the state that corresponds to both submodel automata's respective initial states. The transition function is defined as follows [128]:

$$\delta((q_{\mathcal{A}}, q_{\mathcal{B}}), e) = \begin{cases} \delta_{\mathcal{A}}(q_{\mathcal{A}}, e) \times \delta_{\mathcal{B}}(q_{\mathcal{B}}, e) & e \in \Sigma_{\mathcal{A}} \cap \Sigma_{\mathcal{B}} \\ \delta_{\mathcal{A}}(q_{\mathcal{A}}, e) \times \{q_{\mathcal{B}}\} & e \in \Sigma_{\mathcal{A}} \backslash \Sigma_{\mathcal{B}} \\ \{q_{\mathcal{A}}\} \times \delta_{\mathcal{B}}(q_{\mathcal{B}}, e) & e \in \Sigma_{\mathcal{B}} \backslash \Sigma_{\mathcal{A}} \end{cases} \tag{2.19}$$

Intuitively, this means that if an event e is part of both submodel automata's respective input alphabets $\Sigma_{\mathcal{A}}$ and $\Sigma_{\mathcal{A}}$, then the resulting state is determined by both submodel automata's transition functions. Note that if either $\delta(q_{\mathcal{A}}, e)$ or $\delta(q_{\mathcal{A}}, e)$ are undefined, then $\delta(q_{\mathcal{A}}, e) \times \delta_{\mathcal{B}}(q_{\mathcal{B}}, e)$ is also undefined. In other words, both submodel automata must agree that a transition is possible. If an event is only part of the input alphabet of $\mathcal{A}$, then the next state is $(q_{\mathcal{A}}, q_{\mathcal{B}})$ is obtained by executing transition function $\delta_{\mathcal{A}}$of the submodel automaton $\mathcal{A}$, while the part of the state that relates to the submodel automaton $\mathcal{B}$ remains the same (and vice versa).

Marked states. The definition of finite automata can be extended by specifying sets of *marked states* $Q_m \subseteq Q$ [128]:

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_m \rangle \tag{2.20}$$

Generally, marked states express some desired state that the automaton must reach. If marked states are used, then the definition of the language $L(\mathcal{A})$ is extended by the requirement that all words in $L(\mathcal{A})$ *must* create a run that ends in a marked state. That is, a sequence of events is only accepted if it results in the automaton entering a marked state. When synchronizing automata with marked states, the marked states of the resulting synchronized automaton are defined as the cartesian product of the individual automata's marked states [128]:

$$Q_{m,\mathcal{A}||\mathcal{B}} = Q_{m,\mathcal{A}} \times Q_{m,\mathcal{B}} \tag{2.21}$$

Note that throughout in this work, marked states will be omitted from some automata models for reasons of simplicity. In these cases, the following convention holds: if no marked states are explicitly defined in an

automaton, then all states will be treated as if they were marked. If, however, marked states are explicitly defined, then only those explicitly defined states are treated as marked states, whereas the remaining states are treated as unmarked states.

Extended Finite Automata

Extended finite automata (EFA) extend the aforementioned FA with additional mode-ling formalisms [41, 183]:

- *Integer variables* of finite range can be defined to maintain information about certain system variables, without having to explicitly create a new state for each possible valuation of that variable.
- *Guards* are logical expressions over the variables. For a transition to be executed, the guard expression must evaluate to true. In other words, guard statements protect transitions from being executed until certain conditions are fulfilled. Thus, they can be used to express conditions on the feasibility of certain events.
- *Update rules* are functions which change the values of variables when a transition is executed[1].

The following definition introduces a notation of EFA as it is used throughout this book and its related publications (see also [93]).

Definition 3 - *Extended Finite Automata (EFA)*
An EFA is a tuple

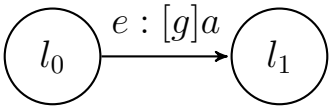
$$E = \langle \Sigma, V, L, \rightarrow, G, M, l_0, V_0, L^m \rangle \tag{2.22}$$

where Σ is a finite set of events, V is a finite set of bounded discrete variables, L is a finite set of locations, $\rightarrow \subseteq L \times \Sigma \times G \times M \times L$ is the conditional transition relation, where G and M are the respective sets of guards and update rules, $l_0 \in L$ is the set initial location, V_0 the initial assignment of the variables, and $L^m \subseteq L$ is the set of marked locations.

[1] Note that literature on EFA often uses the term *actions* rather than update rule. Here, however, the term *action* is already in use, which might lead to confusion. Thus, the term *update rule* is chosen, which is a fitting term since the execution of a transition updates the values of certain variables.

The state of an EFA is given by its current location $l \in L$ and the current values its variables $v_i \in V$. The set of states of an EFA is thus given by $Q = L \times V$.

The definition of input acceptance and accepted languages in EFA is analogous to the definition for FA introduced above, but with some differences. First, the guard statements need to be considered as additional conditions to determine if an event sequence is accepted. Second, instead of marked *states*, EFA have marked *locations*. Graphically, these marked locations are denoted by circles. Transition from one location l_0 to another location l_1 are denoted with a labeled arrow shown below (figure adapted from [93]):

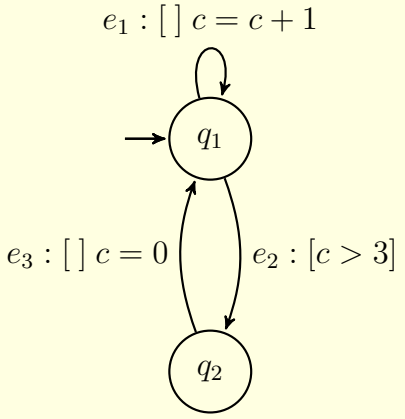


Note that the transition is labelled by an event $e \in \Sigma$, a guard $g \in G$ and a update rule $m \in M$. Guard statements are logical expressions over the variables in V. A transition can only occur if the guard statement g evaluates to *true*. When the event e occurs and the guard statement evaluates to *true*, the current location changes and the update rule m is executed, updating some of the values of the variables. Note that guards and update rules are not necessarily defined for all transitions, they may also be omitted. Example 1 illustrates the structure of EFA. As this example already suggests, finite automata and EFA are equal in terms of their expressiveness. In fact, it can be shown that extended finite automata can be converted into ordinary finite automata that are equal in terms of the accepted language [183, 147]. The main advantage of EFA however is that they allow for a more concise system representation, which makes the modeling of complex systems more manageable for human users. By breaking down systems into submodels and using EFA to describe these submodels, complex systems can be modeled in a relatively compact manner. The (more complex) automaton of the system as a whole does not require manual modeling, as it can be derived algorithmically by converting the submodel EFA into ordinary automata and synchronizing them as described in the previous subsection. This makes the application of automata modeling more scalable and potentially interesting for real-world use cases. Note that this advantage only refers to the process of *modeling*, but not necessarily the process of *analysis*. This is due to the fact that EFA do not reduce the

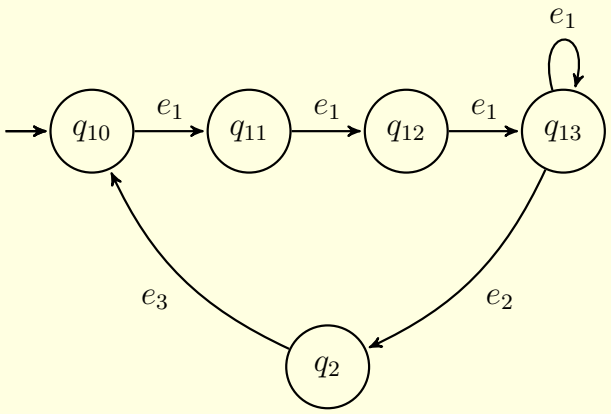
state space itself, they merely represent states in a different manner (i.e., as locations paired with valuations of the variables instead of explicit states) [183].

Example 1 ***- Structure of EFA***

The EFA below consists of two locations l_1, l_2 and has three events: e_1, e_2 and e_3. The automaton is extended into an EFA by introducing a variable c (initial value: $c = 1$) and two update rules associated to events e_1 and e_3. The update rule associated to event e_1 increments the counter and the action associated to event e_3 resets the counter. Further, a guard statement $[c > 3]$ is introduced which ensures that the transition associated to event e_2 can only be taken if the counter is larger than 3:



The result is an automaton that requires at least three occurences of the event e_1 until the event e_2 can be executed. After e_2, the event e_3 is required to reset the counter and return to the initial state. Note that due to the additional modeling formalisms which allow for a more concise specification of behaviors, EFA models are often more compact than FA models. This is illustrated by the following automaton below, which models the same behavior as above, but is an ordinary FA instead of an EFA, resulting in a less compact model.



2.2.4. Supervisory Control Theory

Supervisory Control Theory (SCT), also known as Ramadge-Wonham Framework, is a theoretical framework for analysis, control, and synthesis of discrete event systems (DES) [165]. Consider a DES known as the *Plant*, which represents a system that shall be controlled. The plant is modeled by an automaton $\mathcal{G}$. Let Σ be the set of events that may occur in the plant. The set of all plant events Σ shall be the input alphabet of the plant automaton $\mathcal{G}$. A word over the input alphabet corresponds to a sequence of events $(\sigma_1, \sigma_2, ...), \sigma_i \in \Sigma$. Thus, a word over the input alphabet describes a certain behavior of the plant. Consequently, the language $L(G)$ describes the set of all possible behaviors of the plant [39, 183].

Supervisory control theory provides means of restricting the plant behavior such that it satisfies certain properties. This is done by synthesizing a so-called *Supervisor* which dynamically restricts the behavior of the plant by enabling and disabling events. The supervisor $\mathcal{K}$ observes the events generated by the plant and assigns to each observed event sequence a control decision that specifies which of the events in Σ shall be admissible:

$$\mathcal{K} : pre(L(\mathcal{G})) \rightarrow 2^{\Sigma} \tag{2.23}$$

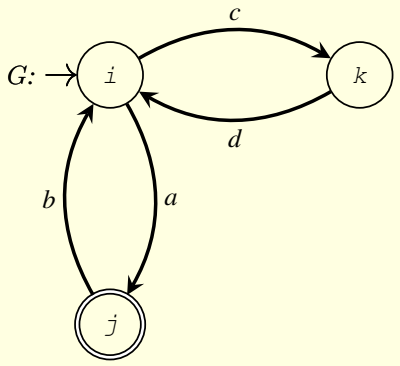
where $pre(L(\mathcal{G}))$ denotes the set of prefixes of the plant automaton's language (i.e., the set of all accepted event sequences and partial event sequences that are a prefix of an accepted sequence), and 2^{Σ} denotes a set of binary vectors indicating for each event whether it is currently admissible or not. Intuitively, one can think of a supervisor as a policy which observes the events generated by the plant in previous timesteps, and then decides to en- or disable certain events of the plant in the next step, based on the previously observed sequence [39, 147].

The aim of SCT is to restrict the plant behaviour to a certain subset of admissible behaviors. The specification of admissible behaviors is given by a specification automaton $\mathcal{SP}$, whose language $L(\mathcal{SP})$ corresponds to the admissible subset of $L(\mathcal{G})$. Supervisor synthesis creates a supervisor $\mathcal{K}$ which restricts the plant behavior in such a way that the specification $\mathcal{SP}$ is satisfied and the controlled systems is *non-blocking*. This means that the controlled system is able to reach a marked state from any reachable state (i.e., the supervisor prevents the system from entering deadlocks). To that end, the synthesis algorithm creates the synchronous composition $\mathcal{G} \parallel \mathcal{SP}$ of plant and specification and then iteratively removes blocking states from

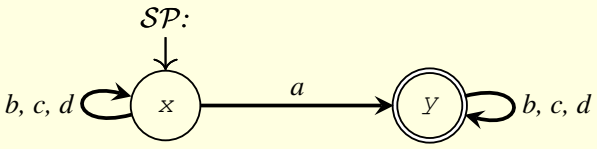
$\mathcal{G} \parallel \mathcal{SP}$ until the remaining system is non-blocking [147, 199]. Note that literature on SCT typically considers the input alphabet to consist of two disjoint subsets of events: $\Sigma = \Sigma_C \cup \Sigma_U$, where Σ_C refers to controllable events which can be enabled and disabled by the supervisor, and uncontrollable events Σ_U which cannot be influenced by the supervisor [39]. Apart from non-blockingness, the synthesis algorithms also ensure controllability of the plant. In the applications presented in this book, however, all events are considered controllable. Thus, the notion of controllability is not discussed further.

Software tools for supervisor synthesis employ fixpoint algorithms for the iterative removal of blocking states. An important property of these synthesis algorithms is that they are guaranteed to yield a *minimally restrictive* supervisor, that is, a supervisor which only removes as much of the originally possible plant behavior as is needed to achieve compliance with the specification and non-blockingness, but not more [199, 147]. In other words, the resulting supervisor is guaranteed to include *all* possible behaviors as long as they comply with the specification and are non-blocking. For an example of such algorithms, the reader is referred to [199]. Modern software tools for supervisor synthesis, such as *Supremica* [135] employ more sophisticated algorithms which exploit the modularity of composed plant models to synthesize partial supervisors for subsystems, thereby mitigating state-space explosion problems [6, 147]. This enables analyses of complex systems up to a million states [135]. The following Example 2 (adapted from [93]) illustrates the principle of supervisor synthesis.

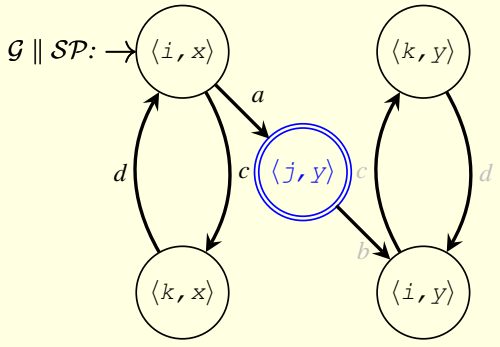
Example 2 - *Supervisor Synthesis.*
Consider the plant system $\mathcal{G}$ shown below with the events $\Sigma = \{a, b, c, d\}$. The behavior of $\mathcal{G}$ in terms of the possible event sequences is easy to see: Initially, the events a or c can occur. Event a must always be followed by event b, and event c must always be followed by event d. Each of the resulting sequences (ab) and (cd) may occur arbitrarily often, but at the end, there must always be one occurrence of event a to ensure that the system reaches a marked state. As a regular expression, this can be denoted as follows: $((ab)^*(cd)^*)^*a$.



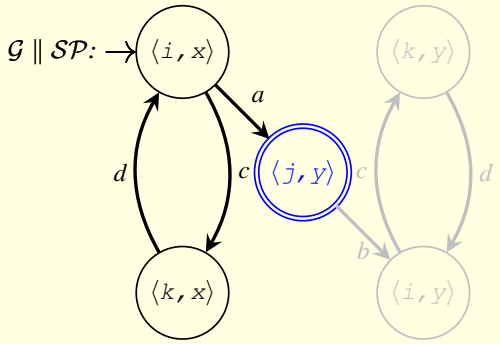
Assume that a restriction on the plant behavior shall be imposed. For instance, a possible restriction could be that the event a must happen exactly once during operation of the plant (but not more than once). This specification is expressed by the following automaton $\mathcal{SP}$. Observe that this automaton allows arbitrarily often repetitions of a, b, c, but requires the event a to happen exactly once in order to reach the marked state. As a regular expression, this is denoted by $(b^*c^*d^*)a(b^*c^*d^*)$:



In the first step of supervisor synthesis, the plant automaton is synchronized with the specification automaton, and the system $\mathcal{G} \parallel \mathcal{SP}$ is obtained. Note that in this system, the only marked state is (j, y) (highlighted in blue), since the synchronous composition requires that both original states of the synchronized automata are marked (compare Eq. (2.21)):



Observe that $\mathcal{G} \parallel \mathcal{SP}$ satisfies the specification, as it requires event a to occur exactly once in order to reach the marked state, but does not allow for further occurrences of event a after that. However, the synchronization has lead to the possibility of blocking behaviors. For instance, event b may occur after event a, leading to state (i, y). While this is allowed by the specification, it results in to a blocking behavior as no return into the marked state is possible from there on. Thus, the synthesis algorithm removes the blocking states (i, j) and (k, y). The following non-blocking system represents the remaining behaviors that are allowed by the supervisor:



3. Related Work

In the following, the current state-of-the-art with respect to risk assessment and hazard analysis methods for safety-critical systems is presented and discussed. As the previous discussions have highlighted, the high system complexity, the unpredictability of human behavior, and the vast range of possible system states and interaction scenarios make it difficult to analyze and validate the safety of HRC systems. Similar problems exist for other safety-critical systems outside the domain of HRC, such as aerospace systems and autonomous road vehicles. The development of methods for risk assessment, hazard analysis, and safety validation is an active field of research across all domains of safety-critical systems. In this chapter, an overview of existing methods is given. The overview focuses on methods that have been applied to HRC or other robotic systems, but it also includes potentially transferable approaches from other domains.

Publications Related to this Chapter
Parts of the work presented in this chapter have been published in:

[92] **Huck, Tom P.**, Münch, N., Hornung, L., Ledermann, C., & Wurll, C. (2021). Risk assessment tools for industrial human-robot collaboration: Novel approaches and practical needs. Safety Science, 141, 105288.

This chapter is largely based on a survey which was conducted in the context of this dissertation and is published in [92] (see green box above). The survey features a literature review and a survey among industrial practitioners. The literature review was conducted in three stages. First, a search was conducted on *Google Scholar, IEEEXplore* and *ScienceDirect*, using the following combinations of keywords:

$$\{\text{Human Robot}\}\ \text{AND}\ \left\{\begin{array}{c}\text{Collaboration}\\ \text{OR}\\ \text{Interaction}\end{array}\right\}\ \text{AND}\ \left\{\begin{array}{c}\text{Safety}\\ \text{OR}\\ \text{Risk}\\ \text{OR}\\ \text{Hazard}\end{array}\right\}\ \text{AND}\ \left\{\begin{array}{c}\text{Assessment}\\ \text{OR}\\ \text{Analysis}\end{array}\right\}$$

Nr.	Criterion
1	The publication proposes an approach that can be used to support at least one aspect of (HRC) risk /hazard analysis.
2	The publication is concerned with safety in the sense of avoiding physical harm due to accidents. Aspects like psychological safety, ergonomics, or security are not considered here.
3	The publication is concerned with safety on a system level, not on the level of individual components like end-effectors, sensors, etc.
4	The publication is concerned with risk assessment/hazard analysis as a design-time activity (in contrast to runtime activities which are not concerned with the safety of a robot system as a whole, but rather with handling specific situations that occur during operation).
5	The proposed approach is novel and goes beyond the procedures which are already established by current safety standards (Section 2.1).

Table 3.1.: Inclusion criteria for the literature review (adapted from [92]).

From this search, 183 potentially relevant publications were first selected on the basis of title and abstract. In a second stage, the publications were examined closer to determine their relevance to the subject. For inclusion of a work into the review, all of the criteria shown in Table 3.1 had to be met. After completion of the selection process, the results were iteratively refined by considering further references in publications that were found to be especially relevant. After completion of the initial selection phase, the search was repeated regularly using the above keywords in order to capture further relevant publications that may have appeared in the meantime. While the contents are focused on HRC, methods from other areas of application which are potentially transferable to a robotics context are also covered.

The results of the literature research showed a wide range of different approaches which vary significantly in terms of goal, scope, and methodology. In the following, the results are structured three categories for a better overview,: *Semi-formal* methods, *formal and rule-based methods*, and *testing-based methods*. The distinction between these categories is as follows:

- *Semi-formal methods* use a formalized system model (e.g., block diagrams or control structure diagrams) to analyze safety critical systems in terms of hazards and risks. This system model is used to guide the user through the hazard analysis. However, the analytical burden is still mainly placed on human reasoning (i.e., the analysis is not automated).

- *Formal- and rule-based methods:* Formal methods evaluate the safety of systems by evaluation of strictly formalized models. In many cases, this allows for automated verification and formal safety proofs. Rule-based expert systems reason about safety properties based on pre-defined rules which encode domain-specific expert knowledge.
- *Testing-based methods* identify hazards by exposing the system to various testing conditions and observing the system's response. This can be done either in the real world or in simulation. While the former is more realistic, its feasibility is limited by cost, time, and availability of physical prototypes. Thus, this Section is mainly concerned with simulation-based testing.

Since this book focuses on the identification of hazards, only approaches which are applicable or related to the hazard analysis phase of the risk assessment procedure are included. Approaches that are mainly related to other parts of the risk assessment procedure (e.g., collision models, methods for the estimation of injury severity) are not mentioned below. For more details, the reader is therefore referred directly to the publication (see green box above).

This Chapter is structured as follows: Section 3.1 discusses semi-formal methods, Section 3.2 discusses formal- and rule-based methods, and Section 3.3 discusses testing-based methods. The survey among industrial practitioners is presented in Section 3.4. Finally, Section 3.5 discusses the limitations of the related work and highlights research gaps.

3.1. Semi-Formal Hazard Analysis Methods

Semi-formal hazard analysis relies on *system models* for hazard identification. These models, however, are only loosely formalized. Therefore, they cannot be evaluated automatically. The hazard analysis is thus based on human reasoning. The models serve as a tool to structure the hazard analysis and to guide human analysts through the procedure.

3.1.1. Systems-Theoretic Process Analysis (STPA)

Leveson introduced the Systems-Theoretic Process Analysis (STPA) as a method to identify hazards in complex social, technical, or socio-technical systems [131]. STPA is based on the premise that the majority

of hazards do not emerge due to faults or failures on the level of individual system components, but are introduced into the system in the design phase in form of unintended functionality that may lead to unsafe states. This is usually a consequence of system developers being unable to comprehend all possible interactions between system components due to high system complexity. Developers are therefore unaware of unsafe states that may result from unintended interactions between system components.

STPA aims to counter this problem through the use of control structure diagrams. Control structure diagrams in STPA are hierarchical diagrams of system components where each component serves as a controller for one or multiple underlying components (note that the term *controller* in this context does not necessarily refer to a technical component, but to any system component that serves a control function, including human operators). Each controller provides control inputs to and receives feedback information from the underlying components. An example of a control structure diagram for a mobile robot system is shown in Figure 3.1 After modeling the control structure and specifying what control input and feedback are provided by each component, the user must identify potentially hazardous control actions (HCA). This is done by analyzing each control action and assessing whether the control action could lead to an unsafe state if executed inadequately. STPA assumes four basic types of inadequate control actions that a user should consider [131, p. 213]:

- A control action that should be provided is *not* provided, or the controlled component does not follow/execute the control action.
- A control action that should *not* be provided *is* provided.
- A control action is provided *too early* or *too late* (either in terms of time, or in terms of sequence)
- A control action is stopped *too soon* or applied *too long*.

If the user determines that any of these inadequate control actions can lead to an unsafe state, the related control action is called a *hazardous control action* (HCA). For each HCA, all components that are involved in the respective control loop are further analyzed to identify if and how they could cause the HCA.

Although early STPA applications were seen mainly in the field of critical infrastructure and defense systems, there are also examples of this technique being applied to robot systems: Mitka and Moroutsos derive safety

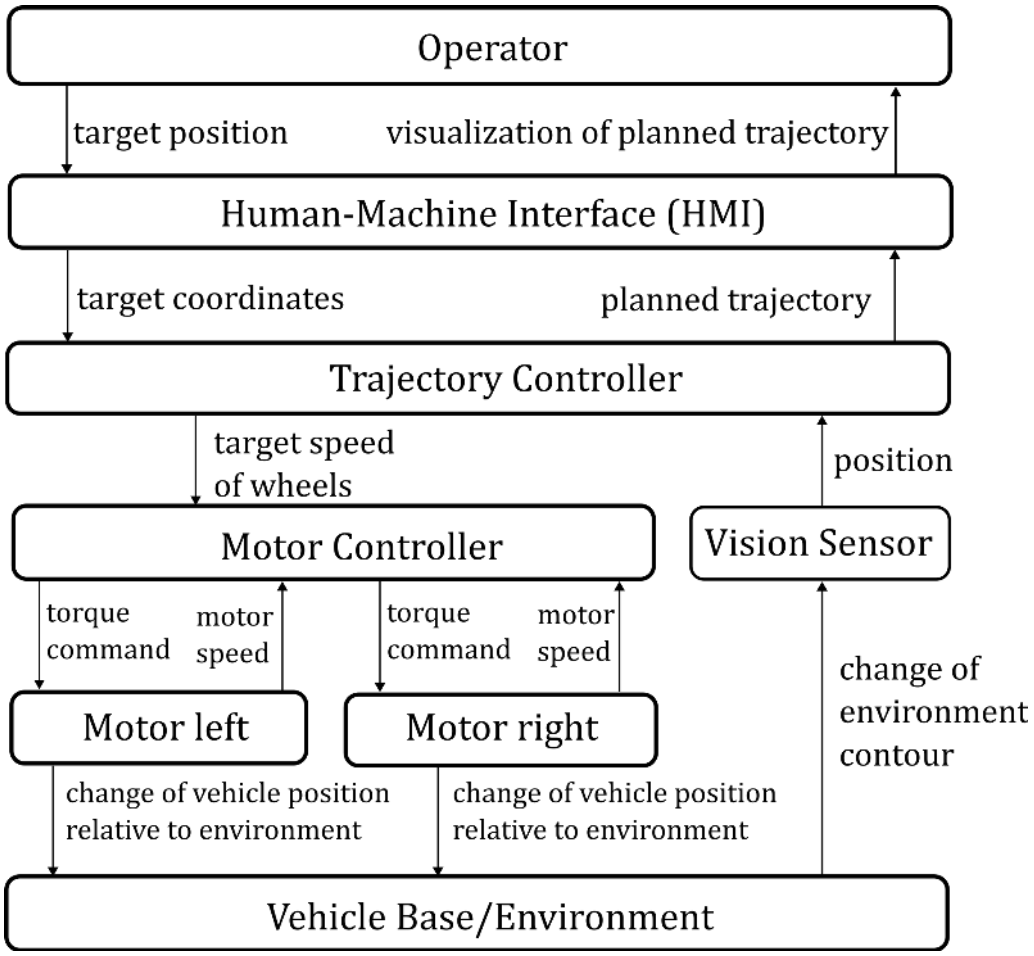


Figure 3.1.: Fictional example of an STPA control structure diagram for a mobile robot, inspired by [4]. Note the hierarchical structure: A controller may control multiple processes, and each process may consist of further controllers which control further sub-processes. (Author's own figure)

requirements for domestic robots on the basis if STPA hazard analyses [145]. Bensaci et al. apply STPA to identify hazards of a complex multi-robot system [29]. Adriaensen et al. present a case study of STPA being applied to a collaborative mobile robot [4]. Nuchalifah et al. present a case study on analyzing safety and security of a mobile disinfection robot using STPA-Safesec, an STPA extension which covers not only safety, but also security threats [151].

3.1.2. HAZOP and HAZOP-UML

Hazard and Operability Analysis (HAZOP) is a hazard analysis technique which was originally developed for the chemical process industry. Its procedures are defined in the IEC 61882 standard [97]. Similarly to STPA, HAZOP is based on a graphical system model. In the chemical process industry, this is usually a flow diagram. A set of guide words (e.g., *high pressure*, *low pressure*, *high temperature*, etc.) is then systematically applied to the different parts of the flow diagram to identify process deviations that could be potentially hazardous (note that this guide-word approach is similar to STPA's identification of HCAs).

Because the original HAZOP guide words are specific to the chemical process industry, it is difficult to apply the original HAZOP method to the field of robotics. For the application to robot systems, Martin-Guillerez et al. introduced HAZOP-UML [137]. Instead of the original flow diagrams, HAZOP-UML uses Unified Modeling Language (UML) diagrams to model robot use cases and an adapted set of guide words to identify hazards. Case studies of HAZOP-UML in a robotics context are presented by Guiochet et al. [77, 76].

3.1.3. Task-oriented Hazard Analysis

Apart from the two previous methods which are based on control structure diagrams and UML diagrams, there are also semi-formal methods which use a task model as a basis for hazard identification. Generally speaking, the approach of these task-oriented methods is to break down a human-robot interaction task, such as a collaborative assembly sequence, into different sub-tasks and assign potential hazards to these subtasks.

One such example is presented by Marvel et al. [139], who propose a task-based hazard analysis method that is based on an extension of Hierarchical Task Analysis (HTA). HTA is a long-standing method to describe and analyze manual assembly tasks [186]. Marvel et al. decompose collaborative workflows hierarchically into subtasks. Each subtask is assessed on the basis of a task ontology. The ontology describes the subtasks in terms of *subjects* and *predicates*. Subjects are the physical components that are involved (Robots, Humans, Tools, etc.). Predicates are properties and capabilities of task and subjects (e.g. what kind of collaboration

takes place, what properties the involved tools have, etc.). Potential hazards are assigned to each subtask based on the respective subject /predicate combinations. Although in some cases, automatic hazard identification is theoretically possible (e.g. if certain subject/predicate combinations are already known to cause specific hazards), the overall analysis is still mostly a manual procedure.

Antonelli and Stadnicka present the Process Failure Model and Effects Analysis technique for HRC systems (HRC-PFMEA) [12]. Like HAZOP, HRC-PFMEA is based on the already established hazard analysis technique FMEA [34] and has been adapted to HRC use cases. In its core, the PFMEA method consists of a set of pre-defined rules and criteria which help the user to identify and assess hazards that are related to collaborative workflows. The workflow is broken down into individual process steps. Each process step is then analyzed by the human users with respect to potential *failure modes*. This includes mistakes made by the robot, mistakes made by the human, and mistakes made due to inadequate interaction of both parties. If a failure mode is found to be applicable to a certain process step, then the user can apply a pre-defined set of rating criteria to assess if the failure mode is safety-critical, how critical it is, and what type of risk mitigation measures should be used.

Another similar approach is shown in a case study by Gopinath and Johansen [72]. They use a method known as Job Safety Analysis (JSA) and apply it to a collaborative assembly process. Similarly to HRC-PFMEA, JSA also breaks down a collaborative task into sub-procedures, and analyzes these with respect to potential hazards. Yet, the JSA procedure appears to be more generic and informal and less strictly defined than the HTA- and ontology-based approach of Marvel et al. [139]

3.2. Formal and Rule-based Methods

In contrast to the semi-formal methods presented above, formal methods rely on system models which are more strictly formalized. The strict formalization of these models allows it to provide formal safety proofs, in some cases even automatically. How-ever, human experts are still required to create and tailor the models, and to formulate specifications. In the following, two formal methods are discussed which have been applied to robotics, namely *model checking* and safety proofs with *differential dy-*

namic logic. Furthermore, *rule-based expert systems* are discussed. Although these do not provide a safety proof, they can also be considered formal methods as they perform reasoning on a pre-defined rule base of formalized expert knowledge.

3.2.1. Model Checking

Model checking is an analysis technique where the behavior of analyzed systems is captured in a formal model which is then verified against certain specifications such as safety constraints [43]. The most prominent model checking tool in the context of HRC is SAFER-HRC ("**S**afety **A**nalysis Through **F**ormal v**ER**ification in **H**uman-**R**obot **C**ollaboration"), which was proposed by Askarpour et al. [17]. SAFER-HRC uses on linear temporal logic (LTL) as a modeling language to describe HRC systems and safety specifications. LTL consists of logical operators and propositional variables. Logical operators include the operators known from boolean logic, as well as additional operators that describe time relations. In particular, SAFER-HRC uses the language TRIO, which is an extension of LTL that features a quantitative notion of time [67]. The system model expressed in TRIO is checked against safety specifications using bounded model checking to determine if the system conforms to certain safety specifications. The SAFER-HRC modeling approach is based on a composition of several submodels: one model describes the human operator (O), one the robot system (R), one the layout of the workcell (L). Each of these models describe safety-related properties of the respective elements as well as constraints for the configuration of each system (e.g., to avoid unrealistic human body configurations in the O-model). Additionally, task models (T) are defined to specify the interactions between the components in the O, R, and L models. These task models break down tasks into elementary actions, where each action is characterized by preconditions (i.e., conditions that must hold for the action to occur), postconditions (i.e., conditions that hold after the action has occurred), and safety constraints that must not be violated during the action. Additionally, the user can assign priorities to decide what actions should be performed first if preconditions of multiple actions are fulfilled at the same time. Furthermore, safety specification are formulated by the user in a formal language. Since the submodels are all represented by TRIO formulae, the system model can be checked automatically against the specifications using bounded satisfiability checking. In the case of SAFER-HRC, the satisfiability checker ZOT [161] is used.

ZOT analyzes if there exist any system behaviors which satisfy the constraints of the O-, R-, and L-models as well as the pre- and postconditions of the T model, but violate a safety constraint. If this is the case, then a hazardous situation has been identified and the unsafe system configuration is presented to the user as evidence. Following the iterative approach of the ISO 12100 risk assessment procedure (compare Section 2.1.3), users can iteratively add safety measures by modifying the model (e.g., by specifying that the robot must stop in certain situations) and then re-conduct the satisfiability check to identify whether the respective hazard has been eliminated [17].

Over the recent years, several extensions of the SAFER-HRC methodology have been developed [17, 18, 191], including a model of operator behavior and human error [19]. There have also been efforts to combine formal verification and simulation: Askarpour et al. combine SAFER-HRC with 3D simulation to visualize and analyse the identified hazards in more detail [20]. Rathmair et al. present a hybrid approach where formal verification is combined with 3D models to identify potential clamping hazards between the robot and static objects. They also consider system variability, namely variations of robot speed and trajectory, to avoid the need for repeating the hazard analysis in case of minor modifications to the robot movement [166]. Another combined approach consisting of formal verification, simulation, and real-world testing is presented by Webster et al. [194]. Outside the industrial domain, formal verification has also been applied to other fields such as medical and assistance robotics [193, 42, 130] or mobile outdoor robotics [162].

3.2.2. Safety Proofs with Differential Dynamic Logic

While the aforementioned model checking approaches generally rely on the enumeration and exploration of discrete state-spaces, there are also methods which rely on mathematical proof. These methods are especially useful for Cyber-physical systems (CPS), which are characterized by both continuous-time behaviors (such as movements or other physical processes) as well as discrete behaviors (such discrete control decisions by a computer). Robot systems are a typical example for CPS because they feature both discrete computational aspects and interaction with the physical world [159].

Due to the partly continuous nature of the state-space of CPS, exhaustive exploration of discrete states, as it is done in many model checkers, is not possible. Instead, safety properties need to be shown through mathematical proof. To that end, Differential Dynamic Logic (dL) has been developed. dL is a logic for reasoning about the behavior of CPS and proving properties of CPS. dL uses differential equations with domain constraints to describe the physical behavior of CPS, and discrete operators such as assignment, choice, and repetition, to describe the discrete behavior of CPS. Apart from first-order logic operators, dL also features quantifiers that express reachability properties for the continuous evolution [159].

In a hazard analysis context, dL can be used to prove (or disprove) safety properties of CPS. Although safety proofs for complex CPS are challenging, the compositional nature of dL formulae allows for a "divide and conquer" approach [159]. Furthermore, an appropriate choice of abstraction level can help to keep complexity manageable. Examples of dL safety proofs for robots are found for instance in [146, 118, 32]. A theorem prover for dL, named *KeYmaera X*, is available to assist with proof efforts [66].

3.2.3. Rule-based Expert Systems

Rule-based expert systems for hazard analysis encode domain-specific safety know-ledge in the form of pre-defined rules. This representation allows for automatic identification of potential hazards. Awad et al. [21] have developed a rule-based method for automated hazard analysis and risk assessment of HRC systems on the basis of the *PPR*-model [63], which describes collaborative robot workflows by three finite sets of properties: Product properties P, resource properties R, and process properties A. The union of all three sets is called the set D of workplace design properties: $D = P \cup R \cup A$. Further, there is a a finite set H which contains hazards that may occur at a HRC workplace. For an inference about potential hazards, pre-defined rules map the design characteristics $d_i \in D$ of a given workplace to the relevant workplace hazards $h_i \in H$:

$$\text{rule} : P \cup R \cup A \rightarrow H \tag{3.1}$$

Additionally, some rules are defined to assist users with additional tasks such as risk estimation or selection of risk mitigation measures.

Wigand et al. [197] present a similar knowledge-based hazard identification technique, which extends on the ideas of Awad et al. They also apply

a rule-based mapping of hazards, but do so on three different abstraction levels to capture different perspectives on the system. The three levels are: intra-component (e.g., hazards resulting from sharp edges on workpieces), inter-component (e.g. crushing hazards due to gaps between work-pieces), and workstation (e.g. hazards due to robot movement).

Rule-based systems can also be combined with simulation. The tools *CobotPlanner* [65] and *Dynarisk* [26], developed by the Fraunhofer institutes IFF and IWU, can be considered hybrids of rule-based expert systems and 3D simulation tools. They rely partially on pre-defined rules and formulae (e.g. for determining safety distances or safe robot velocities), and partially on 3D simulation models (e.g., for calculation and visualization of hazardous areas).

3.3. Testing-based Methods

The previously discussed approaches rely on *white-box* system models. White-box models allow users to analyze systems by explicitly reasoning about certain well-known features of the system's internal structure or functionality. In contrast, *black-box models* capture input-output relations of systems, but do not explicitly consider the system's internal structure [133]. The applicability of white-box methods is limited when the analyzed system is either highly complex, or contains black-box components like compiled third-party software or data-driven algorithms. In such cases, safety needs to be validated through testing. To that end, the system which is analyzed (referred to as system under test, SUT) is subjected to external stimuli in a testing environment and it is observed if the SUT responds to these stimuli in a manner which satisfies safety specifications. The testing environment may either be a real-world environment or a simulation. Since this work is primarily concerned with the early design phase where real-world systems are usually not available for testing, this section focuses on simulation-based approaches. A major research challenge in the context of simulation-based testing is the creation of appropriate test cases which need to be realistic, representative of the real-world conditions that the system will be subjected to, and critical enough to expose potential errors and design flaws in the SUT. Especially the latter is difficult to achieve, since it is usually not known a-priori what safety flaws a system contains or under which conditions it may fail. There are different approaches for generating

test cases and guiding the test case generation, which are briefly introduced below.

3.3.1. Agent-based Testing

Agent-based testing is a test method that focuses on the interaction of a system with its environment. In agent-based testing, a simulation model of the SUT is introduced into a simulation environment that contains autonomous agents with which the SUT has to interact. In this environment, the SUT is exposed to various different agent behaviors. This allows analysts to observe how the SUT interacts with these agents and responds to their behaviors [60].

Agent-based testing is thus particularly suitable for domains such as HRC, where safety depends on dynamic reaction and interaction. Araiza-Illan et al. propose an agent-based approach for testing HRC systems. In their approach, the SUT is introduced into a system environment where it interacts with human agents [16]. In order to create realistic testing conditions, the agents are modeled to have certain beliefs, desires, and intentions which are reflected in their behavior [16, 13]. Another agent-based approach is presented by Grzeskowiak et al. [74], who aim to test mobile robot navigation by exposing the robot to a simulated crowd of pedestrians. To simulate the crowd's motion, each pedestrian in the crowd is considered as an individual agent who aims to optimize their path with respect to a certain cost function, that depends on the pedestrian's current state as well as its environment (e.g., the presence of other agents or obstacles).

Chance et al. propose a similar agent-based testing approach, but in the domain of autonomous vehicles [40]. They present a multi-agent framework, where an autonomous vehicle needs to navigate safely in the vicinity of a group of pedestrians which are modeled as agent who make autonomous decisions (e.g., whether to cross the road in front of the vehicle at a given point in time).

3.3.2. Coverage-based Testing

High-fidelity simulations are computationally expensive and the underlying simulation models often have large state-spaces. Thus, simulation-based testing usually cannot guarantee complete coverage of all system

states which can possibly be reached in the simulation model. To assess the trustworthiness of simulation-based tests, it is therefore necessary to quantify the degree of test coverage that has been achieved with a particular set of test cases. Simply counting the number of tests is not sufficient for this, as the number of test cases alone does not reflect the quality or diversity of the testing scenarios. Instead, *coverage metrics* are used. Coverage metrics typically define a degree of coverage based on certain domain-specific criteria. Generally, coverage metrics are a relative quantity which measures the degree of coverage achieved by a set of tests:

$$\text{Coverage} = \frac{\text{Number of test cases covered}}{\text{Number of total test cases possible}}$$

However, what constitutes complete testing is often not clearly defined and sometimes, there are infinitely many possible test cases. Thus, coverage metrics are usually based on domain-specific criteria. In software testing, for instance, coverage measures quantify to what extent a given piece of program code is executed during testing. This can be determined based on factors such as the number of executed statements, functions, or branches in a program [148]. Coverage metrics can also be defined based on of the share of requirements that have been tested [196].

In the context of simulation-based safety testing, coverage metrics can be used as a quality criterion for the generation of test cases. Araiza-Illan et al. propose the use of code coverage criteria to guide the generation of test scenarios when testing robot control code [14]. Lesage and Alexander [129] present another coverage-guided testing approach for human-robot systems. Instead of code coverage, they propose the use of a domain-specific metric called *situation coverage*, which is based on the situations that have been covered by testing rather than execution metrics of the control code. In this context, situations are defined on the basis of certain human-robot configurations. A similar concept is presented by Hawkins and Alexander in the context of autonomous vehicles, where situation coverage is defined based on aspects such as the configuration of the road or the positions of parked vehicles positions in a traffic scenario [83].

3.3.3. Falsification

Apart from creating realistic testing scenarios and achieving a high degree of coverage, it is also important that testing scenarios are sufficiently

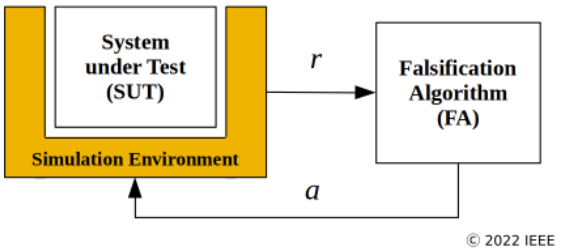


Figure 3.2.: Falsification principle [49].

critical to expose hazards in safety critical systems. This is especially crucial when hazards are "hidden", that is, when unsafe behaviors remain unobserved in most situations and only emerge under specific conditions. Falsification methods aim to expose conditions that lead to failure of the SUT [113]. In the context of safety testing, this essentially means creating adversarial test cases which are as critical as possible and thereby cause the SUT to fail, enter an unsafe states, encounter an accident, or otherwise behaves in an undesired way. The general principle of falsification is shown in Figure 3.2: The SUT is embedded in a simulation environment. The environment is controlled by a falsification algorithm (FA). The FA decides on some input a which affects the simulation environment. This input may consist, for example, of actions or events which are executed in the simulation, disturbances acting on the SUT, or parameter values which change the properties of the simulation environment. The adapted simulation is executed and the behavior of the SUT under these new conditions is monitored to detect any violation of safety constraints. After the simulation has terminated, a reward r is returned to the FA. The reward indicates the level of criticality of the simulation scenario. Such a rewards can be based, for instance, on the occurrence of unsafe states or some other domain-specific metric chosen by a user. One prominent example of this approach is *Adaptive Stress Testing* (AST) [125, 126, 117]. AST considers an SUT which is embedded in a simulation environment which subjects the SUT to stochastic disturbances[1]. The interactions between SUT and environment are formulated as an MDP. Algorithms such as Monte Carlo Tree Search [126] and Deep Reinforcement Learning [116] are deployed to find disturbances that lead to the most likely failure events. Adaptations

[1]This work uses a similar approach which is inspired by AST. However, in contrast to AST, this work does not consider stochastic disturbances, but instead behaviors of human agents to which the SUT is subjected. Furthermore, while AST aims to find the most likely failure scenario, this work focuses on finding the most critical agent behaviors, that is, behavios which maximize a domain-specific risk metric. Chapter 4 discusses this in detail.

of AST leverage dissimilarity rewards (i.e., rewards that encourage a diverse exploration of the search space) [46] and domain expert knowledge [58]. Apart from AST, there is also a large number of other falsification approaches, which use different algorithms to create testing scenarios, but follow in principle similar concepts. Most of these examples are found in the field of autonomous vehicle testing [22, 55, 114, 73, 115]. A comprehensive survey is presented in [48].

3.3.4. Testing in Virtual Reality

Testing in virtual reality is a compromise between real-world testing and simulation-based testing. Testing in virtual reality can be useful for analysis of HRC systems, as it allows safety engineers to experience simulation models or digital twins of a system in a realistic three-dimensional visualization [140, 170, 54]. Coupling virtual reality technology with human pose detection and digital human models also allows human-in-the loop simulations where real human operators can interact with a digital twin of the robot system. This enables safety engineers to spot potential collision scenarios without endangering real humans, and allows them obtain first insights regarding possible collision scenarios [143, 176]. Furthermore, virtual reality can also be used for safety training of operators [53].

Instead of performing a full human-in-the loop approach, it is also possible to import pre-recorded human motion data into simulators to create a testing environment. One such example is presented by Bobka et al. [31], who have developed the *Human Industrial Robot Interaction Tool* (HIRIT). This tool allows users to import recorded human motion data and simulate human motions in conjunction with collaborative robots. During the simulation, safety-related quantities such as robot speed and minimum human-robot distance are evaluated. This system is primarily intended to validate robot controllers that react to approaching humans (e.g., through velocity scaling on the basis of human-robot distance). The import of motion data allows a realistic simulation of human motion. A drawback, however, is that once recorded and imported, the motions cannot be altered. If one wants to change the behavior of the human in the simulation, e.g. for a closer investigation of certain safety-critical situations, new data must first be recorded.

3.4. Current Industrial Practice

In addition to the literature review, a series of expert interviews and an online survey were conducted to assess the current industrial practice of risk assessment and hazard analysis for HRC systems. In a first step, in-person interviews with 16 experts from the German automation industry were conducted. All participants had professional experience in conducting hazard analyses/risk assessments and implementing HRC applications. The interviews were conducted in the form of structured qualitative interviews and evaluated according to the qualitative content analysis methodology of Mayring [142]. In a second step, an online survey with further 28 participants was conducted to validate the findings of the initial interview series and content analysis. Below, the findings are presented in an abbreviated and aggregated manner. For a full overview of the survey, including details on questionnaire, content analysis, and quantitative evaluation, the reader is referred to [92].

The survey indicates that industrial practitioners largely follow the methods prescribed by the EU machinery directive and ISO 12100. Yet, the methods specified in these documents are rather general and not strictly formalized (Sec. 2.1.3). Regarding the tools used in hazard analysis, a majority of participants stated that they use office software (e.g., spreadsheets) as a tool for hazard analysis. Approximately half of the participants stated that they use commercial risk assessment software, such as *WEKA CE Manager* [195], *Safexpert* [94], *Docufy* [57], or *GESIMA* [5]. These tools, however, are mainly geared towards generating documentation and guiding the user through the risk assessment procedure. They do not provide any advanced features such as automated hazard identification or collision force estimation. Notably, none of the participants stated that they use any of the novel methods presented in this chapter. A majority of participants stated that checklists are useful for support of hazard identification and that risk assessments should be conducted by a group multiple people. Furthermore, the participants almost unanimously stated that experience and know-how are the most important factors for risk assessment.

When asked about desires for future tools and methods, a majority of participants stated that simulation-based collision analysis would be desirable. However, less than half of the participants expect that simulation has the potential to replace real-world collision measurements entirely. When asked about suggestions for future risk assessment tools, participants mentioned the use of CAD data for automated calculation of safety distances,

automated checking for critical geometries such as shearing points or sharp edges, and CAD-aided cell layout planning. Furthermore, suggestions included automated or partially automated hazard identification (e.g. by means of artificial intelligence), pre-defined risk-templates for the assessment of frequently occurring tasks, and the use of digital human models in HRC simulation (e.g., to assess collision possibilities with critical body parts such as the head).

While most participants seemed to be open to improvements and new features, awareness of the novel methods from scientific literature among the participants was limited, and those methods which are known to the experts are largely viewed skeptically. Figure 3.3 shows the participants' attitude towards some of the novel methods discussed above. It is noteworthy that apart from HAZOP-UML (which is likely well-known because its foundation, HAZOP, is already an established method), all of the methods are largely unknown to the participants. Among those participants which did have some knowledge of the respective methods, a significant share views them as "not promising".

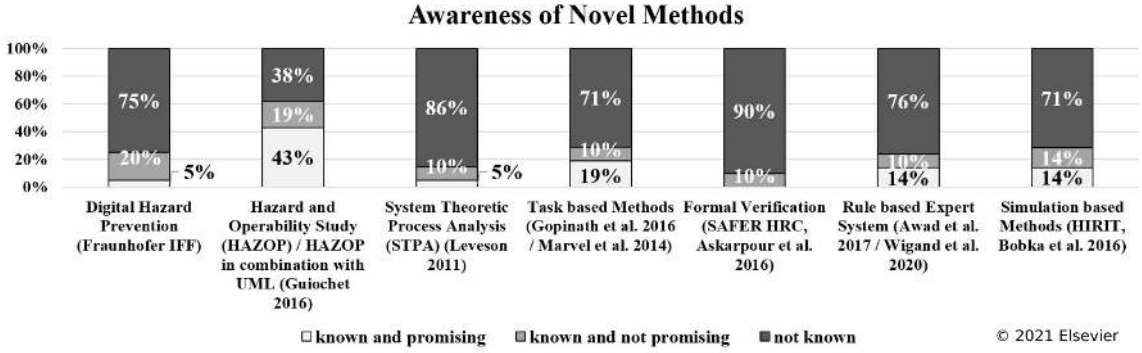


Figure 3.3.: Attitude towards novel methods amongst industrial practitioners [92].

Reprinted from: Risk assessment tools for industrial human-robot collaboration: Novel approaches and practical needs. Huck, Tom P., Münch, N., Hornung, L., Ledermann, C., & Wurll, C. (2021). Safety Science, 141, 105288. With permission from Elsevier. ©2021 Elsevier.

The survey indicates several important points: First, current industrial practice makes only very limited use of dedicated tools and methods beyond the general procedures prescribed in the respective standards. Tool support seems to be mainly limited to check-lists, commercial documentation tools, and simple office software such as spreadsheets. Second, tools and methods with novel capabilities are, at least in principle, seen as desirable by industrial practitioners. Third, despite novel tools being seen as generally desirable the methods from scientific literature are not yet

established in industry. This may be because they are largely unknown, but also because they are seen with some skepticism. Although the number of participants in this survey is too small to draw general conclusions, the aforementioned points are also corroborated by other works which report similar problems around risk assessment and hazard analysis methods [1, 68, 86, 82].

3.5. Discussion of Related Work and Limitations of Current Methods

This section discusses and compares the characteristics, advantages, and disadvantages of the methods presented in this Chapter.

As shown by the survey, the current industrial practice of hazard analysis relies heavily on human reasoning, intuition, and expert knowledge (Section 3.4). It only draws on limited support from systematic hazard analysis methods or software tools [92, 86]. While human reasoning remains an important pillar of hazard analyses, its efficacy is limited when it comes to the analysis complex dynamic systems. To identify potential hazards, one needs to consider a vast set of system states and interactions between sub-systems, which may easily overburden the analytical skills of humans (compare the challenges discussed in Section 2.1.6). Thus, additional methods and tools are needed to support current hazard analysis practices.

One of the most crucial aspects in hazard analyses is the achievable *completeness of identified hazards*, that is, whether the analysis has identified all hazards and/or potentially hazardous behaviors of a system. In the context of model-based hazard analyses, one needs to differentiate between (i) *completeness with respect to the real system* and (ii) *completeness with respect to the system model*. The former means that the analysis identifies all hazardous situations or behaviors that can appear during operation of the system in the real world whereas the latter means that the analysis identifies all hazardous situations or behaviors that can appear in the model (regardless whether that model accurately corresponds to the real-world system). Completeness with respect to the system model is primarily a matter of runtime and computational cost. Often, exhaustive exploration of all possible system behaviors is only feasible for simple or highly abstracted models. Completeness with respect to the real-world system re-

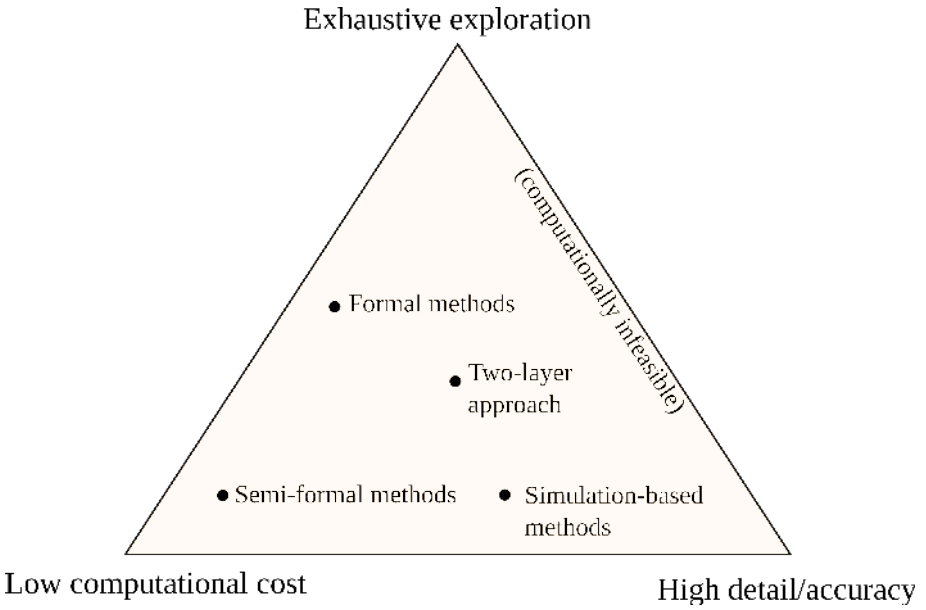


Figure 3.4.: Conflicting goals of hazard analysis methods. (Authors own figure)

quires not only that the model is searched exhaustively, but also that the model accurately represents the real-world system. These can be opposing goals, since the former requires that the model is simple enough to facilitate exhaustive search without exploding computational cost whereas the latter requires that the model is detailed enough to represent the real-world system accurately. Therefore, it is difficult to achieve both exhaustive exploration and a high level of detail given a limited computational budget. In other words, a higher level of detail requires a more detailed model of the analyzed system, which in turn increases computational cost and makes it more difficult to achieve exhaustive exploration. Conversely, given limited computational resources, the desire to exhaustively explore all possible model states places restrictions on the modeling detail due to state-space explosion problems. These conflicting goals can be imagined as a "magic triangle" as shown in Figure 3.4. This triangle shall be used in the following to illustrate the differences among the state-of-the-art approaches.

Semi-formal methods guide users through the procedure based on system models (e.g., control structure diagrams or UML diagrams). Compared to methods that are based on intuition and experience, this approach provides a more rigorous and structured hazard analysis procedure (Section 3.1) which helps to decrease the likelihood of hazards being overlooked or forgotten. However, semi-formal methods only have limited potential for automation, since the core of the procedure still depends on human rea-

soning. This makes semi-formal methods vulnerable to the same problems as described in the previous paragraph, albeit to a lesser extent. In Figure 3.4, semi-formal methods are found towards the lower left corner of the triangle, as they are generally associated with low computational cost (as most of the analytical work is done by human reasoning), but also with low detail and limited exhaustiveness.

Formal verification techniques based on model checking, such as SAFER-HRC, are capable of automated hazard analysis without relying on human users to identify hazards (Section 3.2). Given sufficient computational power, these methods can analyze complex systems with large state spaces that would be infeasible to analyze by human reasoning alone. However, the formalization of the hazard analysis procedure introduces new challenges: First, the user must translate system and safety specification into a formal model, which requires time and training and is error-prone. Second, to keep modeling complexity manageable and avoid state-space explosion problems, formal modeling frequently requires significant modeling abstractions, such as discretizations of continuous state variables. This is illustrated in Figure 3.5, which shows a model of a HRC system used in a formal verification case study. Note that the collaborative workspace is abstracted through a discretization into several large two-dimensional areas. Abstractions like these limit the achievable level of detail compared to more fine-grained approaches such as simulation[2]. Formal safety proofs by means of differential dynamic logic can ensure that certain safety properties hold at all times given certain initial conditions, even for mixed discrete-continuous hybrid systems that cannot be verified by discrete model checking. However, finding proofs is challenging and requires experts who are trained in the proof techniques. Furthermore, such proof techniques cannot be applied to systems which are composed of black box components. Furthermore, even if a system *design* is proven to be correct w.r.t. certain safety specifications, there is no guarantee that there are no errors such as software bugs in the system's *implementation*. Thus, although formal proofs are highly valuable, they do not alleviate the need for further assurance measures. Formal methods are generally associated with a relatively high degree of exhaustiveness, but also with considerable computational cost and limited level of detail, placing them somewhere between the top and left corner of the triangle in Figure 3.4. A

[2]It should be noted that this particular figure is only intended as an example. The discretization does not necessarily have to be as coarse-grained as depicted.

combination of both high detail and exhaustiveness is often computationally infeasible, as indicated on the right edge of the figure.

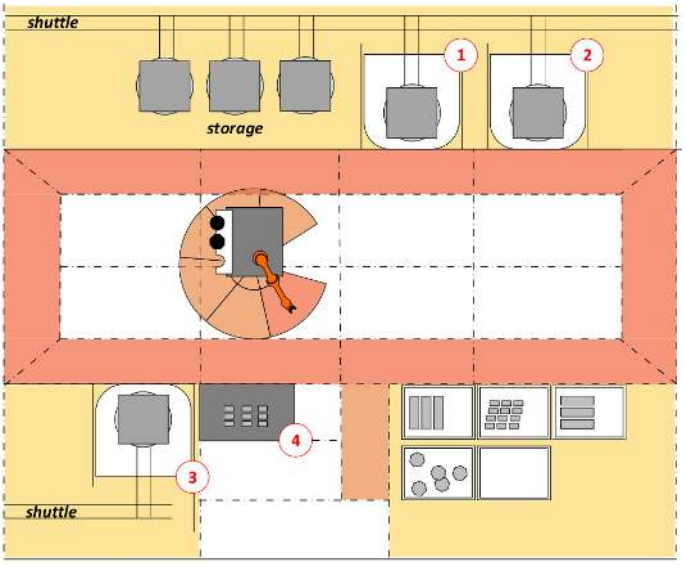


Figure 3.5.: Example of modeling abstractions: A discretized human-robot collaborative workspace from a SAFER-HRC case study [191].

The approach proposed in this work uses simulation-based testing as a tool for hazard identification. This promises several advantages:

- *Availability*: In contrast to formal languages or rule-based description formats, 3D-simulation is already widely used in the robotics and automation industry [42]. Numerous commercial and non-commercial simulators are available [52]. Since 3D simulation is already being used for other tasks such as layout planning, cycle time optimization, or virtual commissioning, a simulation model is often already available in the development process and can be used as a basis for hazard analysis.
- *Detail*: Although all models require a certain amount of abstraction, a detailed 3D simulation requires much less modeling abstractions than a formal verification approach. The high accuracy of state-of-the-art 3D simulators allows users to model spatial and dynamic properties in a detailed manner (e.g. movements, collision geometries, collision forces, etc.) which is of particular importance in collaborative robot applications.
- *Scalability:* Most existing hazard analysis methods are so-called *white-box* methods that require in-depth reasoning about the sys-

tem's internal structure and function. For instance, analyzing a UML description or building a formal model requires an in-depth understanding of the analyzed system. In principle, this is not a disadvantage, since it is certainly desirable that users try to understand the analyzed system as much as possible. However, white-box approaches do not scale well with increasing system complexity, because maintaining an in-depth understanding about the system's internal structure and function becomes increasingly difficult as systems become more complex. Simulation-based methods are more scalable with respect to system complexity because they enable the user to adopt a *black-box* approach. The simulation model is considered a black box which is exposed to different testing conditions (i.e., simulation inputs) and the resulting system behaviors (i.e., simulation outputs) are checked for violations of safety criteria [48]. Since this approach relies on input-output relations and does not require an analysis of the internals of the simulation model, it is suitable for complex systems which cannot be analyzed with a white-box approach [126].

While there are already examples for simulation-based safety analysis techniques in literature (Section 3.3), most of these techniques are being applied in other domains, especially in the context of autonomous vehicles [47, 22, 55, 114, 73, 115]. In the domain of collaborative robotics, there are still relatively few works which have considered simulation-based hazard analysis, and those works have focused mainly on the testing and validation of software-related aspects (e.g., code validation [15, 16] or testing of control strategies [31]). For a comprehensive support of hazard analysis, however, a more extensive approach is needed, which takes into account system-level behavior, interactions between system and system environment, and physical aspects of HRC, such movements, geometries, and collision forces. Such approaches will be introduced in the following chapters. More specifically, the remainder of this book will propose simulation-based techniques as well as a two-layer approach which combines simulation with formal methods. While the simulation-based techniques generally trade exhaustiveness for a higher level of detail (placing them on the lower right of Figure 3.4), the two-layer approach attempts to strike a balance between both aspects (as indicated by the central position in Figure 3.4).

4. A Simulation- and Agent-based Hazard Analysis Approach

A NOVEL APPROACH FOR HAZARD ANALYSIS of collaborative human robot systems is proposed in the following. The approach is based on simulation and combines prior work in the fields of agent-based testing (especially the approach of Araiza-Illan et al. [13]), falsification (especially AST [126]), and formal verification. Section 4.1 states a formal problem definition for the hazard analysis problem, and Section 4.2 discusses solutions for the stated problem.

4.1. Problem Definition

In this Section, a formal problem definition is presented. The general concept is to formulate the hazard analysis as a *search problem* which can be interpreted as an MDP solved by heuristic search algorithms. The problem definition and the general approach are inspired by AST [126] (see also Section 3.3.3).

4.1.1. Simulation Model and Safety Specification

It is assumed that the analyzed system is represented by a simulation model $\mathcal{M}$. The set of all possible states of the simulation model $\mathcal{M}$ shall be called state space S. Let s_0 denote the initial state of the system and let S_{reach} denote the set of all states that are reachable from s_0. Furthermore, it is assumed that the safety constraints to which the system must adhere are given by a safety specification[1] spec:

[1]The term "specification" is also common in systems and sofware engineering, where a specification is typically a textual document that states requirements for a system. Here, however, the term is used

Definition 4 - *Safety Specification*
The safety specification spec *is a mapping from the simulation state space* S *to a boolean value that indicates whether a given state is safe or unsafe:*

$$\mathrm{spec} : S \rightarrow \{\mathrm{true}, \mathrm{false}\} \tag{4.1}$$

Where true *corresponds to a safe state and* false *to an unsafe state. The safety specifications may encode any user-defined criterion that is decidable on the basis of the information given by the simulator state space. The safety specification can consist of multiple sub-specifications which are concatenated by logical conjunctions or disjunctions.*

From a practical perspective, it should be noted that the simulation state s may not always be fully observable (e.g., because the programming interface of the simulation software does not allow full access to the simulator's internal state variables, because the system contains black-box components, or because the simulation state is too complex and handling the full vector of state variables is impractical). However, state-of-the art 3D simulators typically allow the user to retrieve numerous safety-related variables, such as speeds or distances, which are calculated from the internal state. Thus, strictly speaking, spec is not a mapping from the actual simulator state, but rather from set of state observations. For the sake of simplicity, however, no explicit distinction between the actual state the observation will be made in the following. It is assumed that the observation provides sufficient information to decide whether a state is safe or unsafe. The following example illustrates what the specifications may look like:

Example 3 - *Safety Specifications*
Consider the two following safety specifications which are given in textual form:

(A) "Human and robot must never have contact while the robot is moving."

(B) "Human and robot must never collide with a collision force that is greater than the respective body-region specific limit."

Suppose that the simulation allows the user to retrieve the following information about the current simulation state s *in every simulation timestep:*

- $d_{HR,i}(s)$*: The distance between robot and the* $i-$*th human body part (index* i *represents the respective body part).*

differently: The safety requirements are stated mathematically, through a function that descr a set of states that need to be avoided.

- $v_R(s)$*: The robot's current cartesian speed (for articulated robots, consider the speed of the fastest robot joint).*
- $F_{c,i}(s)$*: A body-part specific estimation of the force exerted on the human body in case of a human-robot contact.*

Further, let $F_{max,i}$ *denote a body-region specific collision force limit. Now, the specifications above can be expressed as follows:*

$$\textit{(A)} \quad \mathrm{spec}_A(s) = (v_R(s) == 0) \vee \bigwedge_i (d_{HR,i}(s) > 0) \tag{4.2}$$

$$\textit{(B)} \quad \mathrm{spec}_B(s) = \bigwedge_i ((d_{HR,i}(s) > 0) \vee (F_{c,i}(s) > F_{max,i})) \tag{4.3}$$

Note that Eq. (4.2) specifies that in a given state s *the robot must either be stopped (first part of the logical disjunction) or there must be no human robot contact, that is, all body parts must have a distance greater than zero from the robot (second part of the logical disjunction).*
Eq. (4.3) specifies that for all body regions, the respective body region must either have a distance from the robot greater than zero (i.e., not be in contact) (first part of the logical disjunction) or that the exerted contact force on the respective body part must be below the body-part specific threshold (second part of the logical disjunction).

For a given safety specification, the set of unsafe states U is defined as the set of all states where $\mathrm{spec}(s)$ evaluates to false:

Definition 5 - ***Set of unsafe states***
The set of unsafe states U *is the set of all states* $s \in S$ *for which* $\mathrm{spec}(s)(s)$ *evaluates to* false*:*

$$U = \{s \mid \mathrm{spec}(s) = \mathrm{false}\} \tag{4.4}$$

Note that the set U is only defined *implicitly* via the definition of the safety specification: for a given state s, it can be decided if $s \in U$ by evaluating $\mathrm{spec}(s)$, but the set of unsafe states U is not known at the start of the hazard analysis, since it is generally unknown a-priori whether a violation of the safety specification is possible in the given system, and, if yes, in what specific states these violations occur.

4.1.2. Agent-based Simulation

In principle, the goal of the hazard analysis is to determine if the system can reach any unsafe state, that is, to determine if $S_{reach} \cap U \neq \emptyset$. In practice, however, obtaining the set of reachable states S_{reach} is usually not tractable for the type of system models considered here. While the calculation of all reachable states might be feasible for certain types of systems (e.g., certain classes of hybrid and continuous systems whose dynamics can be described analytically [10]), it is not tractable for the complex 3D-simulation models that are needed to simulate HRC systems, as these are generally not based on an analytical description of system dynamics, but rather on numerical simulation.

Since an exhaustive computation is too costly, this work relies on a sampling-based approach to discover unsafe states: rather than attempting to calculate the full set of reachable states, the state-space is sampled by executing multiple simulation runs (see Figure 4.1). Each simulation run generates a trajectory through the system's state space. Along this trajectory, every state s_i is checked for a violation of the safety specification by evaluating $\mathrm{spec}(s_i)$. The goal of this approach is to find simulation runs along which the system enters an unsafe state. If such a simulation run is found, it serves as a counterexample showing that the system violates the safety specification. Furthermore, visualizing and re-playing the simulation run can help the user to identify and fix the underlying safety flaws in the system.

Adopting this approach raises the question of how to generate meaningful simulation scenarios. This work addresses the issue by adopting the agent-based approach proposed by Araiza-Illan et al. [13]. In agent-based testing, it is assumed that one or multiple virtual agents interact with a simulation model of the system which is being analyzed (Section 3.3.1). These agents interact with the system model and thereby create varying simulation conditions that cause the system to respond in different ways. In other words, trajectories through the state space are generated by sampling agent behaviors and passing them to the simulator, where they ultimately result in different state-space trajectories.

To enable this agent-based approach, the simulation model $\mathcal{M}$ is partitioned into two interacting sub-models. One submodel represents the analyzed robot system and is henceforth referred to as *System under Test* (SUT). Note that the SUT includes not only the robot itself, but also its

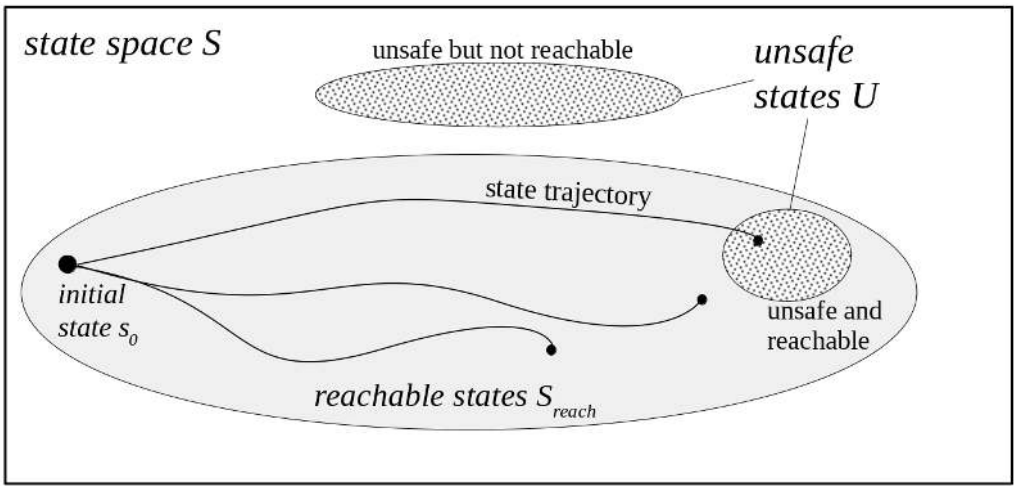


Figure 4.1.: The state space of the analyzed system contains two subsets: Reachable states and unsafe states. The goal of the hazard analysis is to states which are reachable and unsafe. Since computing the whole of S_{reach} and U is computationally expensive, the state-space is sampled by creating individual simulation runs and checking for unsafe states along the resulting state trajectories. (Author's own figure.)

periphery (e.g., sensors, tools, protective fences, etc.). The other subsystem represents the testing agent. As this work is mainly concerned with human-robot collaboration, the testing agents in the following examples will be virtual models of humans that are collaborating with the robot system. However, it should be emphasized that the agent does not necessarily have to take the form of a human model. In other use cases, the agent may also represent other entities that are relevant in that specific case. In a mobile robot navigation task, for instance, the agents may take the form of dynamic obstacles, while in a multi-robot system, the agents may represent other robots.

The behavior of the SUT depends on the programming of the SUT itself and on the behavior of the testing agent. It is assumed that the SUT responds deterministically to a given agent behavior. Since the SUT's behavior depends on the agent's behavior, the agent behavior can be considered as an input to the simulation model $\mathcal{M}$, whereas the behavior of the SUT is an emergent behavior which arises within the simulation as a consequence of the agent behavior and the programming of the SUT.

Generally, the agent's behavior may feature discrete aspects (e.g. the execution of discrete worksteps in a collaborative assembly task), contin-

uous aspects (e.g. motion parameters such as velocities), or combinations of both. How to encode agent behaviors will be discussed in more detail in Chapter 6. For now, it is assumed that there is a given set if possible behaviors denoted by B (how to model and describe B will be discussed in more detail in Chapter 6). This set forms the input space of the simulation. Because the behavior of the SUT is a consequence of the given agent behavior, the simulation model can be regarded as a function which maps the set of agent behaviors B to a power set of the state-space. In other words, the simulation model receives an given agent behavior $b \in B$ as input and returns a corresponding trajectory through the state-space of the simulation model.

Definition 6 ***Agent-based Simulation Model***
Given a set of agent behaviors B and a state-space S, an agent-based simulation model is a mapping

$$\mathcal{M} : B \times S \rightarrow S^k \quad k \in \mathbb{N} \tag{4.5}$$

That is, for a given initial state $s_0 \in S$ and a given agent behavior $b \in B$, the simulation model returns a resulting state sequence of finite length k which represents a trajectory through the model's state-space:

$$\mathcal{M}(s_0, b) = (s_i) = (s_0, s_1, s_2, ..., s_k), \quad s_i \in S \tag{4.6}$$

4.1.3. Hazard Analysis as a Search Problem

Given the definition of unsafe states and the definition of agent-based simulation, the hazard analysis problem can now be defined formally. A system is considered unsafe if at least one unsafe state is reachable from the initial state, that is, if there exists at least one agent behavior which, when performed in interaction with the SUT, causes the system to enter an unsafe state. Such a behavior shall be called an *unsafe behavior*. The question of determining whether the SUT interacts safely with the agent therefore comes down to determining if there exists an unsafe behavior. Thus, as shown in Definition 7, the problem of hazard analysis can be framed as a search problem (see also [126]). This principle is illustrated in Figure 4.2.

The behavior of the agent is given by a sequence of discrete actions a_i. The goal is to find an agent behavior (i.e., an action sequence[2]) which transfers the system from the initial state s_0 to an unsafe state $s_U \in U$.

Definition 7 ***Hazard Analysis as a Search Problem***

The hazard analysis problem shall be described by the following 5-tuple:

$$\langle \mathcal{M}, s_0, S, B, \text{spec} \rangle \tag{4.7}$$

Where $\mathcal{M}$ is a a simulation model, s_0 the initial state of the model, B the set of possible agent behaviors, and spec *the safety specification. The goal of the search is to find at least one unsafe behavior, that is, a behavior $b_u \in B$ for which the simulation model enters an unsafe state s_u along the resulting state trajectory:*

$$(s_i) = \mathcal{M}(s_0, b_u) \quad s.t. \quad \exists\, s_u \in (s_i) : \text{spec}(s_u) = \text{false} \tag{4.8}$$

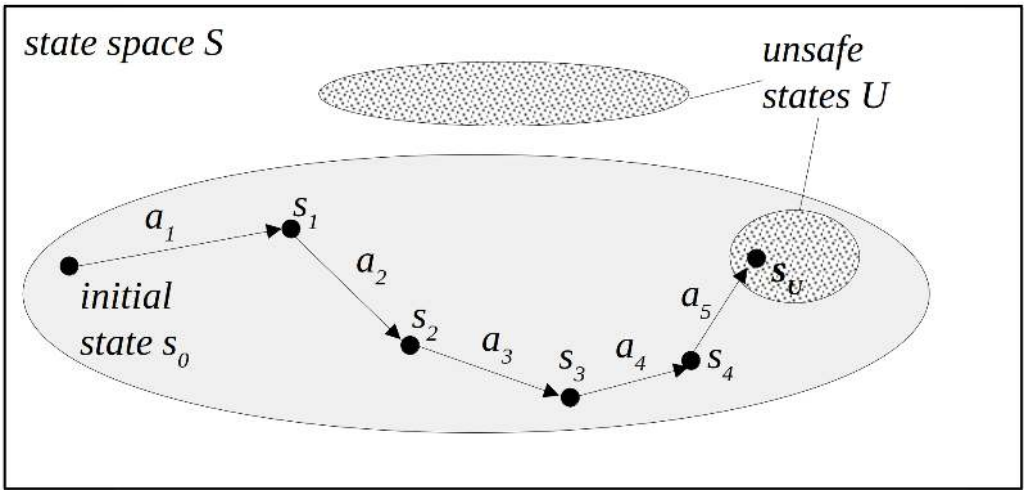


Figure 4.2.: A state trajectory $(s_0, s_1, s_2, ..., s_k)$ is created by sampling an agent behavior (represented as an action sequence (a_i)) and executing it in the joint simulation model of agent and SUT. The goal is to find a behavior which results in an unsafe state. (Author's own figure.)

[2]Note that in this simplified schematic, the length of the state trajectory k is equal to the length of the action sequence n. In practice, this is not necessarily the case. Typically it will be $k > n$, since an action of the agent usually extends over multiple simulation timesteps, especially for detailed models with a high resolution in the time domain

Note that by framing the problem in this way, the proposed approach combines aspects of agent-based testing with falsification: the agent's goal is to find action sequences that lead to unsafe states, that is, to *falsify* system safety. The agent can thus be interpreted as an *adversarial agent* who attempts to act in an unsafe manner in order to expose hazards. For now, this approach may seem rather abstract. The following Example 4 uses a simple "grid world" system to illustrate the concept.

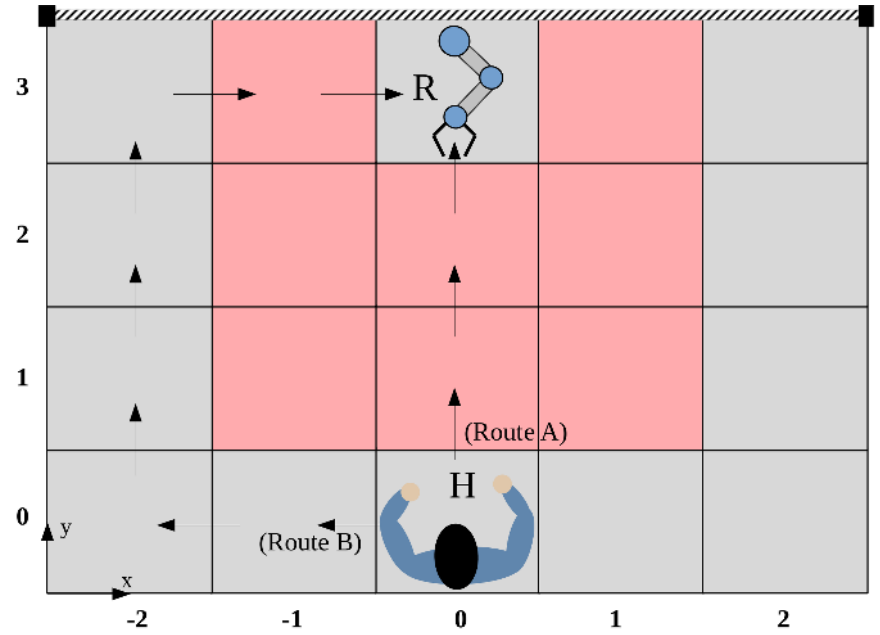


Figure 4.3.: Gridworld system from Example 4. (Author's own figure.)

The example is, of course, strongly simplified. From a practical standpoint, it is not necessary to frame the hazard analysis as a search problem in this simple example, because a human analyst will be able to quickly spot the safety flaw. However, in more realistic and complicated cases, spotting safety flaws through human reasoning is much more challenging. Suppose, for instance, that the system in Example 4 is not a simple grid world, but a highly detailed simulation model of a real robot system. There will be many additional factors to consider. For instance, the robot's stopping time may depend on its current velocity and configuration in a nonlinear manner. Further, the human may be able to reach over or around the monitored safety zone, or there may be bugs in the robot program due to which the robot will not stop as it is supposed to. With increasing system complexity, there will come a point where explicitly reasoning about such safety properties becomes impractical and a search-based approach is the more practical solution. Furthermore, the SUT might contain black-box compo-

nents, whose behavior is not fully known at the start of the analysis (e.g., a piece of compiled software whose source code is unavailable).

Example 4 *Consider the grid world system depicted in Fig. 4.3. The system consists of two subsystems. The SUT is the robot and the testing agent is the human. The system state s is described by three state variables:*

$$s = (x, y, s_R)$$

where x, y denote the agent's position in the grid world and s_R denotes the state of the robot. The robot state can be either "working" or "stopped". The system's initial state is

$$s_0 = (0, 0, working)$$

i.e., the agent is at position (0,0) and the robot is working. The robot is stationary, but the agent can move in the grid world by taking steps in positive and negative x- and y-direction, respectively. The set of events is therefore

$$A = \{+x, +y, -x, -y\}$$

and the set of agent behaviors is given by the set of possible action sequences

$$B = A^n$$

where n is the action sequence length. Suppose that a safety specification requires that there should be no human-robot collisions, that is, the agent must not be able to reach the robot while the robot is working. One can immediately see that the set of unsafe states U (i.e., the set of states that violate this specification) consists of only one state, namely:

$$U = \{(0, 3, working)\}$$

To avoid collisions, the robot monitors its surrounding. The monitored area is highlighted red in Figure 4.3. As soon as a human enters the monitored area, a safety stop is triggered. Due to several factors like sensor delay and required braking time of the robot, there is a delay in the stopping time. Assume that the stopping time takes as long as it takes a human to make two steps. With the previously proposed approach, the safety of this system is assessed by searching for action sequences that lead to U. Consider the two exemplary paths in Figure 4.3. The event sequence

$$(+y, +y, +y)$$

(corresponding to route A in the figure) results in the state $(0, 3, stopped) \notin U$, which is not unsafe (note that the robot stops in time because the human needs to take two steps through the monitored area which gives the robot sufficient time to stop). However, by taking the event sequence

$$(-x, -x, +y, +y, +y, +x, +x)$$

(corresponding to route B), the agent only steps through the monitored area once, which doesn't give the robot sufficient stopping time and thus results in the unsafe state:

$$(0, 3, working) \in U$$

It is thus shown that the system is unsafe.

In cases like these, a search-based approach is the only viable option, as prior knowledge about the system's behavior is not sufficient to make reliable safety claims. In addition, in contrast to Example 4, the set of unsafe states U is usually not known explicitly. Even if the safety specification is clearly defined, the concrete ways in which the SUT might fail to conform to that specification are often not known a-priori. Thus, at the beginning of the analysis, it is not known what specific system states belong to U, or even if there are any unsafe states. The necessity to first characterize U by discovering examples of unsafe states is a further reason to use a search-based approach.

4.2. Proposed Solutions

Now, that the hazard analysis has been formalized as a search problem, the question arises what methods should be used to solve the search problem. The main challenge in this context is search space complexity: In complex HRC systems, the search space of possible agent behaviors can become so complex that an exhaustive simulation of all possible agent-SUT-interactions is computationally infeasible. Therefore, strategies are needed to increase search efficiency and limit search space complexity. This work proposes solutions to address this issue. A brief summary of these solutions is given below. Over the course of the following three Chapters, the solutions are discussed in more detail. Since the Chapters build on each other, they should be read in successive order.

4.2.1. Risk-Guided Search

A naïve approach for creating agent behaviors would be to sample actions of the agent randomly, execute them in the simulation, and evaluate the safety specification in every simulation timestep to monitor the simulation state for violations of the specification. However, this naïve approach does not take advantage of the fact that the simulation state not only allows for binary statements about whether a state is safe or unsafe, but also allows further inferences about the degree of risk associated with a particular state. For instance, consider a safety specification which specifies that all human-robot collisions shall be considered unsafe states. Then, a simulation state entailing a near-collision is not indicated as an unsafe state

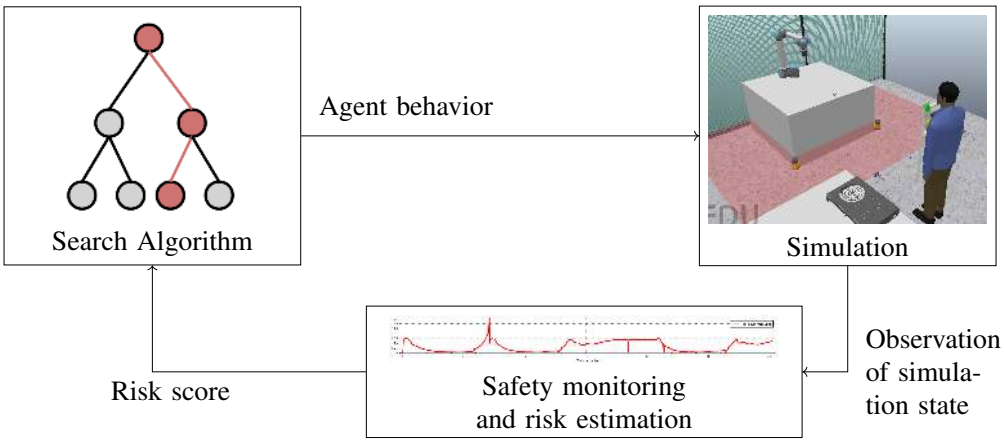


Figure 4.4.: Schematic of the risk-guided search approach. (Author's own figure.)

according to the safety specification. Yet, it is still associated with a much higher risk than a state where the robot is standing still or the human-robot distance is large. Leveraging such risk information in the search for unsafe states has the potential to increase search efficiency and result in a more reliable discovery of unsafe states. To that end, this book proposes the concept of *risk-guided search*. The risk guided search deploys search algorithms which are guided by a domain-specific risk metric. Based on this risk metric, the algorithms adapt their search strategy in order to maximize the risk metric. This leads to the creation of high-risk behaviors which result in unsafe states and thereby expose hazards in the analyzed system. The principle is shown in Figure 4.4 as an interplay of the functional components (i) *search algorithm*, (ii) *simulation* and (iii) *safety monitoring and risk estimation*:

- *Search algorithm*: The search algorithm iteratively samples agent behaviors for execution in the simulator. The algorithm receives feedback from the simulation in form of a domain-specific risk metric which is calculated by the safety monitoring and risk estimation component.

- *Simulation*: The simulation model consists of two interacting submodels: a model of the agent and a model of the SUT. The simulation receives the agent behaviors sampled by the search algorithm. The behaviors are then executed by the agent model in interaction with the SUT, which causes the SUT to be subjected to different interaction scenarios.

- *Safety monitoring and risk estimation*: During each simulation run, a risk score is calculated using a domain-specific risk metric. Afterwards, the risk score is fed back into the search algorithm. The risk metric acts as a heuristic to guide the search: by attempting to maximize the risk metric, the search algorithm creates high-risk agent behaviors which lead to unsafe states and thereby expose hazards.

4.2.2. Automata-constrained Risk-guided Search

Automata-constrained risk guided search follows the principle introduced above, but adds formalisms to impose constraints on the agent's behavior. To that end, the agent is abstracted as a discrete event system which is modeled by an extended finite automaton (EFA, see Figure 4.5). The notion of automata acceptance is used to discriminate between feasible and infeasible agent behaviors. Consequently, the set of feasible behaviors is obtained by computing the accepted language of the EFA. Automata-constrained risk-guided search ensures that the search algorithm only samples behaviors from this feasible set. This gives users the possibility to exclude certain behaviors which are considered infeasible or unrealistic. Thereby, the user can limit the search space to a subset of behaviors which is considered particularly relevant, while excluding behaviors that are considered unrealistic.

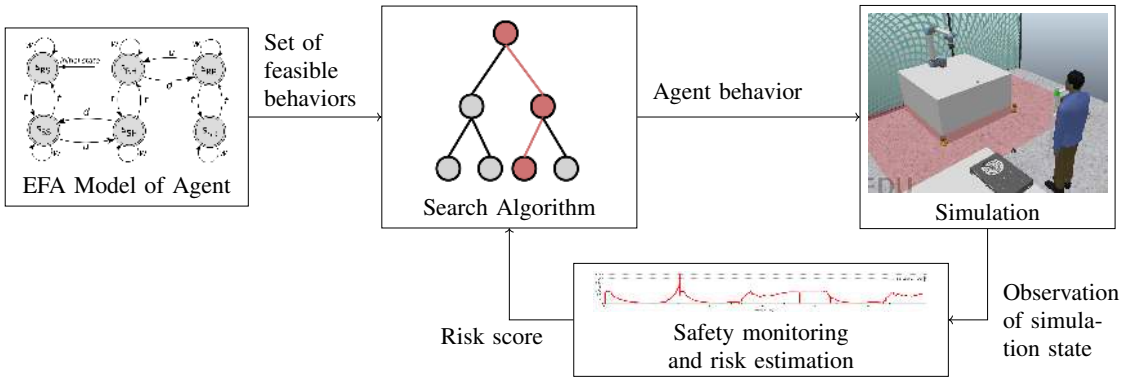


Figure 4.5.: Schematic of the automata-constrained risk-guided search approach. (Author's own figure.)

4.2.3. Two-level Hazard Analysis

The two-level hazard analysis approach is introduced towards the end of this book as an alternative to address some limitations of the risk-guided

search. It analyses the SUT on two different abstraction levels and thus combines aspects of formal verification and simulation (Figure 4.6). On the higher (i.e., more abstract) level, the agent, SUT, and safety specification are modeled as EFA. A set of potentially unsafe behaviors is synchronized from these EFA models using formal synthesis algorithms. Thereby, potentially hazardous behaviors are identified a-priori, that is, before any simulations are conducted. On the lower (i.e., more detailed) abstraction level, these potentially hazardous behaviors then simulated to determine whether they are indeed unsafe with respect to the original, non-abstracted safety specification. The simulation-based analysis therefore does not need to sample from the whole search space of feasible behaviors. Instead, it focuses on those behaviors which are already identifies as potentially unsafe, making more efficient use of computational resources.

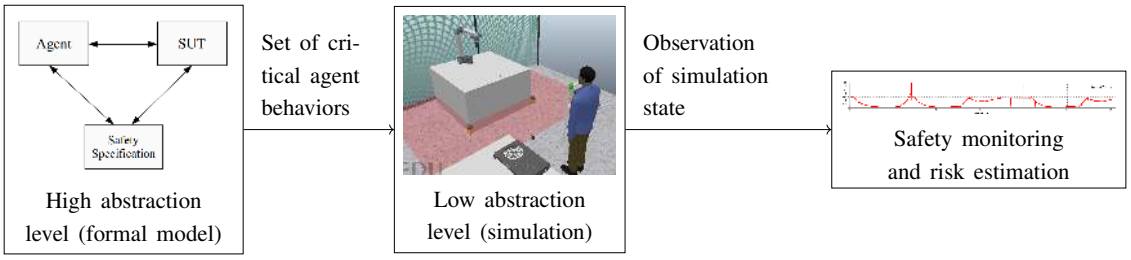


Figure 4.6.: Schematic of the two-level approach. (Author's own figure.)

5. Risk-Guided Search

THE AFOREMENTIONED CONCEPT OF RISK-GUIDED SEARCH (see Figure 4.4) is introduced in more detail in this chapter. Section 5.1 focuses on the risk metric. It discusses how to quantify the level of risk which is associated to a given simulation run. Section 5.2 focuses on the search algorithm. Possible search methods are discussed, and a suitable search algorithm is selected. On the basis of this algorithm, it is discussed how the risk information can be used to effectively find hazardous behaviors. Finally, section 5.3 presents a series of experiments to demonstrate the feasibility of the proposed techniques. In these experiments, risk-guided search is deployed to identify hazards in a set of exemplary HRC systems.

Publications Related to this Chapter
This Chapter is partially based on:

[91] **Huck, Tom P.**, Christoph Ledermann, and Torsten Kröger. "Virtual Adversarial Humans finding Hazards in Robot Workplaces". 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021.

Further information is also found in:

[90] **Huck, Tom P.**, Christoph Ledermann, and Torsten Kröger. "Simulation-based Testing for Early Safety-Validation of Robot Systems". 2020 IEEE Symposium on Product Compliance Engineering (SPCE). IEEE, 2020.

5.1. Risk Metric

5.1.1. Motivation for Introducing a Risk Metric

The purpose of the risk metric is to quantify the level of risk associated to a given simulation sequence. The risk information is used to guide the search for unsafe states. Unsafe states may be rare, especially for systems which already have safety measures that are relatively strong, but not

fully sufficient [126]. Thus, simply evaluating whether an unsafe state was found or not in a given simulation run would only provide sparse feedback to the search algorithm. For a more effective search, it is desirable that the search algorithm receives richer feedback, which not only indicates if an unsafe state *has occured*, but also provides some quantitative feedback indicating *how close* to an unsafe state a given state trajectory has come, even if no unsafe state is visited directly. As discussed in Section 3.3.3, falsification methods typically provide such a metric as a feedback to the falsification algorithm (see Figure 3.2). However, the choice of an appropriate metric is not trivial. Given perfect information about the state space, an obvious approach for this would be to define a distance metric over the state space, calculate the state in U that is closest to the current state s, and return the distance between that state and the current state. However, as mentioned before, U is not known explicitly, as it is only implicitly defined via the safety specification. Thus, a direct distance calculation is not possible and a heuristic needs to be introduced. Since the goal is to find *unsafe* states, a reasonable choice is a heuristic which reflects the level of *risk* that is inherent to a given simulation state. As will be shown at the end of this Chapter, optimizing the search strategy with respect to such a metric can significantly increases search effectiveness.

The need for a heuristic risk metric raises the issue of how to quantify risk. As discussed in Chapter 2, ISO 12100 defines risk as a quantity composed of the components *probability of occurrence* and *severity of outcome* [101]. Yet, the standard does not make any formal definition regarding the quantification of these aspects, as the risk estimation is mainly based on qualitative judgments of human experts. In a more formal approach, Madjumdar et al. [134] characterize the risk of a potentially hazardous situation with a probability distribution $P(\omega)$, $\omega \in \Omega$, where ω is a possible outcome of the given situation (e.g. a collision with a certain body part) and Ω is the set of all possible outcomes. The severity is represented by a cost function:

$$Z : \Omega \to \mathbb{R} \tag{5.1}$$

which assigns a cost (i.e., severity) to each possible outcome. While this definition is attractive from a theoretical standpoint because of its rigor, it is only practical when $P(\omega)$ and $Z(\omega)$ are known or can be estimated sufficiently accurately. Here, however, this is not the case due to two reasons: First, because estimating $P(\omega)$ would require accurate *probabilistic* models of the agent behavior (i.e., models that provide detailed information as to how likely it would be that the agent behaves in certain ways given a

certain situation). Such models are generally hard to obtain, because they require large amounts of data from which such distributions can be estimated. Even if a probabilistic model is available, its validity would only extend to the specific use-case or scenario for which the data has been obtained (e.g., a specific task or workflow). Second, even if a distribution of agent behaviors was known, estimating $Z(\omega)$ would require a large number of repeated simulation runs where individual behaviors $\omega \in \Omega$ are sampled from the distribution and then simulated in order to obtain a severity and estimate $Z(\omega)$. This would defeat the original point of the risk-guided search, which is to reduce computational cost by reducing the number of simulation runs that are required to find a hazard.

A more pragmatic approach is to omit the probabilistic aspects, and instead assign the risk value for a given simulation state directly on the basis of domain-specific safety-related variables such as speeds or distances, which can be observed directly from the simulation in each timestep. Thus, the risk metric is defined as follows:

Definition 8 - *Risk metric*
The risk metric r is a mapping from the simulation state space to a scalar, real-valued, and nonnegative risk score.

$$r : S \to \mathbb{R}^+ \tag{5.2}$$

The mapping is given by a heuristic function which takes into account safety-related variables from the respective simulation state.

Note that $r(s)$ only takes into account one simulation state at a time (i.e., momentary values for distances, speed, etc.). Yet, the goal is to obtain a risk estimate for a whole simulation run. Therefore, $r(s)$ is evaluated in each simulation timestep and after completion of the simulation run, the *maximum* risk value that has occurred over its course is returned. If multiple robots are present in the simulation scene, the risk is evaluated separately with respect to each robot, and the maximum is taken.

Of course, such a simplified metric is only an imperfect approximation, as it neglects the probabilistic aspects of risk. In the context of this work, however, this is acceptable, because the metric is intended as a heuristic to provide guidance and make the search more effective, and not as a definitive criterion for the final risk assessment. Final decisions on risk estimation and evaluation remain in the hands of human experts.

5.1.2. Choice of Risk Metrics

The risk metric should be based on the particular safety specification so that specifically those behaviors which falsify the safety specification receive a high risk metric. If risk metric and safety specification do not correspond well, there may be cases in which the search algorithm converges to behaviors which maximize risk, but do not necessarily result in a violation of the safety specification[1]. Thus, it is reasonable to define the risk metric on the same (or similar) safety-related variables as the safety specifications. For instance, if the safety specification specifies that a certain collision force limit must not be exceeded, it is reasonable to define the risk metric (at least partly) in terms of a collision force estimation. Furthermore, it is beneficial if the risk metric has a nonzero gradient at all points, as this allows the search algorithm to evaluate even subtle differences between subsequent simulation runs, and thus adapt the search accordingly. Metrics which stagnate over larger areas of the search space will not provide this type of fine-grained feedback. Finally, the risk metric should include a case distinction to distinguish between different types of situations which may occur in human-robot interactions. Although such a case distinction may lead to discontinuities in the risk metric, it allows the metric to adjust the risk quantification criteria depending on the current situation.

It should be emphasized that the design of the risk metric is a decision of the user which may vary from case to case. How a risk metric should look like concretely depends on the specific use case, the information that is available from the simulation environment, and the concrete formulation of the safety specification. Thus, one cannot define a generic risk metric which is fitting for all use cases. Throughout this book, different risk metrics will be used depending on the respective use cases and contexts. The following Example 5 (adapted from [89]) shows one possible risk metric that is applicable in the context of human-robot collaboration. A plot of the risk metric from Example 5 is shown in Figure 5.1. The plot shows the risk values for the body regions face, thorax, and hands as a function of the robot speed and the human-robot distance. For the collision force

[1] In the context of reinforcement learning, such effects are known as *reward hacking*. This term comes from the idea that if the reward does not correspond well with the intended goal of the user, the reinforcement learning agent may find a way to "hack" the task, that is, to exploit high rewards without satisfying the user's intended goals. Interesting discussions about this issue are found in [11] and [182]. A specific occurrence of reward hacking is discussed in Section 5.3.4

estimation, the model according to ISO/TS 15066, Annex A was used (see Eq. (2.2)-(2.6)) and an effective robot mass of 10kg was assumed.

Example 5 ***- Risk Metric for Human-Robot Collaboration Scenarios***
In the following, an exemplary risk metric for human-robot collaboration scenarios is presented. The metric distinguishes between three cases:

(a) Low-risk situations. In some configurations it is clear that the robot does not present an immediate hazard to the human worker, for instance where the distance between robot is large so that neither human nor robot can reach each other, or situations where the robot is standing still. In such configurations, the metric should take a very small value (or zero).

(b) Contact situations. Contact force and pressure are the major factors in determining criticality of contact situations [27]. Since pressure is notoriously difficult to estimate (see section 2.1.5), it is pragmatic to base the risk metric on estimated contact force. As discussed in section 2.1.5, contact force estimation needs to be body region-specific. A possible approach to consider this is to quantify collision severity for a risk metric is to take the ratio $\frac{F_c}{F_{max}}$ between the estimated contact force F_c and the maximum collision force F_{max} for the affected body region, so that both the potential collision force and the criticality are taken into account.

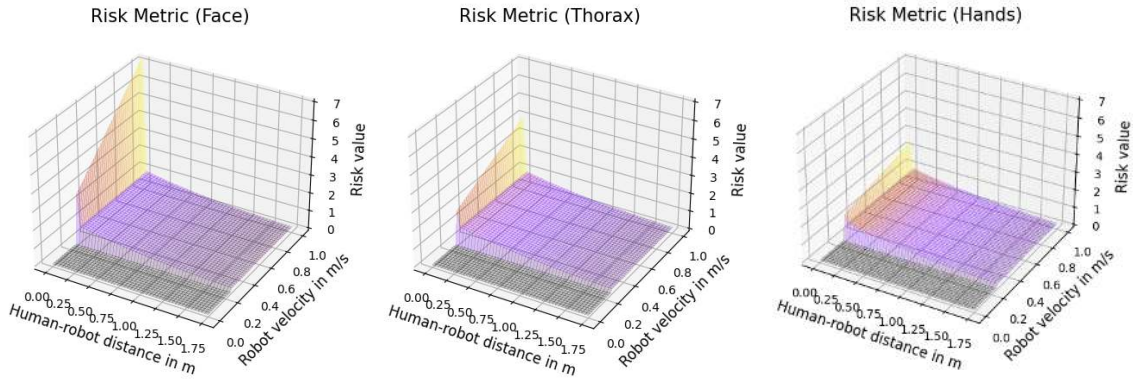
(c) Near-contact situations are in between the first and the second case. While no immediate contact is occurring, there is a potential collision hazard because the robot is moving in close vicinity to the human. In such cases it is reasonable to base the risk metric value on the human-robot distance d_{HR}. The risk metric should decrease as the distance increases and tend towards zero for large distances. One possibility for this is to define an inversely proportional relationship of the form $r \propto \frac{1}{d_{HR}}$ between the risk r and the human-robot distance d_{HR}. However, with an inversely proportional relation, small distances would lead to extremely high risk values, even if the potential collision severity is moderate. An alternative is to define an exponential relationship of the form $r \propto e^{-d_{HR}}$, as the exponential function decreases with increasing distance, but approaches an upper limit of one as the distance approaches zero.

These considerations lead to the following risk metric definition:

$$r(s) = \begin{cases} 0, & if \quad v_R \leq v_{crit} \\ e^{-d_{HR}}, & if \quad (v_R > v_{crit}) \wedge (d_{HR} > 0) \\ \frac{F_c}{F_{max}} + 1, & if \quad (v_R > v_{crit}) \wedge (d_{HR} = 0) \end{cases} \tag{5.3}$$

where d_{HR} denotes the distance between the robot and the closest human body part, F_c denotes the estimated human-robot contact force, F_{max} denotes the maximum permissible contact force for the affected body part, v_R is the robot's speed, which is defined as the cartesian speed of the fastest robot link, and v_{crit} is a user-defined speed threshold above which the robot is considered to be potentially hazardous.

As the plot illustrates, the risk metric decreases exponentially with increasing human-robot distance. The highest risk values are achieved for contact situations (i.e., $d_{HR} = 0$). In contact situations, the risk metric increases with increasing robot velocity and drops to zero below the critical threshold v_{crit} (here, 0.25m/s). The addition of +1 in the last case of Eq. (5.3) ensures that contact situation are assigned higher risk values than non-contact situations. Furthermore, note that the risk values in contact situations are generally higher for more critical body regions (i.e., face) and lower for less critical body regions (i.e., hands).



(a) Face **(b)** Thorax **(c)** Hands

Figure 5.1.: Plot of the risk metric from from Example 5. The risk score is plotted over robot velocity and human-robot distance for different affected body regions. (Author's own figure.)

5.2. Search Method

5.2.1. Search Problem Formulation

In the following, the problem formulation from Chapter 4 is briefly revisited. Recall Definition 7, where hazard analysis was characterized as a search problem given by the following tuple:

$$\langle \mathcal{M}, s_0, S, B, \mathrm{spec} \rangle \tag{5.4}$$

Where M denotes the simulation model, s_0 the initial simulation state, S the simulation state space, B the set of possible agent behaviors and spec the safety specification. Further, recall that the goal of the search problem

is to find an agent behavior $b \in B$ which, when executed in the simulation, results in a state trajectory which includes at least one unsafe state:

$$\mathcal{M}(b, s_0) = (s_0, s_1, s_2, ...) \quad \text{s.t. } \exists i : \text{spec}(s_i) = \text{false} \tag{5.5}$$

To mitigate search-space complexity, an approach is needed to efficiently find behaviors that lead to hazardous states. As discussed before, this may be achieved by deploying a search method which leverages information given by the risk metric to guide the search. This approach is detailed in the following sections.

5.2.2. Risk-guided Search with MCTS

In the following, agent behaviors shall be denoted by sequences of actions from an action space A (a more sophisticated description of agent behaviors will be introduced in the next Chapter):

$$b = (a_i) = (a_1, a_2, a_3, ..., a_n), \quad a_i \in A$$

With this formulation of agent behaviors, the search problem can be interpreted as a sequential decision making problem and formulated as a Markov Decision Process (MDP). Recall from Chapter 2 that an MDP is defined as follows:

$$\langle S, A, T, R, s_0 \rangle$$

Where S is the set of states, A the action space, T the transition function, R the reward, and s_0 the initial state. Here, the set of states is given by the state space of the simulation model, and the action space is given by the agent's action space. The transition function T is not given directly as a function, but implemented by the simulator, which returns a new state for a given state and action[2]. The reward function R is defined on the basis of the previously introduced risk metric.

Monte Carlo Tree Search (MCTS) is used for solving the hazard analysis problem. This algorithm is chosen for the following reasons:

- The suitability of MCTS for falsification problems has already been demonstrated in other application domains, notably in AST [126].

[2]It should be noted that MDPs are typically regarded as having probabilistic state transition whereas the simulator is assumed to be deterministic in this instance. This does not invalidate the analogy with MDP, since the simulator can simply be regarded as a transition function that returns the follow-up state with probability one.

- MCTS is a relatively simple procedure with only a small number of hyperparameters. It is therefore well-suited to demonstrate the proposed concepts without the need for extensive hyper-parameter tuning.
- MCTS incrementally builds a search tree where each node represents a simulation state and each edge an action. This representation form is well-suited for the sequential nature of the search problem.
- MCTS does not require an explicit state representation. Instead, the state is implicitly encoded in the search tree by means of the actions (edges) leading to the respective node. This makes the search algorithm easily applicable even for black-box systems whose internal state is unknown or only partially known [126].

The basic MCTS algorithm has already been introduced in Section 2.2.2 (Algorithm 1). Here, an adaptation of the MCTS algorithm is used (Pseudocode in Appendix B, Algorithm 2). To understand how MCTS is utilized for risk guided search, one can think of the possible agent behaviors and the resulting simulations as a search tree where the nodes of the tree correspond to simulation states and the vertices correspond to actions performed by the agent. The root of the tree corresponds to the initial state, and the depth of the tree to the length n of the action sequence that is simulated.

The simulation is run in lockstep with the MCTS algorithm. When the search algorithm starts in the root note, the simulation is set to its initial state. During tree traversal, each time the algorithm traverses from one node to another, the action corresponding to the respective edge is executed in the simulator. This procedure also holds for the expansion and rollout phases. At the end of the rollout, the maximum risk metric, which has been calculated in the simulator during the simulation run, is retrieved and backpropagated as a reward. Note that by running search algorithm and simulator in lockstep, it is ensured that each node in the search tree corresponds to a particular simulation state. Thus, the search algorithm can learn to find high-risk action sequence on the basis of its internal search tree without the need for an explicit representation of the simulation state.

5.2.3. Alternative Search Algorithms

The MCTS algorithm is chosen as an exemplary method to demonstrate the concepts of this work. However, this work does not claim that

it is necessarily the best or most efficient algorithm for the given problem. Beyond the chosen algorithms, there are further alternatives which can also be considered for risk-guided search.

Alternatives include other RL algorithms, for instance Q-Learning [44], Approximate Q-Learning [153], Deep Reinforcement Learning [64] and other Policy Optimization methods such as Trust-Region Policy Optimization or Proximal Policy Optmization [178]. Furthermore, MCTS adaptations for continuous action spaces could be considered [200, 124].

Similarly to reinforcement learning, which maximizes the cumulative reward, one can also use optimization algorithms which attempt to maximize or minimize an objective function. In other words, the search for hazards can also be characterized as an optimization problem where the goal is to maximize an objective function given by the risk metric. There is a vast range of different optimization algorithms in literature. Optimization algorithms frequently rely on analytical objective functions and/or their derivatives which requires information about the gradients of the objective function [171]. Here, however, the objective is obtained from simulation. Since the simulation is based on numerical calculations, it is generally not possible to obtain an analytical function which maps behaviors to risk metric values. Therefore, neither an analytical objective function, nor information about its gradient is available. This restricts the class of potentially suitable algorithms to the class of so-called *black-box optimization* algorithms, also known as *direct search* algorithms [160, 8]. These algorithms adapt their search strategies based on heuristics and point-wise evaluations of the objective function. Thus, they do not require an analytical formulation of the objective function or information about the gradient of the objective function. Examples for black-box optimization algorithms include Genetic and Evolutionary Algorithms [35], Simulated Annealing [172], Particle Swarm Optimization [192], and the Great Deluge Algorithm [59].

5.3. Experiments

5.3.1. Goal and Methodology

This section presents proof of concept experiments for the risk-guided search approach. The goal of these experiments is to to demonstrate feasibility and investigate in how far the use of risk-guided search improves the detection of unsafe states in comparison to a baseline approach which samples actions randomly and does not utilize any risk information for search guidance.

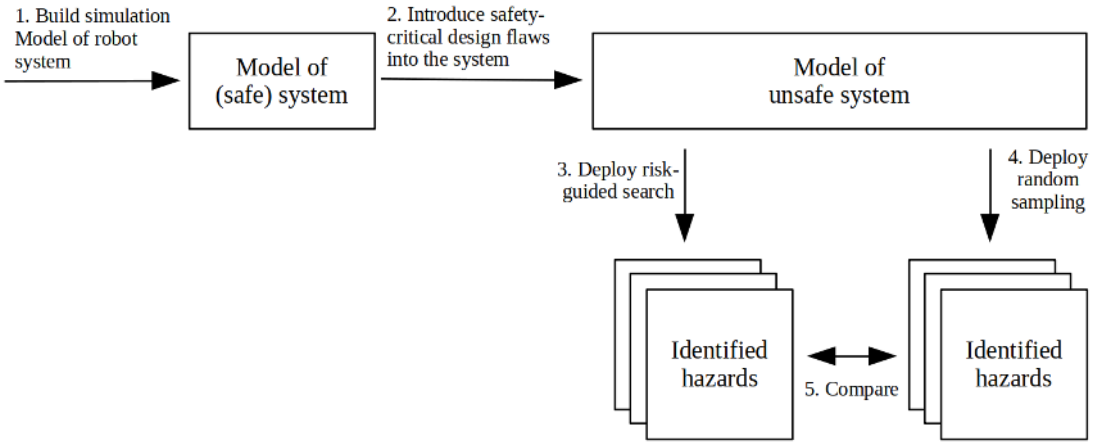


Figure 5.2.: Methodology for the experiments. Safety flaws are deliberately introduced into the system to create hazards. The risk guided search is deployed to find simulation runs leading to uncover these hazards. The performance in finding these simulation runs is compared against a baseline random sampling approach. (Author's own figure.)

The methodology of these experiments is as follows (Figure 5.2): Each experimental scenario consists of a simulation scene with a digital human model as an agent and a robot system as an SUT. In each scenario, a certain safety-critical design flaw is deliberately introduced into the respective SUT, resulting in certain collision hazards. The risk-guided search is deployed to find agent behaviors which provoke collisions and thereby expose the hazards. Since the safety flaws are deliberately introduced into the SUT, the hazards are known a-priori. This approach allows it to create controlled conditions where certain well-defined hazards are present within the test scenarios. It is important to emphasize that this knowledge

about the hazards is *not* encoded into the search algorithm in any form. In other words, the experiments are conducted as if the hazards were initially unknown. In order to compare how the risk information improves the effectiveness of the agent, a random sampling strategy is deployed for comparison. This baseline strategy does not rely on feedback from the simulation in form of a risk metric, but creates agent behaviors by sampling randomly from a uniform distribution over the action space.

5.3.2. Scenarios

The experiments are conducted in six scenarios. In each scenario, a collaborative robot cell is considered which contains certain safeguards to avoid potential human-robot collisions (e.g., laser scanners, light curtains, etc.). Depending on the scenario, the respective safeguards are altered to create certain hazards. For instance, safeguards may be partially removed, reaction times prolonged, or safety-distances reduced. The following Example 6 (descriptions and figures taken from [91]) illustrates two of the six test scenarios exemplary. For further details, the reader is referred to the original publication of the experiments [91] and to the appendix.

Implementation

The simulation scenarios are implemented in the *CoppeliaSim* robotics simulator [169]. For modeling the human agent, CoppeliaSim's default human model *Bill* is used. The action space of the agent in these experiments consists of a discrete set of six basic walking steps and six upper body movements, resulting in an action space of 36 movement combinations which can be concatenated into an action sequence that constitutes a more complex motion. To simplify the simulation and prevent an explosion of the search space, arm motions are not explicitly modeled. To account for the possibility that the human arms or hands could collide with the robot, a reachability check is performed at each simulation time step to check if the robot is in the reachable space of the human arms. Simulation states in which the robot is reachable by the human arms are evaluated as if they were collision states. Any collision where the contact force exceeds the collision force limit of the affected human body region is regarded an unsafe state.

Example 6 ***Test Scenarios***

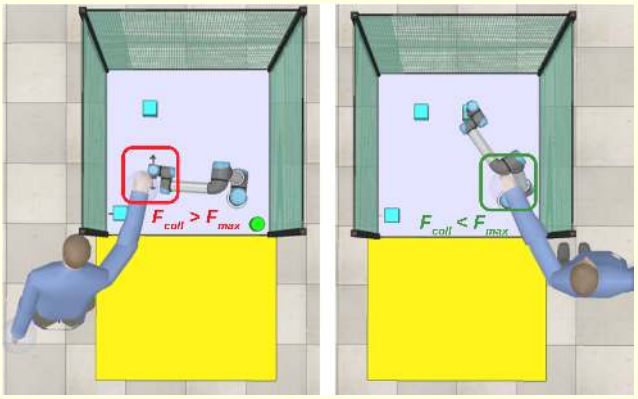


Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Scenario 5-A: *A robot is working in a cell encased by safety fences. The front of the cell is open. As a safety measure, a sensor mat is placed in front of the cell (yellow area in Figure above). The mat senses the human stepping on it and stops the robot when the worker steps on it. However, the mat's coverage is insufficient as it only covers the front of the cell, but not the right and left side. This raises the potential for collisions if the human walks reaches from the side around the fence into the cell (see figure above). Note that the human can cause a collision both on the left side and on the right side of the cell. However, only on the left side, the speed of the colliding robot part is high enough to exceed the collision force limit. If the collision happens on the right side, the contact force (and thus, the risk metric) is not as high as in case of a collision on the left side). The collision on the right side is therefore associated with local maximum of the risk metric.*

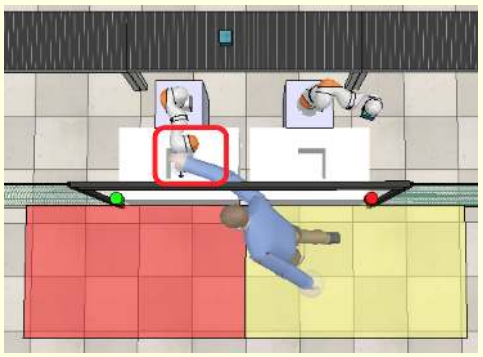


Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Scenario 5-B: *In this scenario, two robots are working in simultaneously besides each other, performing a pick-and-place task. Each robot is surrounded by a laser scanner zone which detects approaching humans and triggers an according safety stop (red and yellow areas in figure above). However, due to improper design of the zone's geometries, the human worker can cause a critical collision by entering through the right laser scanner field but then turning and reaching towards the left robot (or vice versa). In such a case, the robot opposite of the field through which the human has entered comes within human reach before a safety stop is triggered.*

The safety specification is thus defined as follows:

$$\text{spec}(s) = \begin{cases} \text{true}, & \text{if } d_{HR}(s) > 0 \vee F_c(s) < F_{max} \\ \text{false}, & \text{otherwise} \end{cases} \tag{5.6}$$

Contact force estimation and maximum force values are based on ISO/TS 15066 (see Section 2.1.5). For the search algorithm, the MCTS implementation of Lee et al. [126] is used.

The random sampling baseline is implemented by sampling actions sequentially from a uniform distribution over the action space A. In both MCTS and random sampling, an action sequence is terminated when an unsafe state is found, or if the maximum action sequence length is reached. Both MCTS and random sampling are performed in each of the six testing scenarios. Due to the stochastic nature of both approaches, the test runs are repeated multiple times to limit the influence of statistical outliers. Ten test runs with different random seeds are performed for each scenario and search method, resulting in a total of 120 test runs (i.e., 60 for risk-guided search and random sampling, respectively). Each test run is limited to a computational budget of of 320 search iterations (i.e., 320 action sequences, since each iteration corresponds to one action sequence). The maximum action sequence length is set to $n = 8$.

5.3.3. Results

The following criteria are evaluated to assess the performance of both methods:

- **Number of hazards missed** (N_{miss}): For each test scenario and search method, ten test runs are performed. It is counted in how many of the ten test runs the testing scenario's respective hazard was missed. In this context, *missed* means that none of the 320 action sequences contained an unsafe state (i.e., no behavior was found that exposed the respective unsafe state). This gives a measure of how *reliable* the respective methods are in exposing hazards.
- **Avg. number of iterations to discovery of hazard** (N_{iter}): For each scenario and search method, it is calculated how many search iterations were required on average until the first unsafe state is discovered (for test runs were the hazard was missed, the full number

of 320 iterations is counted into the average). This gives a measure of how *quickly* the respective search methods were able to detect hazards.

The results are given in Table 5.1. Out of a total of 60 test runs, the random search was unable to find an unsafe state in seven instances, amounting to a miss rate of ca. 11.7%. In contrast, the MCTS approach only was unable to find an unsafe state in one instance, amounting to a miss rate of ca. 1.7%. Furthermore, the MCTS approach required significantly fewer search iterations to find an unsafe state in all test scenarios. In some scenarios, namely scenarios 2, 3, and 6, the MCTS approach even took less than half the number of search iterations that was required by the random search to find an unsafe state.

Test Scenario	**Random**		**MCTS**	
	N_{miss}	N_{iter}	N_{miss}	N_{iter}
5-A	4	203.4	1	131.6
5-B	2	174.3	0	69.4
5-C	0	37.4	0	12.8
5-D	1	115.8	0	83.1
5-E	0	28.5	0	14.9
5-F	0	71.1	0	24.6

Table 5.1.: Results of the experiments [91] (Note that the order of scenarios differs from [91])

Special attention should be paid to the one test run in scenario 5-A where the risk-guided search was unable to find an unsafe behavior. This test run was examined closer by re-playing the search with the same random seed and observing the resulting behavior. It was found that the algorithm failed to find an unsafe state because it converged to a behavior which constitutes a local optimum in terms of the risk metric. This local optimum was already discussed in Example 6. As can be seen in Example 6 (top figure, right side), rather than avoiding the safety mat by walking around to the left, the human walks to the right around the mat and reaches into the robot cell from the right side. While this behavior does indeed lead to a collision, the collision speed is relatively low, as the robot can only be reached close to its base. Thus, the contact force is smaller and does not exceed the threshold F_{max}. Consequently, the discovered collision state is not an unsafe state. Meanwhile, the actual unsafe state (i.e., the collision on the left side of the cell) remains undiscovered.

5.3.4. Mobile Robot Case Study

In addition to the experiments presented above, a case study was performed, focusing on risk-guided search in an industrial use case. The objective was to identify safety-critical behaviors of a mobile robot navigating in the presence of human workers on an industrial shop floor. This case study is not entirely within the scope of this book, but it raises some interesting aspects which warrant a discussion. Therefore, it is discussed qualitatively in an abbreviated manner. For details and quantitative results, the reader is referred to [87].

The case study is focused on an industrial application with a mobile robot (Figure 5.3) which is navigating a shop floor in the presence of both static obstacles and human workers. Human workers are tracked by ceiling-mounted cameras and their positions are transferred to the mobile robot. The robot uses a nonlinear model-based predictive controller (NMPC). The NMPC predicts both the robot's own dynamics as well as the movements of human workers to calculate control inputs which should lead to collision-free and optimized paths. Additionally, a laser scanner at the front and back of the mobile robot triggers a safety stop should the NMPC be unable to avoid a collision. Obviously, the collision avoidance

Figure 5.3.: The industrial application from the case study. The depicted mobile robot is loaded by a robot arm on a gantry and has to deliver the parts autonomously on the shop floor, where human workers are present. Therefore, collision avoidance is crucial [87].

behavior of the mobile robot is crucial for the safety of human workers and needs to be tested. Due to the high system complexity, testing is conducted by means of simulation-based black box tests. The simulation environment includes a model of the mobile robot, a model of static obstacles, and a model of a human worker. To that end, the NMPC is embedded into the simulation environment as a black-box component which receives the resulting simulation state (i.e., position and orientation of robot and of the human) and sends control commands (i.e., velocity and yaw rate) to the mobile robot. The simulated human worker acts as test agent and attempts to provoke unsafe states. In this context, an unsafe state is defined as a collision where the relative movement between human and robot exceeds a critical limit. The simulation proceeds as follows: in every time step, the agent (i.e., the human worker) selects a velocity and yaw rate from a continuous action space and moves accordingly in the simulation environment. The mobile robot chooses control inputs and performs movements as well, taking the human position into account. From the resulting relative position and movement of human and robot, a risk metric is calculated. Based on the risk metric, a reward is issued to the agent.

In contrast to the previous experiments, Proximal Policy Optimization (PPO) is chosen instead of MCTS, as it is better suited to continuous action spaces. For details regarding the PPO algorithm, the reader is referred to Schulman et al. [178]. The PPO algorithm was deployed with two different reward formulations. In the first reward formulation, the reward is given immediately after each action, that is, the agent accumulates rewards over the action sequences. In the second reward formulation, the reward is recorded but withheld until the end of an action sequence. At the end of the action sequence, the agent is issued the maximum reward that has occcured over the course of the action sequence. Additionally, a baseline method is employed which samples the the action space randomly and is only constrained by a simple collision avoidance mechanism to avoid collisions with static obstacles such as walls. In the following, the RL algorithms with accumulative and maximum reward will be denoted by RL_{acc} and RL_{max}, respectively, and the random sampling baseline by RS.

Each of the three each algorithms (i.e., RL_{acc}, RL_{max}, and RS) is deployed in simulations with different initial states and different settings of the mobile robot. In these simulations, the risk guided search proved to be effective in identifying hazardous situations, and learned to exploit safety flaws of the mobile robot, such as depicted for example in Figure 5.4. As performance criteria, the number of identified critical collisions, as well

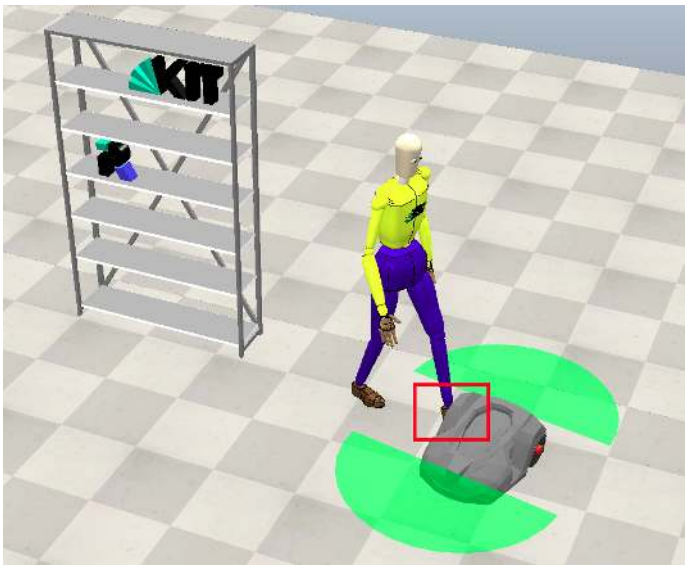


Figure 5.4.: The agent exploits a safety flaw, namely the gap between the robot's laser scanner detection zones (green) by waiting for the mobile robot to pass and then stepping in between the gap. Thereby, the agent creates a hazardous situation where the robot's safety measures cannot avoid a collision [87].

as the diversity of the different collision situations, were evaluated. With respect to both criteria, the risk guided search (i.e., RL_{max} and RL_{acc}) outperformed RS, with RL_{max} generally exhibiting a superior performance to RL_{acc}. However, the performance advantages were only significant with regard to the absolute number of collisions found. With respect to the diversity, performance gains of the risk guided search compared to RS were noticeable, but minor.

A notable finding of the case study was that RL_{acc} exhibited an interesting behavior that can be considered *reward hacking*. Reward hacking is a phenomenon where the agent succeeds in maximizing its reward (i.e., maximizing the risk metric), but fails to achieve the intended goal of the user (i.e., the discovery of unsafe states) due to a poor correspondence between the reward and the user's actual goal [182]. In this particular case, the agent learned to repeatedly provoke near-collisions rather than actual collisions. While an actual collision would have led to a higher reward, it would have terminated the action sequence, denying the agent the opportunity to incur further rewards. In contrast, repeatedly causing near-collision gives the agent an opportunity to collect medium rewards for as long as the maximum duration of the action sequence. This is an example of how misalignment between the reward formulation and the goal of the hazard

analysis can impede hazard identification. The dependence on the reward formulation is further indicated by the fact that the reward hacking behavior was virtually absent for RL_{max}, as this reward formulation does not incentivize the agent to accumulate rewards. Instead, the agent has an incentive to achieve a single reward that is as high as possible.

5.3.5. Discussion of the Experiments

In the experiments, the risk-guided search successfully found unsafe states and thereby uncovered the hazards which were introduced into the systems for testing purposes. Furthermore, the results confirm the premise of risk-guided search, namely that the use of search algorithms guided by a risk metric improves the chances of uncovering hazards in simulation-based testing scenarios compared to common random sampling approaches. The beneficial effect of leveraging the information provided by the risk metric is clearly seen when comparing the miss rate of the risk-guided search to that of the random sampling approach. The risk-guided search also performs significantly better in terms of the number of iterations needed to detect an unsafe condition. The superior performance shown in the experiments was also observed in the case study, albeit in a different use case and with a continuous rather than discrete action space.

Of course, there are some limitations. The experiments were conducted in simplified settings, especially with regard to the modeling of the agent. While this is sufficient to provide a proof of concept, it is not yet suitable analyze realistic HRC workflows. Furthermore, no constraints on the agent's behavior are considered. Also, the experiments show that with MCTS being a heuristic, non-exhaustive search, the approach can miss hazards at times. This is illustrated by the case of the MCTS algorithm failing due to convergence to a local minimum. A further limitation is the potential issue of reward hacking, as exhibited in the case study [87]. As seen in the case study, this problem is highly dependent on the reward formulation, and it may be difficult to decide on an appropriate reward formulation a-priori.

Nevertheless, the results are overall promising. They show that the approach is in principle suitable to achieve fully automated hazard identification on the basis of simulation models, without relying on human analysis or any other form prior knowledge about potential hazards. Furthermore, the experiments confirm the main assumption behind the risk-guided

search approach, namely that leveraging the risk information does indeed improve search performance significantly. The simplifications are reflections of the fact that the experiments were conducted in a proof of concept setting. They do not indicate principled limitations of the approach. Also, the possibility of incomplete hazard identification lies in the nature of of all testing- and falsification-based approaches that do not rely on an exhaustive search. It is not a shortcoming which is specific to the risk-guided search. Both limitations will be addressed in the following chapters. A more detailed modeling formalism for the human behavior, including constraints, will be introduced in Chapter 6, and the issue of completeness will be addressed in Chapter 7.

5.4. Chapter Summary

This Chapter has introduced the concept of risk guided search. In Section 5.1, a heuristic risk metrics has been introduced to quantify the level of risk associated to a simulation run. In Section 5.2, search procedure was introduced which leverages information from the risk metric as a guidance criterion. In particular, Monte Carlo Tree Search has been selected as a suitable algorithm for implementing risk-guided search. Finally, proof-of-concept experiments were conducted. In these experiments, the goal was to expose collision hazards in simulation models of different robot systems. The results of the experiments show that the concept of risk-guided search is feasible. Furthermore, the experiments indicate that leveraging risk metric information in conjunction with a search algorithm significantly improves the discovery of unsafe states compared to a random sampling baseline. However, it was also pointed out that it is possible for hazards to remain undiscovered, for example because search algorithms converge to local optima which exhibit a relatively high risk, but do not constitute an unsafe state.

6. Automata-constrained Risk-guided Search

In the previous chapter, a proof-of-concept was provided, showing that risk-guided search can be an effective approach to expose hazards. However, the proof-of-concept experiments were conducted in simplified settings and with simplified agent models. Furthermore, the experiments did not consider the fact that behaviors of the agent may be subject to constraints. While these simplifications are justified for a proof of concept, a more comprehensive model of agent behavior is needed for practical applications. This includes modeling formalisms to capture agent behaviors with both discrete and continuous aspects, as well as ways to capture and enforce constraints on the agent's behavior. Section 6.1 illustrates these needs with a simple example. An extended model of agent behavior is introduced in Section 6.2. Formalisms to model and enforce constraints on the agent behavior are introduced in Sections 6.3 and 6.4. Finally, Section 6.5 presents a series of experiments where the two newly introduced aspects are combined with the risk-guided search from the previous Chapter.

Publications Related to this Chapter
This Chapter is partially based on:

[89] **Huck, Tom P.**, Christoph Ledermann, and Torsten Kröger. "Testing Robot System Safety by Creating Hazardous Human Worker Behavior in Simulation." IEEE Robotics and Automation Letters 7.2 (2021): 770-777.

[93] **Huck, Tom P.**, et al. "Hazard Analysis of Collaborative Automation Systems: A Two-layer Approach based on Supervisory Control and Simulation." 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023.

Further information is also found in:

[49] Constantin Cronrath, **Tom P. Huck**, Christoph Ledermann, Torsten Kröger, and Bengt Lennartsson. "Relevant Safety Falsification by Automata Constrained Reinforcement Learning". 2022 IEEE International Conference on Automation Science and Engineering (CASE). IEEE, 2022.

6.1. Introductory Example

The following Example 7 (adapted from [93]) presents a HRC scenario which will be used as a running example throughout this Chapter.

Example 7 - ***Exemplary HRC Task***

Exemplary HRC workstation (Author's own figure).

Consider the HRC workstation depicted above. Assume that the collaborative task performed by human and robot is as follows: the human walks from the center area to the storage table, retrieves a part from the table, places it in front of the robot, then walks back to the center, and activates the robot at the control panel. The robot performs a procedure (e.g., drilling) on the part until the worker stops the robot again through the control panel. The worker then retrieves the part and places it back onto the storage table. As a safety measure, the area around the robot is monitored by a laser scanner (red area). A safety stop of the robot is triggered when the worker enters the detection area. To monitor the ongoing work of the robot from close distance without triggering a safety stop, the worker can activate the robot in a safety override mode, where the stop triggered by the laser scanner is overridden.

Note that in the previous Chapter, the behavior of the agent was modeled as sequence of simple movements from a purely discrete or continuous action space (see Section 5.3, Eq. (9.1)). Clearly, this is not sufficient for modeling agent behaviors in scenarios such as the one in Example 7. Furthermore, the previous Chapter has not considered the fact that not all permutations of actions from the action space are necessarily feasible. Of-

ten, the possible action sequences of the agent are limited, as dependencies between actions constrain the feasible order of actions. In Example 7, for instance, the worker can only place the workpiece in front of the robot if it has been picked up previously. Furthermore, the worker can only pick up the workpiece after walking to area B, and so on. Failure to consider such constraints can lead to several problems: First, the search may find supposedly hazardous behaviors that are not realistic. This does not only waste computational resources, but may also distract from actually critical behaviors. Second, attempting to perform action sequences which are infeasible could, depending on the implementation of the simulation, drive the simulation into deadlocks or other error states. These issues highlight the need for more sophisticated modeling of the agent's behavior. Solutions are discussed in the following Sections.

6.2. Description Format for Agent Behaviors

The techniques developed in this book are mainly geared towards HRC as a use-case. Therefore, it is reasonable to assume that the testing agent will usually represent a human that is interacting with the robot. Because real-world human behavior is highly complex, such a description format naturally requires a certain degree of abstraction. Marvel et al. have shown that human behavior in human-robot interactions can be described hierarchically by first decomposing it into relatively complex worksteps, which in turn are then further decomposed over multiple levels until a level of atomic actions is reached [139]. Bobick [30] differentiates between three abstraction levels of human behavior: Atomic *movements*, *activities*, which are sequences of atomic movements, and *actions*, which involve several activities and often entail complex interactions with the environment. Breaking down the agent behavior to such fine-grained hierarchy like in [139] and [30] is not done here to avoid overly complicated agent models. Instead, this work takes a pragmatic approach and differentiates between action- and parameter-level from the perspective of the resulting search space: The discrete action level includes all aspects of the behavior that can be described by sequences of *discrete* labels (e.g., discrete worksteps in an assembly sequence), whereas the parameter-level includes features that are represen-ted through continuous-valued parameters (e.g. coordinates or velocities of movements).

The discrete aspects of the behavior shall be denoted by an action sequence (a_i) of events from a discrete action space A. The continuous parameters shall be described by a real-valued parameter vector (θ_j). Together, action and parameter sequence constitute an *agent behavior* b (see also [89]):

Definition 9 - *Agent Behavior*
The behavior b of the agent is modeled by a tuple of an action sequence (a_i) and a corresponding parameter vector (θ_j):

$$b = \langle (a_i), (\theta_j) \rangle \tag{6.1}$$
$$(a_i) = (a_1, a_2, ..., a_n), \quad a_i \in A \tag{6.2}$$
$$(\theta_j) = (\theta_1, \theta_2, ..., \theta_m), \quad \theta_j \in \mathbb{R} \tag{6.3}$$

The set of all possible agent behaviors is denoted by B:

$$B \subseteq A^n \times \mathbb{R}^m \tag{6.4}$$

Note the following remarks regarding the definition above:

- The length n of the action sequence is not necessarily equal to the length m of the parameter vector, since some events might be associated with multiple parameters, whereas other events might have no associated parameter.
- Intuitively, the action sequence can be thought of as an abstract template describing *what* actions the agent performs in what order, while the parameters describe *how* the agent performs the respective actions.
- In the previous Chapter, the search space of possible agent behaviors consisted of the set of possible action sequences A^n. Now, the search space is given by a subset of the cartesian product of the sets of discrete action sequences and continuous parameter combinations. This leads to a *mixed discrete-continuous* search space (see Eq. (6.4)).

The following example illustrates the new definition of agent behaviors more concretely [93]:

Example 8 ***- Action Sequence and Parameters.*** *Consider again the scenario from Example 7. The agent's action space in this scenario is modeled as follows:*

$$A = \{t_1, t_2, u_S, d_S, u_R, d_R, b_1, b_2, b_3, r\} \tag{6.5}$$

With the individual actions as defined below:

Action	***Explanation***
t_1	*Walking between area A and B*
t_2	*Walking between area A and C*
u_S	*Picking up the part at the storage table*
d_S	*Putting down the part at the storage table*
u_S	*Picking up the part at the robot station*
d_S	*Putting down the part at the robot station*
b_1	*Pressing button to activate robot*
b_2	*Pressing button to activate robot in safety override mode*
b_3	*Pressing button to stop robot*
r	*Retracting hand after reaching motion*

The following action sequence describes a possible workflow where the worker walks to the storage area, picks up a part, walks to the robot and places the part at the robot station, and then walks back to press a button (e.g., to activate the robot).

$$(t_1, u_S, r, t_1, t_2, d_R, r, t_2, b_1) \tag{6.6}$$

Continuous parameters may include the walking velocity of the human worker, which influences whether the robot can stop in time when the human enter the laser scanner zone. Also, one might consider the possibility that the worker's reaching motion is not consistent when placing the workpiece in front of the robot, which might lead to collisions between the robot and the human worker's hands and arms. A possible parameter vector could therefore be as follows:

$$\underline{\theta} = (v_H, \Delta x, \Delta y) \tag{6.7}$$

where v_H denotes the human worker's walking speed, and $\Delta x, \Delta y$ denotes the longitudinal and lateral displacement of the position where the worker puts down the workpiece.

Choosing an appropriate abstraction level for modeling the agent behaviors is a use-case specific question to which no general answer can be given. The abstraction level needs to be chosen on a case-by-case basis under consideration of the specific use-case and the desired level of detail of the analysis. Also, the achievable level of detail may be limited by the capabilities of the simulation software. The same is true for the choice of continuous parameters. Obviously, it is not reasonable to parameterize every detail of the agent's behavior as this would lead to an excessively large

search space. Instead, the choice of parameters should focus on safety-critical aspects. Furthermore, the choice of parameters may also be limited by the degree of detail of the simulation.

Note that the models introduced above only cover the behavior of the agent and do not include any description of the behavior of the SUT. This is due to the fact that the SUT's behavior is already encoded in the simulation model of the robot system. In other words, the agent's behavior is used as a simulation input, while the SUT's behavior is an emergent behavior that results from the SUT's programming and the agent behavior (see also Section 4.1.2). Consequently, describing the agent's behavior is sufficient for the purposes of this Chapter. An extension of the model which covers both agent and SUT will be discussed in Chapter 7.

6.3. Constraints on the Agent's Behavior

In addition to the previously introduced description format, a formalism is needed to express constraints on the feasible agent behaviors. On the continuous parameter level, it is straightforward to enforce constraints by restricting the sampling of the parameters to a certain interval given by parameter limits $\theta_{min,j}$ and $\theta_{max,j}$. On the action level, however, the formulation of constraints is more challenging because the constraints are dynamic, that is, the set of currently feasible actions depends on the history of previous actions and thus varies over time. To express such dynamic constraints, the behavior of the agent is abstracted as a discrete-event system (DES). Several modeling formalisms for DES are known, such as Petri Nets (PN), Finite Automata (FA), and Extended Finite Automata (EFA). In this work, EFA are chosen, since EFA are more intuitive to read than PN and more compact than FA (see also Section 2.2.3). Recall from Section 2.2.3 that an EFA is an automaton model which extends finite automata with additional modeling formalisms, namely a set of variables V, guards G and modifiers M. Guards are logical expressions over the variables in V. A transition which is associated with a guard $g \in G$ can only be taken if g holds true given the current variable values. Upon execution of the transition, the modifier is executed and modifies one or multiple variables $v \in V$. The feasible behaviors of the agent shall be described by an EFA called the *agent EFA model*. The input alphabet of this EFA corresponds to the action space A (i.e., an event is associated with an action of the

agent). The use of variables, guards, and modifiers allows for the encoding of constraints on the feasible action sequences.

Remark: The notion of *locations* in EFA should not be confused with spatial locations of the agent. Although in the example presented below, the locations in the EFA sometimes correspond to spatial locations of the agent, they are not necessarily to be understood in a spatial sense, but can also be used to express other aspects of the agent's current state.

Definition 10 ***Agent EFA***
The EFA which describes the feasible action sequences of the agent is called agent EFA and shall be denoted by $\mathcal{A}$:

$$\mathcal{A} = \langle A, V_{\mathcal{A}}, L_{\mathcal{A}}, \rightarrow_{\mathcal{A}}, G_{\mathcal{A}}, M_{\mathcal{A}}, l_{0\mathcal{A}}, V_{0\mathcal{A}} \rangle \tag{6.8}$$

Where A is the action space of the agent, $V_{\mathcal{A}}$ is a set of bounded discrete variables, $L_{\mathcal{A}}$ is the set of locations, $G_{\mathcal{A}}$ the set of guards, $M_{\mathcal{A}}$ the set of modifiers, $\rightarrow_{\mathcal{A}}$ is the conditional transition relation, $V_{0\mathcal{A}}$ is the initial assignment of the variables, $l_{0\mathcal{A}}$ the initial location, and $L^m_{\mathcal{A}}$ the set of marked locations.

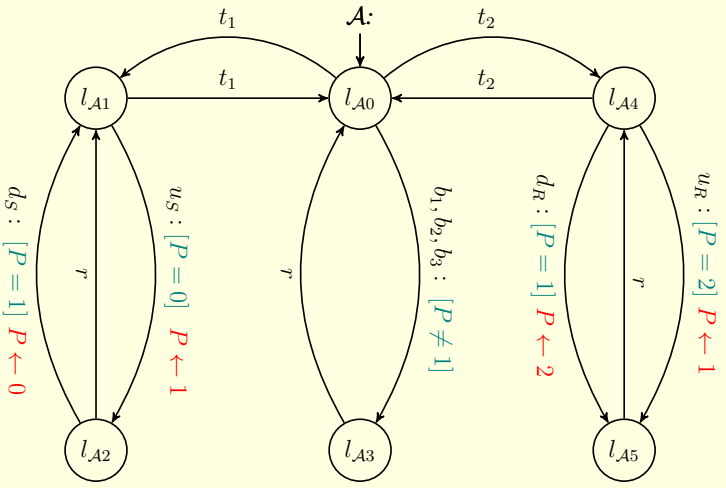
At this point, the reader might ask why it is necessary to run simulation model and agent automaton in parallel when the feasibility of actions could also be checked directly on the basis of the observed simulation state. The reason for this is twofold: First, maintaining a separate model reduces the reliance on the internal state of the simulator, thus preserving the black-box character of the approach.

Second, a lightweight model such as an automaton can be used to identify potentially critical behaviors a priori, that is, before running any simulations. Such a preprocessing step allows it to focus the more computationally expensive simulation efforts on potentially more interesting regions of the search space (more on this approach in the next Chapter).

The following Example 9 illustrates the use of EFA for modeling the agent's behavior and related constraints [93].

Example 9 ***- Agent EFA modeling***

Consider again the HRC scenario from Example 7 with the corresponding action space from example 8. The figure below shows an agent automaton $\mathcal{H}$ for this scenario. Circles represent locations, arrows to events, guards are denoted in green and inside square brackets [ ]. Modifiers are denoted in red behind the square brackets. The events correspond to the action space as defined in Table 8. In addition to the events, the variable $P \in \{0, 1, 2\}$ (initial value: $P = 0$) is introduced to track the position of the workpiece (0: at storage, 1: in worker's hands, 2: at robot station). The initial location is $l_{\mathcal{H}0}$. Note that some events require guards to be fulfilled before the transition is enabled. By means of these guards, workflow constraints are enforced. For instance, to put down a workpiece at the robot station d_R, the guard is $[P = 1]$, because the worker holding the part is a prerequisite for putting it down.



Note that the input alphabet of $\mathcal{A}$ corresponds to the action space A of the agent (i.e., each event corresponds to an action of the human worker). Thus, the notion of *automata acceptance* can be used to distinguish between feasible and infeasible behaviors. For instance, in Example 9, the action sequence

$$(t_1, u_S, r, t_1, t_2, d_R, r, t_2, b_1)$$

is a feasible behavior and constitutes an accepted input for the automaton. The reader can easily verify this by tracing the transitions through starting in location $l_{\mathcal{H}0}$. In contrast, the sequence

$$(t_1, d_S, r, t_1, t_2, d_R, r, t_2, b_1)$$

represents an infeasible behavior where the worker puts down the workpiece before picking it up. Observe that this sequence is not accepted by the automaton, since the guard statement $P = 1$ for the second transition d_S is not fulfilled.

Thus, the set of feasible action sequences corresponds to the the set of action sequences that are accepted by the automaton. Recall that the set of accepted input sequences is called the *language* of the automaton. In other words, an action sequence can only be performed by the agent if it is included in the language $L_{\mathcal{A}}$ of the agent automaton. Although the state space of $\mathcal{A}$ is finite, $L_{\mathcal{A}}$ may include infinitely long action sequences (e.g. loops which contain marked states). Since it is only possible to simulate sequences of finite length n, the set of feasible behaviors is further restricted to a subset $L_{\mathcal{A}}^{n}$, which only includes sequences up a certain maximum length n.

Definition 11 ***- Feasible Action Sequences***
The set of feasible action sequences is the set of action sequences which:

- *are part of the language of the automaton* ***and***
- *have a length less than or equal to the maximum sequence length n.*

This set shall be denoted by $L_{\mathcal{A}}^{n}$:

$$L_{\mathcal{A}}^{n} = \{(a_i) : \ (a_i) \in L_{\mathcal{A}}, \ |(a_i)| \leq n\}$$

Recall that an event sequence must end in a marked location to be a considered a part of the automaton language. Throughout this book, there will be some cases where all locations are considered to be marked and thus no distinction between unmarked and marked locations are drawn. In such cases, the graphical representation with double-lined circles is omitted for notational simplicity. Thus, if one of the following EFA contains no explicitly marked states, all states are treated as marked states, whereas if an EFA does contain explicitly marked states, only these are treated as marked states.

6.4. MCTS Adaptation for Automata-Constrained Risk Guided Search

In the following, the risk-guided search from chapter 5 is combined with the newly introduced agent model. The primary change compared to the chapter 5 is that the space of possible action sequences does not consist of A^n anymore, but only of a certain subset of feasible sequences $L^n_{\mathcal{A}} \subseteq A^n$ that is given by the language of the agent EFA model $\mathcal{A}$ (compare Def. 11). The second change is that the agent's behavior does now not only feature action sequences, but continuous parameters as well.

To facilitate these changes, the risk-guided search is adapted. Pseudocode of the adapted algorithm can be found in Appendix B, Algorithm 3. Compared to the method from chapter 5, there are two major changes:

- Continuous valued motion parameters are introduced. Thus, the algorithm not only needs to sample an action sequence, but also a parameter vector. While the action sequence is sampled by the MCTS algorithm, the parameter vector is sampled randomly from a uniform distribution between the respective parameter limits $\theta_{min,j}$ and $\theta_{max,j}$. The sampled parameters are given to the simulator at initialization of the simulation run.
- Second, the adapted algorithm also maintains a representation of the agent automaton which is executed in lockstep with the simulation. Each time the agent performs an action in the simulator, the same action is also fed into the automaton, which performs a corresponding transition and thus updates its internal state. In the expansion and rollout phase, where new events are sampled, the current state of the automaton is evaluated to determine the currently feasible events. Thereby, the sampling is restricted only to the currently feasible subset of the action space, subjecting the algorithm to the dynamic constraints which are formulated in the automaton. Both simulator and automaton are reset to their initial states at the beginning of a new search iteration.

Regarding the first point, it should be noted that in principle it is also possible to sample the parameters with a parameter optimization method such as Simulated Annealing or the Nelder-Mead Algorithm (see Section

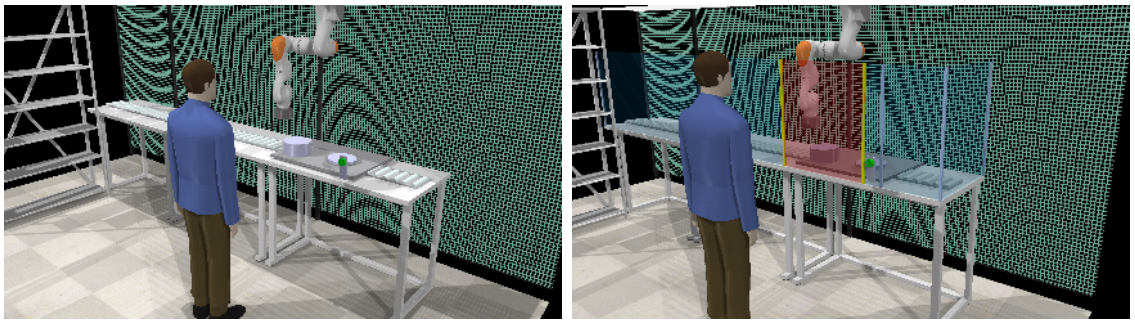
5.2.3) rather than sampling them randomly. Such a parameter optimization method could be used to find a set of parameters which maximizes risk. However, this would lead to a two-stage optimization problem, where each action sequence presents a parameter search problem in itself, where the same action sequence needs to be sampled several times until the parameter optimization converges. For search problems with a large number of feasible action sequences, extensive computation time is needed to solve such an optimization problem for each action sequence. Thus, a random sampling strategy is chosen here. A two-stage search approach which uses parameter optimization methods, namely the Nelder-Mead optimization algorithm, has been implemented in some experiments which are not presented in this book. However, no significant advantage over an MCTS algorithm combined with randomization of continuous parameters was found. For a more in-depth discussion on this, the reader is referred to [89].

6.5. Experiments

6.5.1. Goal and Methodology

In the following, a series of experiments is presented to demonstrate the newly introduced techniques. Like in the previous chapter, the experiments are based on HRC scenarios which contain specific hazards and the developed techniques are deployed to identify these hazards. The main difference to the previous chapter is that the experiments here consider more complex tasks where constraints on the agent's behavior need to be considered. Further, the search space in these experiments consists of a combination of discrete events and continuous-valued parameters. As in the previous experiments, the performance of the risk-guided search is compared with a random sampling approach. The experiments were published in [93]. For more details on the experiments, the reader is referred to the publication.

The experiments are conducted in three scenarios 6-A to 6-C. Scenario 6-A is based on the same exemplary HRC system that was also used as a running example throughout Examples 7-9. In this scenario, there are two safety flaws which can lead to potentially hazardous situations:



(a) Scenario 6-B **(b)** Scenario 6-C

Figure 6.1.: Simulation scenarios 6-B (left) and 6-C (right). Scenario C features the same collaborative task as scenario B, but with the difference that access to the shared workspace is only possible through a light curtain (red area). (Author's own figure.)

- A time delay between the moment that the human enters the safety zone and the moment that the robot stops can lead to a collision as the robot may not be stopping fast enough depending on how the worker approaches.
- There is a safety override button. This allows the worker to deactivate the safeguard, e.g. for inspecting the process. However, the overriding of the safety function can also lead to a collision hazard.

Scenario 6-B is shown in Fig. 6.1a. It is modeled after a real-world HRC application in a German automotive manufacturing plant [121]. In this scenario human and robot collaboratively assemble a gearbox. In a normal assembly sequence, the worker first retrieves parts (bearings) from a shelf. One of the parts is inserted into the housing of the gearbox. Afterwards, the worker presses a button to activate the robot which then moves over the housing and presses the gearwheel into the housing in a downward motion. Meanwhile, the worker inserts the second bearing into the cover. Finally, the worker mounts the cover onto the housing. Potential hazardos in this scenario include the hand of the worker being crushed between gearwheel and housing, as well as a collision between the worker's head and the robot's elbow joint, especially of the worker leans over the gearbox while the robot is moving towards the gearbox or upwards.

Scenario 6-C is based on scenario 6-B and features the same workflow, but the robot system is safeguarded by an additional light curtain (see Fig. 6.1b) which is placed in front of the robot as a safety measure. In this scenario, the safety flaw consists of a prolonged response time of the light curtain and the robot safety stop. This leads to potential collision hazards.

However, the light curtain makes collisions much rarer, and thus, it is significantly more difficult in this scenario for the search algorithm to find hazardous behaviors. A more detailed description of the scenarios, including the definition of action space, agent EFA, and motion parameters, is given in Appendix A.2.

6.5.2. Implementation

As in the previous chapter, simulation models for the three scenarios are created in *CoppeliaSim*. The agent automata are modeled in SUPREMICA, a tool for analysis and synthesis of DES [135]. SUPREMICA automatically converts the EFA representation of the agent automaton into an equivalent FA which is then is exported as an XML file (recall that any EFA can be converted into an equivalent FA as discussed in Section 2.2.3). The exported file serves as the automata representation for the algorithm. The algorithm saves the current state of the automaton. Each time a state transition is performed, the state is updated based on the transitions in the XML file. The set of feasible actions is obtained by looking up all feasible transitions given the current state. The random sampling approach is implemented by calculating the set of feasible action sequences in advance from the XML file, and then randomly sampling from a uniform distribution over the calculated sequences. The motion parameters are sampled randomly from a uniform distribution over the interval $\theta_{min,j}, \theta_{max,j}$, both in the risk-guided search and in the random sampling approach.

As a safety specification, it is defined that all collision states with a contact force exceeding 70% of the ISO/TS 15066 collision force threshold shall be deemed as unsafe[1]:

$$\mathrm{spec}(s) = \begin{cases} \text{true}, & \text{if } (d_{HR}(s) > 0) \vee (F_c(s) < 0.7 \cdot F_{max}) \\ \text{false}, & \text{otherwise} \end{cases} \tag{6.9}$$

For the contact force estimation, the collision model according to ISO/TS 15066 is used (see Section 2.1.5). The reward for the MCTS algorithm is defined as the maximum risk metric which has appeared over the course of the action sequence. The risk metric is defined according to Eq. (5.3) from Section 5.1.2.

[1]Note that this safety specification is stricter than in the previous experiments, where only collisions exceeding the full limit were deemed unsafe. The stricter specification was chosen to find a more diverse range of unsafe states, including collision states where the force threshold is not fully exceeded.

For both risk guided search and random sampling, ten test runs with different random seeds are performed in each of the three scenarios. For a fair comparison, each test run is limited to the same computational budget, namely 500 simulation runs with a maximum length of $n = 12$ (scenario 6-A) and $n = 10$ (scenarios 6-B and 6-C), respectively. For each simulation run ending in an unsafe state, the corresponding action sequence, parameter vector, and maximum risk metric are logged.

6.5.3. Results

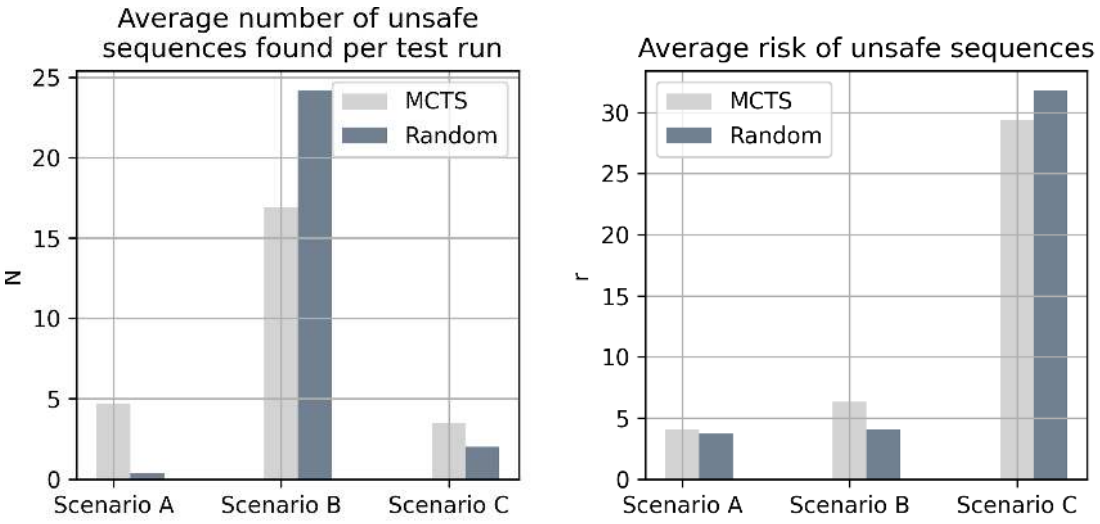
The performance is compared in terms of:

- N: The number of found action sequences leading to an unsafe state.
- r_{mean}: The mean risk value of the found unsafe sequences.

Note that these criteria differ from the performance criteria in the previous chapter. Previously, the aim of the experiment was to show that risk-guided search generally improves the chances of discovering a given hazard. In this case, where the scenarios are more complex, there may be multiple hazards or multiple action sequences leading to an unsafe state. Thus, instead of just determining whether a hazard was found or not, it is investigated how many different unsafe action sequences are found. Furthermore, the risk metric values are recorded to investigate not only how many unsafe sequences are found, but also how critical the associated behaviors are.

The results are shown in Figure 6.2. As seen in Figure 6.2a, the risk-guided search significantly outperforms the random baseline in terms of the number of unsafe action sequences found in scenarios 6-A and 6-C. In scenario 6-B, however, the risk-guided search finds *fewer* unsafe action sequences than the random baseline. Furthermore, as seen in Figure 6.2b, the behaviors created by the risk-guided search are, on average, associated with higher risk scores than those created by the random sampling. A likely explanation for this is that the risk-guided search converges to certain behaviors which are then sampled repeatedly with different parameters, thus managing to achieve on average higher risk in a hazardous sequences than the random sampling, where high-risk sequences are not more likely to be sampled repeatedly than other sequences.

While the strong performance in scenarios 6-A and 6-C is in line with the previously found performance advantage of the risk-guided search, the



(a) Number of hazardous sequences **(b)** Average risk

Figure 6.2.: Results of the Experiments from Section 6.5. (Author's own figure.)

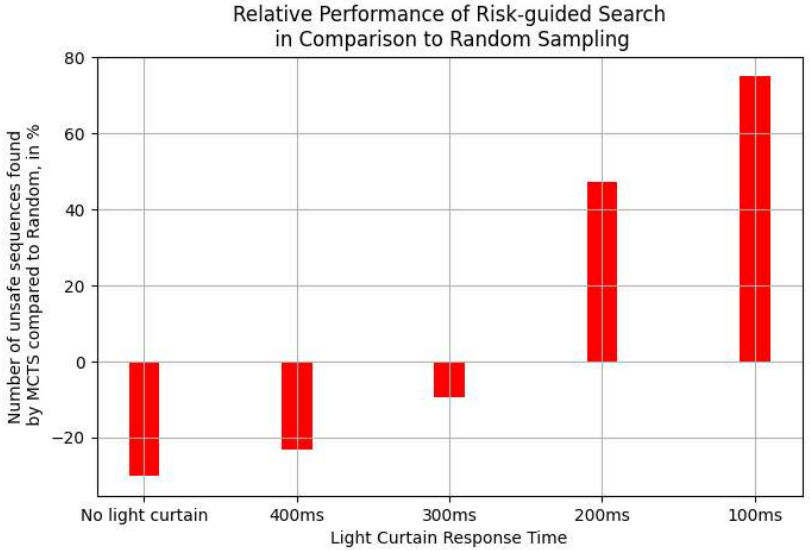


Figure 6.3.: Relative performance of risk-guided vs. random search for different light curtain response times. (Author's own figure.)

relative under-performance of the risk-guided search in scenario 6-B is somewhat unexpected. After all, in the previously presented experiments, the risk-guided search has consistently outperformed random sampling. A possible explanation is that the safety measures in scenario 6-B are relatively weak compared to those in the other scenarios. Thus, unsafe sequences are easier to find. In other words, the probability that any given sequence is unsafe is relatively high in scenario 6-B. This favors an unbiased random search which samples a broad and diverse set of sequences. In contrast, the more challenging scenarios 6-A and 6-C, where unsafe states are rarer and unsafe sequences are thus more difficult to find, favor the risk-guided search.

To investigate this assumption closer, further experiments were conducted. In these experiments, multiple variations of testing scenario 6-C were created. The instances differ only with regard to response time of the robot's safety stop which occurs as a reaction of the activation of the light curtain. The smaller the response time, the quicker a safety stop is triggered. Obviously, with a quicker safety stop, it is more challenging to create agent behaviors which result in collisions. Thus, the search problem becomes more challenging as the response time decreases. The results are shown in Figure 6.3. The horizontal axis denotes the response time. On the left, the graph starts with an infinite response time, that is, no reaction is triggered (this essentially corresponds to scenario B where not light curtain is present). Then, the response time is decreased gradually, making the search problem increasingly challenging, until it reaches the original response time that corresponds to scenario C from Figure 6.2. The vertical axis denotes the relative performance of risk-guided and random search. The relative performance is measured as follows:

$$\text{relative performance} = \frac{N_{risk-guided} - N_{random}}{N_{random}}$$

where $N_{risk-guided}$ is the number of unsafe sequences found by the risk-guided search and N_{random} the number of unsafe sequences found by the random search. As Figure 6.3 shows, the risk-guided search initially underperforms compared to the random search in the less challenging cases, while outperforming the random search in more challenging ones. In other words, the comparative advantage of the risk-guided search grows as the test scenarios become more challenging.

6.6. Chapter Summary

In this Chapter, the risk-guided search method was extended to enable more comprehensive and realistic agent behavior. To that end, two additions were introduced. First, the agent behavior, which formerly only covered discrete actions, was extended by a set of parameters to describe continuous-valued aspects of the agent behavior.

Second, an automata-based description format was introduced to capture constraints with respect to the agent's feasible action sequences. For this, Extended Finite Automata (EFA) are used. Although their expressive power is not higher than that of ordinary finite automata, EFA offer additional modeling formalisms which allow for easier and more concise modeling, and their representations are more compact and easier to understand for human users. This makes the modeling of complicated workflows more manageable. With the automata-based model, the notion of accepted languages is used to decide if a given action sequence is feasible or not. In particular, the set of feasible action sequences is defined as the language of the automaton.

The risk-guided search approach was subsequently adapted to include the constraints expressed by the EFA. To that end, the EFA is run in lockstep with the MCTS algorithm. In each search step, the EFA is evaluated to provide the currently feasible set of actions, from which the MCTS algorithm then samples its next action. Additionally, random sampling of the continuous parameters is introduced to capture the variability of some of the agent's continuous parameters (e.g. varying walking speeds of human agents).

The adapted search was tested against a random baseline in different industrial HRC scenarios. As in Chapter 5, each test scenario was designed to include certain hazards, and the search method was deployed to find behaviors leading to corresponding unsafe states. The experiments indicated that the random baseline performed better than the risk-guided search in easier scenarios (i.e., scenarios with weak safety measures and relatively frequent unsafe states). However, with increasing difficulty (i.e., as safety measures get stronger and unsafe states become more rare), the risk-guided search increasingly outperforms the baseline.

7. Two-level Hazard Analysis

So far, the hazard analysis has been treated as a search problem which is solved through iterative sampling and execution of agent behaviors directly in simulation. This Chapter investigates an alternative approach which relies on a combination of formal verification and simulation. To that end, the EFA model which has been used previously to model the set of feasible agent behaviors is now extended to cover not only the agent, but the complete system of agent and SUT as well as the safety specification. This yields two system models on different abstraction levels: An abstract formal model consisting of EFAs, and a more detailed simulation model. The hazard analysis proceeds in two steps: First, the formal model is evaluated to obtain a set of potentially critical agent behaviors. Then, these agent behaviors are executed in the more detailed simulation model for closer examination.

This Chapter is structured as follows: Section 7.1 revisits the characteristics of formal verification and simulation and lays out the motivation for the two-level approach. Section 7.2 gives an overview of the two-level analysis. Sections 7.3 and 7.4 discusses the two levels in more detail. Finally, in Section 7.5, the approach is demonstrated and compared to the methods from the previous Chapters.

Publications Related to this Chapter
This Chapter is partially based on:

[93] Huck, Tom P., Selvaraj, Y., Cronrath, C., Ledermann, C., Fabian, M., Lennartson, B., and Kröger, T. "Hazard Analysis of Collaborative Automation Systems: A Two-layer Approach based on Supervisory Control and Simulation." 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023.

7.1. Motivation

In Section 3.5, the various advantages and disadvantages of different hazard analysis methods, including simulation-based and formal methods, have already been discussed in detail. For a better understanding of the motivation behind the two-level approach, the comparative advantages and disadvantages of formal and simulation-based methods are briefly revisited (see also [194] for further discussions). Table 7.1 summarizes the main differences between formal verification and simulation, as introduced in Section 3.5.

As illustrated by the "magic triangle" (Figure 3.4), model-based hazard analyses should achieve the following goals (while keeping computational complexity limited):

- The system model on which the analysis is conducted should be as detailed as possible, so that it accurately represents the real-world system which is analyzed.
- The analysis should be as exhaustive as possible. Ideally, a fully exhaustive exploration of all possible behaviors is desired, in order to provide a safety guarantee (assuming, of course, the model on which the analysis is performed correctly represents the real-world system).

These are two opposing goals: As the level of detail grows, so does the model's state space. With an increasing size of the state space, there comes a point where exhaustive state-space exploration is not feasible anymore from a computational point of view. Especially for complex systems, formal verification approaches which provide provable safety claims are not always tractable. On the other hand, with a detailed simulation model which is computationally expensive, one often cannot perform enough simulation runs for an exhaustive analysis [60].

In other words, there is a fundamental trade-off between the exhaustiveness of the search for hazards, and the accuracy with which a system can be modeled and subsequently analyzed. As seen from Table 7.1, formal verification and simulation-based testing can be interpreted as two opposite ends of this spectrum, with specific advantages and disadvantages which complement each other. Given the complementary nature of the two approaches, this section presents a two-level analysis to combine both. It

should be noted, however, that this is not the first work to explore a combination of formal verification and simulation-based testing in a robotics context. In particular, similar approaches have been proposed by Webster et al. [194] and Askarpour et al. [20]. Nevertheless, there are certain features which set the following approach apart from existing literature. Webster et al. [194] consider combined approaches of formal verification and simulation, but only in the sense of an overarching development process where human engineers act as intermediaries between the different verification and validation steps. Here, in contrast, the approach is integrated, with simulations being created automatically from the output of the formal model without the need for user interference. Askarpour et al. [20] use formal verification as a hazard analysis method and deploy the simulation primarily as a means of visualizing the findings of the formal verification. Here, however, both simulation and formal verification are treated as integral parts of the hazard analysis procedure. Furthermore, this work is fundamentally different in the way it utilizes the formal models: Rather than setting up the formal models for a fully-fledged analysis, they are used as a relatively simple, abstract and light-weight preprocessing layer for the more detailed simulations in the second step. Finally, the present work is the first to use Supervisory Control Theory (SCT) in a hazard analysis context. Compared to model checkers which are typically used in formal verification (see Section 3.2), the use of SCT in this context has several advantages which will discussed later in this Chapter.

Formal Verification		**Simulation-based Testing**
Formal system model (e.g. automaton)	$\leftrightarrow$	System represented by simulation model
Abstract model, limited accuracy	$\leftrightarrow$	Detailed model, high accuracy
(Relatively) small state space	$\leftrightarrow$	Large state space
(Near) exhaustive exploration of state space	$\leftrightarrow$	Exhaustive exploration infeasible
Can provide safety guarantees w.r.t. formal model	$\leftrightarrow$	Can only provide examples of unsafe behaviors, but no guarantee that a system is safe

Table 7.1.: Comparison of formal verification and simulation-based testing, (see section 3.5)

7.2. Overview of the Proposed Approach

The two-level analysis utilizes system models on two distinct abstraction levels. On the higher level of abstraction (referred to as *first level*), the system is abstracted as a Discrete-Event System (DES). The *second level* consists of a simulation model, as seen in the previous Chapters. DES abstractions have been used previously in Chapter 6 to model the feasible behaviors of the agent. Now, the DES model is extended to cover the complete system consisting of the agent and the SUT, as well as the interactions between agent and SUT. The safety specification is also abstracted as an automaton which uses the notion of marked and unmarked locations to differentiate between unsafe and safe states. Then, a supervisor synthesis (as introduced in Section 2.2.4) is performed to obtain a set of critical event sequences by which the agent may potentially cause the system to violate the safety specification. Each event sequence is executed by the agent in simulation multiple times with different configurations for the continuous agent parameters. During these simulations, the risk metric as well as the original (non-abstracted) safety specification are evaluated. This allows for a more detailed judgment as to how critical the synthesized agent behaviors actually are.

7.3. First Level: Synthesis of Critical Event Sequences

7.3.1. Abstraction of the Collaborative System

For a formal analysis, system and safety specification are abstracted as discrete event systems (DES). Like in the previous Chapter, Extended Finite Automata (EFA) are used as a DES modeling formalism. To that end, the agent EFA model which was introduced in the previous Chapter is extended by another EFA which represents the SUT. Together, the EFA models for agent and SUT form a DES abstraction of the collaborative system (see Definition 12). In this DES abstraction, interactions between agent and SUT can be modeled through shared events which appear in both EFA, and through shared variables which can be read and/or manipulated by both EFA (see Section 2.2.3). Hence, the agent's action space A and the

set of SUT-related events $\Sigma_{\mathcal{SUT}}$ as well as the variable sets $V_{\mathcal{A}}$ and $V_{\mathcal{SUT}}$ are generally not disjoint.

The joint event space $A \cup \Sigma_{\mathcal{SUT}}$ contains all events that can occur in the model of the collaborative system. Thus, $A \cup \Sigma_{\mathcal{SUT}}$ is the input alphabet of the synchronized composition.

Definition 12 - *Abstract System Model*
The collaborative system is modeled by two EFA $\mathcal{A}$ and $\mathcal{SUT}$ which represent the behavior of agent and SUT, respectively.

$$\mathcal{A} = \langle A, V_{\mathcal{A}}, L_{\mathcal{A}}, \rightarrow_{\mathcal{A}}, G_{\mathcal{A}}, M_{\mathcal{A}}, l_{0\mathcal{A}}, V_{0\mathcal{A}} \rangle$$
$$\mathcal{SUT} = \langle \Sigma_{\mathcal{SUT}}, V_{\mathcal{SUT}}, L_{\mathcal{SUT}}, \rightarrow_{\mathcal{SUT}}, G_{\mathcal{SUT}}, M_{\mathcal{SUT}}, l_{0\mathcal{SUT}}, V_{0\mathcal{SUT}} \rangle$$

Where A denotes the action space of the agent and Σ_{SUT} denotes the set of events that can appear in the SUT. Further elements of the automata are defined according to Definition 3. The behavior of the complete system of both agent and SUT is given by the synchronous composition of the two EFA:

$$\mathcal{A} \parallel \mathcal{SUT}$$

Within this framework, an event sequence

$$(e_i) = (e_1, e_2, \dots), \; e_i \in A \cup \Sigma_{\mathcal{SUT}}$$

denotes one particular behavior of the system. The language of the synchronized composition $L(\mathcal{A} \parallel \mathcal{SUT})$ contains all feasible event sequences and thus represents the set of all possible behaviors that can occur in the collaborative system.

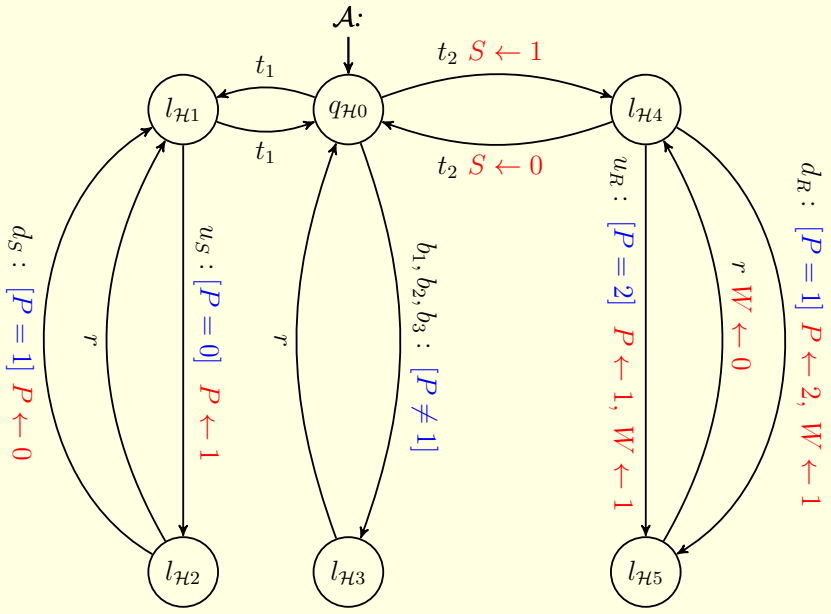
For reasons of brevity, the concrete modeling procedure for obtaining the EFAs is not explained here. Instead, the EFA models are assumed as given. Appendix C gives a detailed step-by-step modeling guideline explaining how to obtain the EFA models from scratch.

The following Example 10 (figures and descriptions based on [93]) illustrates how a system model in the form of EFAs could look like. Note that the individual EFA models may appear smaller and less complex than the actual behavior they encode would suggest. This is due to two reasons: First, states are not only encoded in the form of locations, but also in the form of variables, which makes for a more compact modeling. Second,

the system behavior is given by the synchronous composition $\mathcal{A} \parallel \mathcal{SUT}$, whose state space is the cartesian product of the state-spaces of the individual EFA. Fortunately, however, it is not necessary to construct the synchronous composition explicitly, as this can be done on-the-fly by efficient DES tools such as SUPREMICA [135]. The ability to construct complex models through composition of simple modules makes EFA a powerful modeling tool.

Example 10 ***Abstract System Model***

Recall the HRC system from example 7. The behavior of the agent (i.e., the human worker) in this system can be modeled by the following EFA:

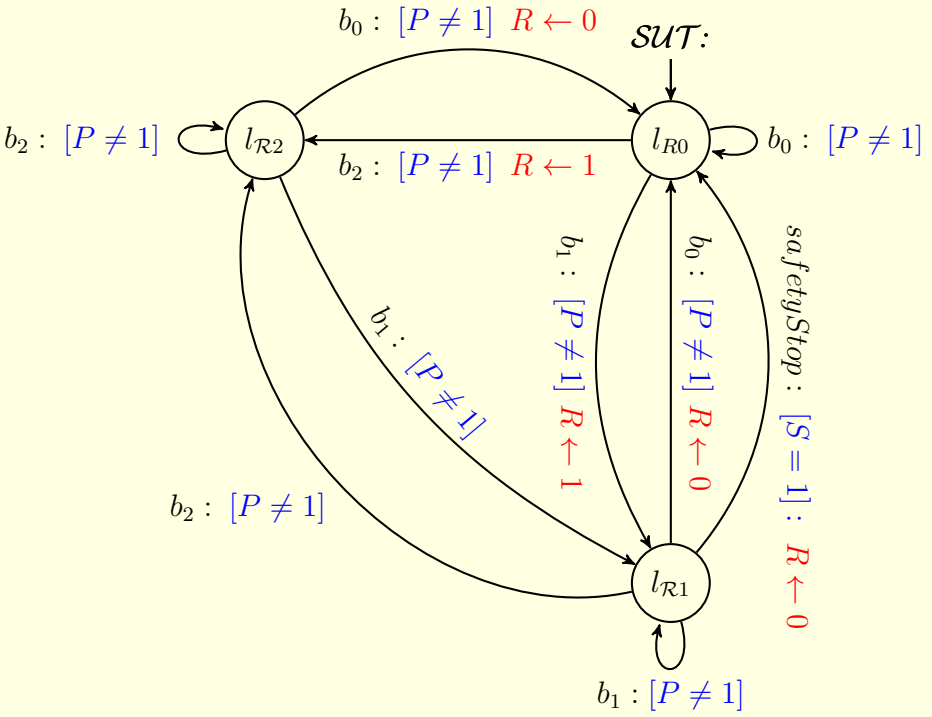


Where the events (i.e., the action space of the agent) are defined as follows:

t_1	*Walk between area A and B*
t_2	*Walk between area A and C*
u_S (d_S)	*Pick up (putting down) part at the storage table*
b_0	*press button to stop robot*
b_1	*press button to start robot*
b_2	*press button to start robot in safety override mode*
r	*retract hands (after reaching motion or button pressing)*

Furthermore, the discrete integer variables P, W, and S are introduced. $P \in \{0, 1, 2\}$ denotes the position of the workpiece (0: at storage, 1: in worker's hands, 2: at robot station). $W \in \{0, 1\}$ denotes if the agent currently occupies the shared human-robot workspace (0: not occupied, 1: occupied). $S \in \{0, 1\}$ denotes if the agent currently occupies the laser scanner zone (0: not occupied, 1: occupied). Observe that some events require guard statements to be fulfilled before the transition is enabled. Through these guard statements, constraints on the agent's

behavior are enforced. For instance, the guard is $P = 1$ needs to hold in order to put down a workpiece at the robot station d_R, because the worker must first be in possession of the part to put it down.
The model of the SUT (i.e., the robot system) is shown below:



In this model, each operation mode of the robot is represented by a location $l_{\mathcal{R},i}$ (idle: $l_{\mathcal{R}0}$, working: $l_{\mathcal{R}1}$, working in safety override mode: $l_{\mathcal{R}2}$). The agent changes the operation mode by pressing the buttons (events b_0, b_1, and b_2). Note that b_0, b_1, b_2 are shared events: While they are performed by the agent, they appear both in the agent's and in the SUT's automata model, as the agent's behavior also changes the state of the SUT. As discussed in Section 2.2.3, shared events are one of the ways how interactions between system components can be modeled in EFAs (the other way being shared variables).
Additionally, the SUT features an event $(safetyStop)$ which represents the safety stop triggered by the laser scanner. This event transfers the robot from working into idle mode. The safety stop is only enabled if the robot is running in normal operation mode and the laser scanner is occupied (as denoted by the guard statement $S = 1$). Note that $(safetyStop)$ is not available in safety override mode (i.e., in $q_{\mathcal{R}2}$), as the robot's safety function is deactivated in this mode of operation. Furthermore, the variable $R \in \{0, 1\}$ is introduced. It denotes if the robot is currently idle ($R = 0$) or active ($R = 1$).

7.3.2. Abstraction of the Safety Specification

The original safety specification spec as introduced in Definition 4 is defined over the simulation state-space S. Since the simulation state-space is more detailed than the state-space of the abstracted system model from Definition 12, it is generally not possible to evaluate the original safety specification spec on the abstracted models (for instance, if spec uses collision forces as a criterion, but the abstracted system is not sufficiently detailed to estimate collision forces, then it is not possible to evaluate spec on the abstracted model). Thus, an abstraction of the safety specification is needed which, analogously to the original safety specification, maps the state-space of the abstracted model to a binary safety decision. This can be achieved with a third automaton that uses the notion of marked locations to differentiate between safe and unsafe states.

Definition 13 ***Abstract safety specification***
The abstract safety specification is given by an EFA named $\mathcal{SP}$:

$$\mathcal{SP} = \langle \Sigma_{\mathcal{SP}}, V_{\mathcal{SP}}, L_{\mathcal{SP}}, \rightarrow_{\mathcal{SP}}, G_{\mathcal{SP}}, l_{0\mathcal{SP}}, V_{0\mathcal{SP}}, L^m \rangle \tag{7.1}$$

where the initial location l_0 is unmarked and the remaining locations are marked:

$$L_{\mathcal{SP}} = \{l_{0\mathcal{SP}}\} \cup L^m$$

The initial unmarked location $l_{0\mathcal{SP}}$ represents a safe state and the marked locations represent unsafe states. The event space Σ_{SP} represents unsafe events that can occur in the system (e.g., a collision). These unsafe events transfer the EFA from the initial (safe) state to a marked (unsafe) state. The set of variables is a subset of the variable space of the system model:

$$V_{\mathcal{SP}} \subseteq V_{\mathcal{A}} \times V_{\mathcal{SP}}$$

The guards $G_{\mathcal{SP}}$ represent the conditions under which the unsafe events can occur.

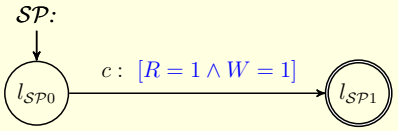
Intuitively, $\mathcal{SP}$ from Def. 13 can be thought of as an observer which observes the variables of the system model and evaluates the variables with respect safety criteria that are encoded in the guards $G_{\mathcal{SP}}$. If a safety criterion is violated, the corresponding guard statement is fulfilled, and $\mathcal{SP}$ changes location from the initial unmarked location to a marked location.

This ensures that $\mathcal{SP}$ will always enter a marked state if a safety criterion is violated, while remaining in an unmarked state otherwise. $\mathcal{SP}$ should have no shared events with the system model EFAs: $\Sigma_{SP} \cap (A \cup \Sigma_{SUT}) = \emptyset$, and $\mathcal{SP}$ should have no update rules for variables (i.e., it can observe, but not change the values of variables). Thereby, it is ensured that $\mathcal{SP}$ can only observe the behavior of the system model, but cannot influence its behavior. If this is not adhered to when modeling $\mathcal{SP}$, one may inadvertently impose false restrictions on the system behavior and thus exclude potentially safety-critical behaviors from the analysis.

Encoding the safety specification in form of an automaton instead of simply specifying a logical formula over the variable space may seem unnecessarily complicated at first glance. However, having both the system model and the safety specification represented as an EFA will prove useful in Section 7.3.4, where a synthesis technique will be applied which expects both system model and safety specification to be given in that form.

The following Example 11 illustrates this principle with a corresponding safety specification for the models from the previous Example 10 (see also [93]).

Example 11 ***Abstract Safety Specification***
The EFA model $\mathcal{SP}$, which represents the safety specification, is shown below:



This EFA only has two locations. $l_{\mathcal{SP}0}$ represents a safe state and $l_{\mathcal{SP}1}$ an unsafe state, that is, a human-robot collision. Note that the $l_{\mathcal{SP}1}$ is marked, but only reachable through event c (collision). For the event $collision$ to occur, the following guard statement has to be fulfilled: $R = 1 \wedge W = 1$. This guard statement expresses the safety criterion: A state is considered to be a collision if the agent is in the collaborative workspace while the robot is working.

7.3.3. Overapproximation of Unsafe Behaviors

Abstracting system and safety specification as DES decreases state-space complexity and enables enables an exhaustive identification of hazardous behaviors. However, as discussed in Section 7.1, abstraction comes at the cost of limited detail. The loss of detail can lead to inaccuracies of

the abstract model compared to the simulation model. These inaccuracies can manifest themselves as follows:

- The abstract model classifies an event sequence as safe, although it is actually unsafe when analyzed in simulation. In the following, such an occurrence shall be called *missed alarm*, since the abstract model fails to highlight a potential hazard.
- The abstract model classifies an event sequence as unsafe, although it is actually safe when analyzed in simulation. This shall be called *false alarm*, since the abstract model falsely claims the presence of a hazard.

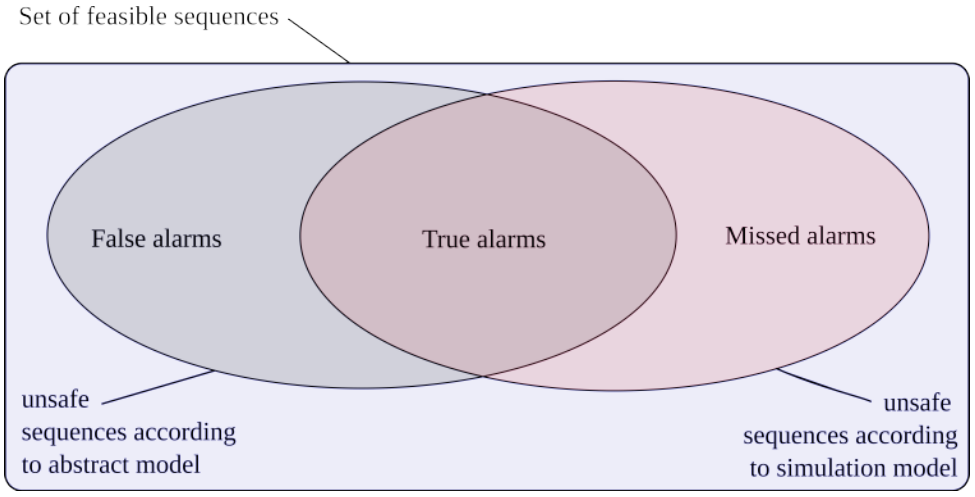


Figure 7.1.: Relationship between false, true, and missed alarms. (Author's own figure.)

The relationship between true, false, and missed alarms is illustrated in Figure 7.1. From a hazard analysis perspective, missed alarms are clearly more critical than false alarms. While false alarms can be filtered out easily by a more detailed simulation-based analysis, a missed alarm would mean that a critical event sequence is discarded before simulation, causing the respective hazard to remain unnoticed. Therefore, the abstract model should be designed in such a way that it overapproximates the set of unsafe behaviors (i.e., missed alarms are avoided at the cost of having more false alarms). To achieve this, certain modeling principles should be obeyed. These are discussed in the following.

Conservative abstraction of safety specifications. The safety specification should be abstracted conservatively, so that the abstract safety specification $\mathcal{SP}$ is more conservative than the original safety specification spec. Typically, the state-space of the abstract model contains less

information than the state-space of the original simulation model. Thus, the abstract safety specification has less detailed information available to decide whether a given state is safe or unsafe. To arrive at a conservative safety specification, safety-critical information that is missing should be assumed to take a worst-case values. For instance, if the original safety specification states that a collision should not exceed certain collision force values, but the abstracted model does not contain collision force information, then all collisions should be assumed to exceed the limits, and the abstracted safety specification should classify all collision states as unsafe (as seen e.g. in Example 11).

Overapproximation of spatial occupancy. Since EFA only allow for discrete states, the shared workspace of agent and SUT needs to be discretized. Variables and update rules need to be introduced to track the current occupancy of agent and SUT in each state (as seen e.g. with the variabls R and W in Example 10). The discretzation should be designed in such a way that the spatial occupancy of agent and SUT is overapproximated, which leads to a more conservative detection of collision hazards.

Non-deterministic timing. EFA feature no quantiative notion of time. This means that EFA models consider the *order* of events (e.g., "event e_2 occurs after event e_1"), but *not their timing* ("e.g., "event e_2 occurs 1 second after event e_1"). This can lead to issues if the exact timing behavior is important for safety. In Example 10, for instance, the laser scanner needs to stop the robot before the human can reach the robot's workspace. This depends on the time that the human needs to approach the robot as well as the time that is required for the laser scanner to detect the human and stop the robot. Of course, the EFA models could be augmented to feature a quantitative notions of time (e.g., by introducing clock variables). However, this would increase model complexity and require users to make assumptions about the durations of events which is error-prone. A more elegant solution is to avoid timing constraints by keeping the timing of critical events non-deterministic. When synthesizing critical event sequences, this forces the synthesis algorithm to consider all possible interleavings of events, and not just one sequence with a prescribed timing. Of course, this may also include event sequences which are potentially unsafe, but not actually feasible in terms of timing. However, these sequences are filtered out in the second layer of the analysis, where the simulation accurately models actual timing behavior of the systems. This principle can be seen, for instance, in the model for the SUT from Example 10. Here, the robot is supposed to enter a safety stop as the human enters the laser scanner zone. However,

instead of assuming that the safety stop occurs immediately upon entry in the safety zone, the safety stop is modeled as a separate event which is associated with the guard $[S = 1]$. In other words, the model merely states that entering the laser scanner zone is a *precondition* for the safety stop, but it does not assume that the safety stop occurs simultaneously or with any other specific timing.

7.3.4. Synthesis of Critical Event Sequences

A *supervisor synthesis* is performed to extract a set of critical event sequences from the abstract models. As explained in Section 2.2.4, supervisor synthesis yields a subset of all possible behaviors of a system. The synthesized subset contains exactly those behaviors that comply to a given specification. By performing a supervisor synthesis for the plant $\mathcal{SUT} \parallel \mathcal{A}$ with respect to the specification $\mathcal{SP}$, one can thus obtain the set of all possible event sequences that result in unsafe states. Recall that the supervisor synthesis aims to create non-blocking behaviors, that is, event sequences which result in marked states.

Let $\mathcal{K}$ denote the synthesized supervisor, and $L(\mathcal{K})$ the language of the synthesized supervisor. Then, $L(\mathcal{K})$ contains exactly those event sequences which lead to marked states, while those which do not result in a marked state are removed. Since $\mathcal{SP}$ is designed in such a way that unsafe states correspond to marked states, the synthesis yields a set of *unsafe* event sequences. Since the supervisor synthesis is also minimally restrictive, the synthesized set of unsafe event sequences is *exhaustive*, that is, it contains *all* possible behaviors that result in unsafe states, and not just some (see Section 2.2.4). Herein lies one of the main advantages of SCT compared to other formal verification methods. Typical formal verification approaches use so-called *model checkers*, such as *SPIN* [85]. When a violation of safety specifications is found, model checkers return a single *error trace*, that is, an exemplary sequence of events which leads to an unsafe state. It is then up to the user to correct the system error that has led to the unsafe state, adapt the model accordingly, and re-run the model checker. While such an approach is feasible in conventional model checking applications where false alarms are assumed to be relatively rare, it is not suitable for the present use-case. The reason for this is that the overapproximative nature of the first abstraction level can lead to a number of false alarms which is so large that manually checking the resulting error traces is not feasible.

The synthesis approach, in contrast, yields an exhaustive set of event sequences that violate the safety specification without requiring the user to iteratively inspect each sequence, adapt the model, and re-run the analysis. A more detailed analysis of the error traces is then done automatically on the second level.

7.4. Second Level: Simulation-based Evaluation of Synthesized Event Sequences

In the next analysis step, the synthesized event sequences are executed in simulation for a more detailed analysis. At this point, it is important to emphasize the following aspect, which is a key difference compared to other works such as [20]: The second step of analysis does not strictly recreate the complete event sequences found in the formal analysis. Instead, it only recreates the behaviors of the *agent*, while the behavior of the SUT emerges as a reaction of the agent behavior in simulation. In other words, the formal analysis can be seen as a preprocessing step which identifies potentially interesting agent behaviors a-priori using prior knowledge. The SUT is then further scrutinized by exposing it to the synthesized agent behaviors in simulation and observing it's response, rather than using the simulation merely as a tool for visualization. Therefore, only a subset of all events, namely those associated with the agent's behavior, are used as simulation inputs. In contrast, events associated to the SUT are *emergent*, that is, they arise from within the simulation as a consequence of the SUT's behavior. Therefore, in each of the previously generated set of unsafe event sequences, those events related to the SUT are discarded, and the remaining sequence of agent-related events are used as inputs to the simulation. Furthermore, it is only possible to simulate finite sequences of actions, whereas the language of the supervisor may contain infinite sequences. Thus, it is necessary to clip the action sequences to a maximum length n.

The set of action sequences which is thus created shall be denoted by $\tilde{L}^n_{\mathcal{K}}$. As explained above, $\tilde{L}^n_{\mathcal{K}}$ is obtained from $L_{\mathcal{K}}$ by taking the event sequences $(e_i) \in L_{\mathcal{K}}$ and removing all events which are not part of the agent's action space. Example (adapted from [93]) illustrates this concept. The remaining set of action sequences $\tilde{L}(\mathcal{K})^n$ is then simulated. During each simulation run, the risk metric and the original (i.e., non-abstracted) safety specification spec are evaluated to assess whether the event sequence is indeed safety-critical, or whether it is a "false alarm", that is, whether it was falsely deemed critical on the first level of the analysis.

Example 12 - *Removal of non agent-related events.*
Suppose that the language $L_{\mathcal{K}}$ of the synthesized supervisor contains the following event sequence:

$$(\text{activateRobot}, \text{approachRobot}, \text{robotStops}, \text{enterWorkspace})$$

Here, the actions $activateRobot$, $approachRobot$ and enterWorkspace *are from the agent's action space A, whereas the event* robotStops *is from $\Sigma_{\mathcal{SUT}}$, that is, from the event space of the SUT. The non-agent related event is removed and only the following action sequence is used as input to the simulation:*

$$(\text{activateRobot}, \text{approachRobot}, \text{enterWorkspace})$$

The event robotStops *is not prescribed as a simulation input, but emerges internally in the simulation as a consequence of the behavior encoded in the robot's simulation model.*

At this point, the first analysis level has already narrowed down the search space to an (ideally relatively small) set of critical action sequences. Therefore, it is assumed that – in contrast to the simulation-only approach – an exhaustive simulation of all remaining sequences is possible.

Note that so far, only events relating to the agent's discrete action space have been considered. Continuous parameters have only be considered on the second level, as the DES abstraction on the first level does not allow continuous variables. However, if the computational budget allows for multiple simulation runs per event sequence, then it is also possible to consider different continuous parameters of the agent's behavior. To that end, each of the synthesized event sequences is simulated multiple times with different parameter configurations. The selection of parameters can be done through random sampling or heuristically. The simulation budget can be uniformly distributed over the set of critical event sequences so that each sequence is simulated equally often.

In the following, an approach with heuristic parameter sampling is presented. Suppose there are m continuous parameters in the agent's behavior space, and each parameter θ_j is subject to parameter limits $\theta_{j,min}$ and $\theta_{j,max}$. Then, the set of feasible parameter combinations is given by an m-dimensional cube with 2^m vertices (as shown in Figure 7.2 for the case $m = 3$). The heuristic sampling approach relies on two assumptions: First, it is assumed that parameter configurations in the corners of the cube are more likely to lead to unsafe states, since these are associated with more

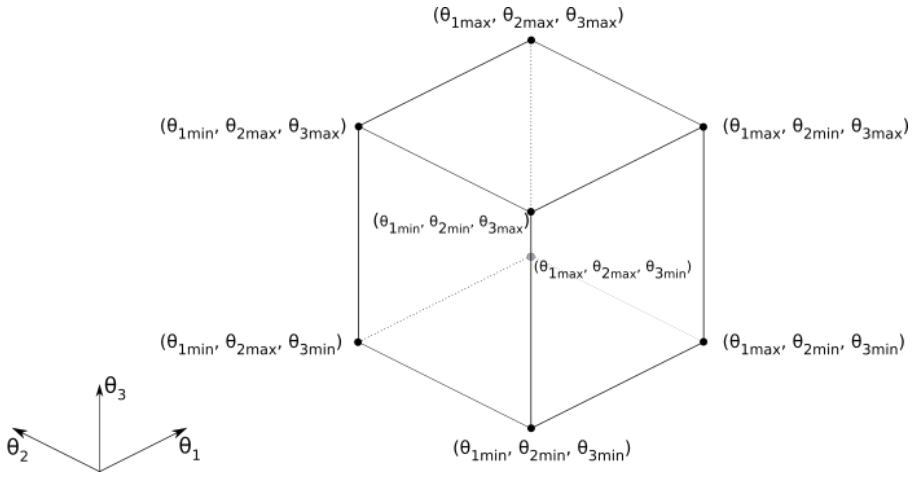


Figure 7.2.: Parameter space for $m = 3$. (Author's own figure.)

extreme behaviors which are farther away from the nominal and expected agent behavior. Second, it is assumed that edges with more maximum values are more safety-critical than those with minimum values. This is especially the case since continuous parameters are frequently associated with velocities of the agent's movement, and high velocities make it more difficult for the safety measures of the SUT to respond. (Of course, this is an assumption which may not always hold. However, it can be argued that it is a reasonable assumption in the absence of further knowledge of the parameter's effects). Therefore, the vertices are prioritized as follows:

$$\begin{aligned} v_1 &= (\theta_{1,max}, \theta_{2,max}, \ldots, \theta_{m-1,max}, \theta_{m,max}) \\ v_2 &= (\theta_{1,max}, \theta_{2,max}, \ldots, \theta_{m-1,max}, \theta_{m,min}) \\ &\vdots \\ v_{(2^m)} &= (\theta_{1,min}, \theta_{2,min}, \ldots, \theta_{m-1,min}, \theta_{m,min}) \end{aligned}$$

Assume that there is a fixed simulation budget of N simulation runs. Then, the number of parameter combinations which can be calculated per action sequence is given by

$$\tilde{N} = \left\lfloor \frac{N}{|\tilde{L}^n(\mathcal{K})|} \right\rfloor$$

If $\tilde{N} < 2^m$, only the first $\tilde{N}$ parameter combinations $v_1 \ldots v_{\tilde{N}}$ of the list are simulated. If $\tilde{N} \geq 2^m$, all parameter combinations from the list are simulated, and in the case of $\tilde{N} > 2^m$, the additional simulation budget is used for randomly sampled parameter combinations from within the cube.

Alternatively, the simulation budget can be prioritized according to the risk metric. This can be achieved by simulating each sequence once to ob-

tain an initial risk estimate, and then the distributing the remaining available simulation runs with a prioritization scheme that assigns more simulation runs to those sequence with a high initial risk estimate (the assumption behind the latter approach is that sequences with a high initial risk estimate are more likely to reach an unsafe state when the parameters are varied).

7.5. Experiments

7.5.1. Goal and Methodology

A series of experiments was conducted to investigate the feasibility and the performance of the two-level approach, and to compare it to the result of random and the risk-guided search from the previous Chapter[1]. Note that the latter two do not rely on any synthesis algorithm, but only on simulation to identify unsafe behaviors. They are therefore called "simulation-only" methods in the following. Concerning the comparison between the simulation-only methods and two-level approach, the following questions are of interest:

- Does the two-level approach find more unsafe behaviors than the simulation-only methods?
- Does the two-level approach miss any hazards which are found by the simulation-only approach (missed alarms)?
- Does the abstract model raise any "false alarms" which are then not confirmed to be hazardous in simulation? And if yes, how frequent are these false alarms?

Regarding the methodology, scenarios, and evaluation criteria, the experiments in this Chapter are identical to those performed in Chapter 6 (i.e., scenarios 6-A to 6-C). The difference is that the in addition to the simulation model, each scenarios is also modeled in the form of EFA in order to deploy the two-level approach is deployed additionally to the random and risk-guided search. Performance criteria for comparison remain identical to the previous Chapter. Implementation, execution, and results of the experiments are described in the following. As in Chapter 6, the reader is referred to the original publication [93] and to the appendix for further details.

[1] Note that these are the same experiments as presented in Chapter six, but with the addition of the two-level hazard analysis.

7.5.2. Implementation

Modeling

The simulation models of the test scenarios are identical to the those described in Section 6.5. For the agent models, the existing EFA models from the experiments in Section 6.5 are used. In addition, EFA models for SUT and safety specification are created. using the software tool SUPREMICA [135]. For scenario A, a model with 90 states and 182 transitions[2] was created. For scenario B and C, a model with a model with 161 states and 367 transitions was created (note that since scenarios B and C are based on the same simulation model, they also share the same EFA model). For reasons of brevity, the EFA models are not discussed in detail here. A detailed description of the newly created models is given in Appendix A.3 and a step-by-step explanation of the modeling procedure is given in Appendix C.

Synthesis and Simulation

In each scenario, a supervisor synthesis is performed with *Supremica* [135] (details of the synthesis procedure are given in Appedix C.2 and C.3). The next step is to extract the agent's event sequences from the supervisor. *Supremica* represents the synthesized supervisor as a finite automaton whose language contains all event sequences that comply with the specification. The automaton is exported as an XML file and parsed to extract the event sequences that are part of the accepted language of the synthesized supervisor. However, as explained above, only the events that are part of the agent's action space are used as simulation input. Thus, the events related to the SUT's behavior are discarded in the parsing procedure. Although the number of states of the supervisor is finite, the lanugage of the automaton may contain event sequences of arbitrary length, possibly even of infinite length of the sequences contain loops. Since it is only possible to simulate sequences of finite length, the length of the action sequences needs to be limited. In this case, only sequences which do not contain more than a finite number of n agent actions events are extracted ($n = 12$ for scenario A and $n = 10$ for scenarios B and C). This resulted in 22

[2]The number of states and transitions given here is the number of states of the synchronous composition of all EFA models, including Agent, SUT, ans safety specification.

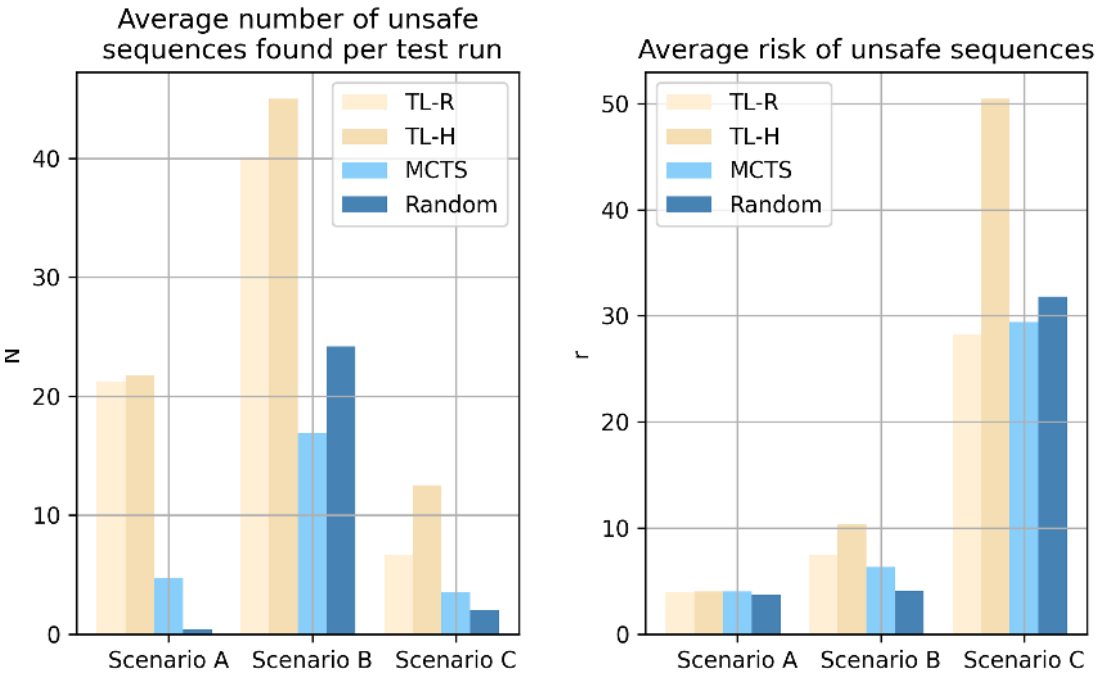
potentially critical sequences for scenario A, and 83 potentially critical sequences for scenarios B and C. The resulting event sequences are then fed into the simulator and evaluated with different continuous parameter combinations as explained in Section 7.4. To evaluate the effectiveness of the heuristic parameter sampling approach, a baseline which relies on purely randomly sampled parameters is also deployed. For each configuration of scenario and sampling method, ten test runs with different random seeds are performed in order to limit the influence of statistical outliers.

7.5.3. Results

Search space reduction. In all three scenarios, evaluating the first level before performing simulation led to a significant reduction of the search space compared to simulation-only methods. In scenarios B and C, evaluating the abstract model through supervisor synthesis returned 83 potentially critical sequences out of 366 feasible sequences. In scenario A, the reduction was even greater. Here, the abstract model returned 22 potentially critical sequences out of 17441 feasible sequences. This indicates that even relatively simple abstract models, such as the ones used here, can already lead to a significant reduction of the search space.

Search performance. The search performance is measured in terms of the number of identified unsafe behaviors and in terms of the average risk values associated to these behaviors. Figure 7.3 shows the results in comparison to the the results of random search and MCTS from the previous Chapter. Note that the experiments for the two-level approach are based on the same scenarios and use the same computational budget as in the previous Chapter. Thus, a direct comparison to the results from the previous Chapter is possible.

Figure 7.3a shows how many unsafe event sequences were found by the respective methods (note that the figure only includes true alarms, while false alarms of the two-layer method are excluded to make the comparison fair). The two-layer approach significantly outperforms both MCTS and random sampling in all three scenarios. Between the two variants of two layer approach, the one with parameter sampling heuristics (TL-H) performs slightly better than the one with randomly sampled parameters (TL-R). Figure 7.3b shows the performance in terms of the average risk score. Here, TL-H clearly outperforms the simulation-only methods, while the TL-R version slightly underperforms them.



(a) Number of hazardous sequences **(b)** Average risk

Figure 7.3.: Performance of two-level analysis with randomly sampled parameters (TL-R), two-level analysis with parameter sampling heuristics (TL-H), Random Sampling, and MCTS in comparison. (Author's own figure.)

Disagreements between abstract model and simulation. Furthermore, it is interesting to investigate if there were any false or missed alarms. However, determining the true number of false and missed alarms would require a "ground truth", that is, a definitive knowledge of which and how many unsafe event sequences actually exist. To achieve this, each feasible event sequence would have to be simulated exhaustively with all possible parameter combinations, which is computationally infeasible. However, it is still possible to gain some insights by investigating disagreements between ETA models and simulation models in the test runs that were performed. To that end, the following sets of sequences are obtained from the results:

- $L^n(\mathcal{K})$: The set of sequences that are unsafe according to the abstract model (see Section 7.4).

- X_T: The set of sequences that are unsafe according to the two-layer analysis (i.e., sequences which are in $L^n(\mathcal{K})$ *and* are confirmed to be

unsafe through simulation). These are accumulated over all test runs for the respective scenario.

- X_S: The set of sequences that are unsafe according to the simulation-only methods (i.e., random search and MCTS). Again, these are accumulated over all test runs for the respective scenario.
- $L^n(\mathcal{K})\backslash X_T$ contains all sequences which are unsafe according to the abstract model, but where not unsafe states have been found in simulations. These sequences indicate false alarms[3].
- $X_S\backslash L^n(\mathcal{K})$ contains all sequences which were found to be unsafe in simulation, but not in the formal model. These sequences indicate missed alarms.

Scenario	$\lvert L^n(\mathcal{K})\rvert$ (unsafe sequences in abstract model)	$\lvert X_T\rvert$ (true alarms)	$\lvert L^n(\mathcal{K})\backslash X_T\rvert$ (false alarms)	$\lvert X_S\backslash L^n(\mathcal{K})\rvert$ (missed alarms)
A	22	21	0	1
B	83	45	38	11
C	83	13	70	0

Table 7.2.: Disagreements between formal model and simulation

The results are shown in Table 7.2. In scenario A, the abstract model found 22 unsafe sequences, out of which all were confirmed to be unsafe in simulation (i.e., no false alarms). However, there was one missed alarm. In scenario B, the abstract model found 83 potentially unsafe event sequences. 45 of these were confirmed to be unsafe in simulation, while the remaining 38 were false alarms. There were eleven missed alarms. In scenario C, the interactions between agent and SUT are, in principle, the same as in scenario B. Thus, the abstract model again finds the same 83 potentially unsafe sequences as in scenario B. Yet, the main difference is that scenario C features stronger safety measures in form of an additional light curtain (see Section 6.5). Due to this, many of the sequences found to be unsafe in the abstract model now become safe, including the eleven sequences which constituted missed alarms in scenario B. Yet, the since abstract model is does only capture the order of events but not their exact timing, (see Section

[3] note that technically, sequences in this set can result from two conditions: (i) from a false alarm by the abstract model or (ii) from a true alarms where an unsafe state is possible, but the simulation was not able to expose with the continuous parameters that were sample. For reasons explained above, it is not possible to know which of these conditions are the case. For reasons of brevity, however, these sequences will be referred to simply as false alarms.

7.3.1), it needs to assume that sequences which invoke the light curtain are unsafe. This is reflected by the large number of 70 false alarms compared to only 13 true alarms.

Investigation of missed alarms. As stated above, there were some missed alarms in scenarios A and B. This indicates that the abstract model in this case did not truly over-approximate the set of unsafe sequences despite the measures that were taken to achieve such an over-approximation. A closer examination of the simulation sequences revealed unforeseen interactions between agent and SUT in simulation which were not modeled in the abstract model. One such case is shown in Fig. 7.4. Here, the worker accidentally presses the button with its elbow which reaching for the workpiece, which results in an activation of the robot and a potential collision with the workers head or hands. The modeling techniques discussed in Sec. 7.3.3 are unable to avoid this issue, since it did neither result from inaccuracies in the safety specification, nor spatial occupancies, nor timing, but rather from an entirely unforeseen type of interaction which was not considered at all in the model. The simulation-only methods, on the other hand, make fewer assumptions about potential hazards and were thus able to identify this hazardous behavior.

7.5.4. Discussion

The main advantage of the two-level approach is that it leverages an abstraction of the system which is exhaustively explored to identify potentially interesting (i.e., safety-critical) agent behaviors in a first analysis step. This restricts the search space for the following simulation-based analysis and makes the search more effective. The increased effectiveness is demonstrated by the fact that in all testing scenarios, the two-level approach found, on average, significantly more unsafe sequences than MCTS or random sampling.

However, the advantage comes at the cost of an increased modeling effort. Where the purely-simulation based approaches only need one DES model to determine the feasible agent sequences, the two-level approach requires a DES abstraction of the complete system consisting of agent, SUT, and safety specification. On the other hand, the experiments have demonstrated that even relatively simple models can already reduce search complexity significantly, so performance gains compared to simulation-only methods were already achieved with relatively small modeling effort.

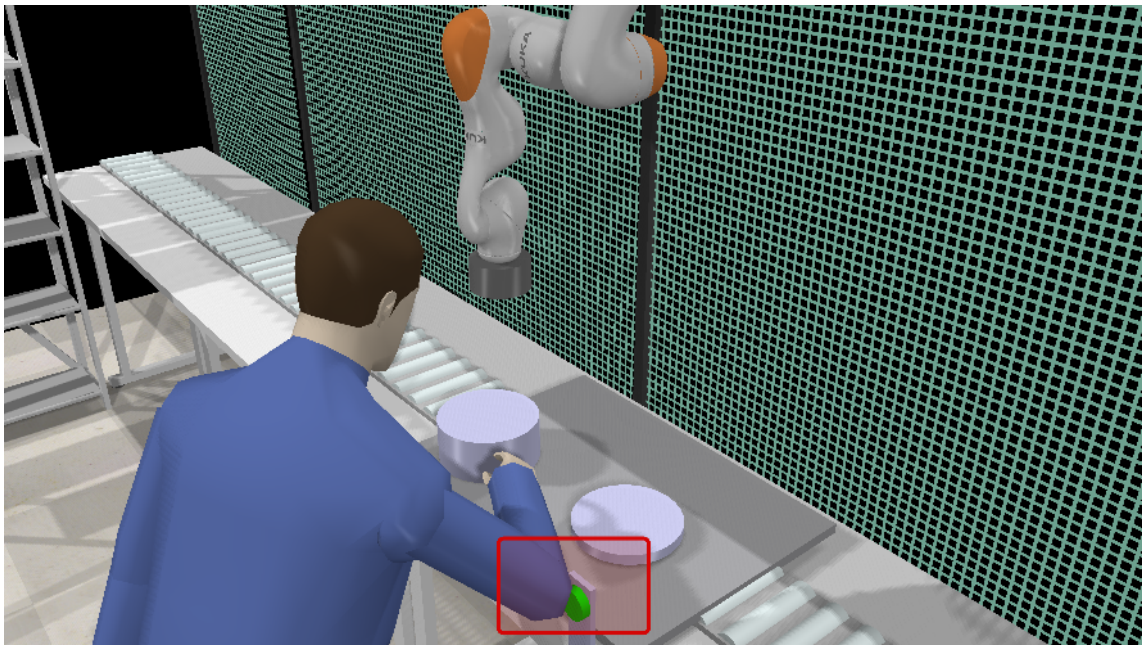


Figure 7.4.: Example of an unsafe behavior which was not detected in the abstract model: The human worker attempts to reach for the workpiece and inadvertently presses the robot activation button with the elbow (as highlighted by the red box). This activates the robot and leads to a potential collision with the human head or hands. The abstract model was unable to identify this unsafe behavior, as the depicted effect was not captured in the model. (Author's own figure.)

The limited accuracy of the abstract models can be compensated by adopting a more conservative modeling approach. While this can lead to a large number of false alarms (as seen in the results from Scenario C), the overall search performance in the experiments was still higher than that of the simulation-only methods (see Fig. 6.2), even when adjusted for false alarms. Yet, the experiments have also highlighted that despite the conservative modeling approach, missed alarms cannot be fully avoided if there are certain unforeseen interactions between agent and SUT which are not captured in the abstract model (see Scenario B, Fig. 7.4).

One the other hand, one should also keep in mind that this issue is not specific to the two-layer approach. Missed alarms also happen in purely-simulation based approaches. In fact, even when taking the missed and alarms into account, the two-level approach *still* outperforms both MCTS and random sampling.

A promising direction for further improvement would be to design a two-level approach where the second level uses the sequences obtained from the first level as a starting point, but is not restricted to simulating only these sequences. Instead, the second level should also be able to deviate and explore other sequences, even if they have not been labeled as unsafe by the formal analysis. Such an approach would mitigate the problem of missed alarms. A possible way to implement such an approach would be a to use a MCTS algorithm which does not start with an empty search tree, but with an initial search tree that is pre-populated with the sequences obtained on the first analysis level. Another area of improvement would be to employ other search heuristics on the level of continuous parameters such as Simulated Annealing or the Nelder-Mead Algorithm, see Section 5.2.3.

Finally, it should also be noted that in some cases, the applicability of the two-level approach is limited because determining suitable modeling abstractions is difficult if the behavior of the SUT is influenced by black-box components whose internal behavior is not fully understood (e.g., artificial neural networks or compiled third-party software). In such cases, a simulation-only approach such as MCTS or random sampling may be more suitable.

7.6. Chapter Summary

This Chapter has introduced a two-level analysis as an alternative to purely simulation-based risk-guided search. The two-level analysis first uses an abstract model of the system to identify potentially hazardous event sequences, which are then scrutinized further in simulation. To that end, the previously introduced DES model of the agent is extended by DES abstractions of SUT and safety specification. Supervisory Control Theory (SCT) is leveraged to extract a complete set of potentially critical event sequences which serve as input to the simulation. In the experiments, the two-level approach outperformed purely simulation-based approaches in terms of the number of unsafe behaviors that have been found. However, the two-level approach also has some drawbacks. These include the increased modeling effort as well as the possibility of false alarms (i.e., the formal model indicates hazards where there are none) or missed alarms (i.e., an unsafe event sequence is not simulated because the formal model misjudges it as safe).

8. Discussion and Outlook

THIS CHAPTER IS DEVOTED TO A DISCUSSION of the presented work and an outlook on future work. Section 8.1 discusses the contributions of this work. Section 8.2 critically examines the limitations of the proposed methods. Section 8.3 presents potential avenues for future research, while Section 8.4 explores the applicability of the developed methods to other domains beyond industrial robotics. Lastly, Section 8.6 concludes the book with some final remarks.

8.1. Contributions

Below, the contributions of this work are discussed. To that end, the three questions from Section 1.1 are revisited. For each question, it is briefly summarized what new insights have been gained in this work.

Q1: How to analyze increasingly complex robot systems for potential hazards?

In the introduction to this book, it was argued that hazard analysis of robotic systems is becoming increasingly challenging, especially due to the fact that complexity of these systems is increasing. The literature research and expert surveys presented in Section 3 have shown that there are currently four predominant approaches to identify hazards in safety-critical systems:

- Informal methods (e.g., brainstorming, use of expert knowledge)
- Semi-formal methods (e.g., HAZOP, STPA, FMEA)
- Formal verification
- Test-based methods (especially simulation-based testing)

While the first three approaches are relatively well-established, their applicability to complex systems is limited because they are so-called *white-box methods*. That is, they rely on an in-depth understanding of the system's internal structure and functionality, either because they rely directly on human reasoning, or because they require a formal model of the system to be built. In contrast, simulation-based methods allow users to adopt a black-box approach where the behavior of the system under test can be observed in a large number of different testing conditions without requiring explicit knowledge about its internal functionality. This makes simulation-based methods more scalable with regard to increasing system complexity. Yet, the literature research has also shown that simulation-based methods are still used relatively rarely in the domain of robotics in general and human-robot collaboration in particular. Therefore, it was concluded that more research on simulation-based testing in the context of human-robot collaboration is needed.

Q2: How to formalize and automate the task of hazard analysis?

Following the findings of RQ1, it was decided to focus on simulation-based hazard analysis as the main approach for this work. This, in turn, leads to the next research question, namely how to formalize the hazard analysis problem so that it can be solved in a simulation environment, preferably in an algorithmic manner that allows for an automated analysis. The book addresses this question by proposing an agent-based testing approach. To that end, the simulation model is partitioned into two interacting subsystems: The robot system which is being analyzed (referred to as *System under Test* or SUT) and a so-called *testing agent*. The testing agent interacts with the robot system and modifies the simulation environment in which the SUT is embedded. Thereby, it creates a variety of different testing conditions for the SUT. Given this framework, the hazard analysis can be framed as a search problem where the goal is to find critical agent behaviors that cause the SUT to enter a state that violates a given safety specification. To demonstrate the feasibility of this approach multiple series of experiments were conducted. In these experiments, the agent was represented by a digital human model, while the SUTs were different robot systems. Safety-critical design flaws were deliberately introduced into the SUTs and the developed techniques were deployed to expose these safety flaws by identifying unsafe agent behaviors which lead to critical human-robot collisions.

Q3: How to mitigate search-space complexity in simulation-based testing?

A common problem in hazard analyses is the vast number of possible interactions between agent and SUT that need to be analyzed to find potential hazards. This is also an issue in the agent-based testing approach of this work: With the exception of strongly simplified and abstracted system models, it is usually infeasible to simulate all possible agent behaviors. Instead, the computational budget should be concentrated on *critical* testing conditions under which the SUT is likely to violate safety specifications. Yet, creating such critical conditions in a systematic manner is challenging, since there is only limited a-priori knowledge about the specific conditions under which the SUT might fail. Furthermore, the agent behaviors need to be constrained in order to avoid infeasible or unrealistic agent behaviors. To address these issues, several techniques have been proposed in this book:

- Risk-guided search (Chapter 5)
- Automata-constrained risk-guided search (Chapter 6)
- A two-layer analysis combining formal verification and simulation (Chapter 7)

The effectiveness of these techniques has been demonstrated in experiments from the domain of industrial HRC. In the majority of experiments, the risk-guided search outperformed the random baseline in terms of effective hazard identification. The two-layer approach consistently outperformed both risk-guided search and random baseline, albeit at the cost of a higher modeling effort.

However, it should also be noted that the developed techniques have certain limitations, as will be discussed in the following Section.

8.2. Limitations

As explained in the discussion of the state-of-the-art (see Section 3.5), exhaustive exploration of all possible system behaviors is rarely feasible for anything other than very simple or highly abstract models. If more detail is required and the computational cost prohibits exhaustive exploration, one needs to prioritize the use of the limited computational budget. The risk guided search proposed in this book performs such a prioritization, since it focuses the search towards areas of the search space that exhibit high risk scores and are, therefore, more likely to expose hazards. Although this approach performs significantly better than random exploration, there is no guarantee that all hazards are found, as the search is not exhaustive. Furthermore, the simulation can only consider action sequences of finite length. For a given maximum sequence length, there may be unsafe states which are not discovered because longer action sequences are required to reach them. Referring back to Figure 3.4, one can see that it is difficult to achieve both exhaustive exploration and a high level of detail given a limited computational budget. The risk-guided search (Chapters 5-6) trades exhaustiveness for an increased level of detail while the two-level approach (Chapter 7) strikes a balance between formal verification and pure simulation-based testing. Nevertheless, one should be aware of the fact that there is no single best hazard analysis method. The methods proposed in this book should therefore be seen as an *addition* to existing methods, and not as a replacement.

Modeling assumptions with respect to the behavior of the testing agent, especially with respect to the agent's action- and and parameter space, can also lead to hazards being missed. Agents whose action space is too restrictive may be unable to expose certain hazards. On the other hand, choosing an excessively large action space with few restrictions leads to an explosion of both search space and modeling effort. Striking a balance between these two extremes is difficult, and general rules cannot be given as the problem is use-case specific. Applying an agent-based testing approach therefore requires some intuition as to which agent-related events or parameters might be particularly relevant and which constraints should be placed on the agent's behavior. Without any such intuition, it can be difficult to craft effective agent-based testing scenarios. Further constraints can be introduced by the modeling formalism itself. Extended Finite Automata (EFA), for instance, only allow event sequences that can be described by regular languages. While this limitation can be easily overcome by choos-

ing a more powerful modeling formalism, this would limit the possibilities for the two-level approach which relies on identification of critical behaviors by formal methods.

Another source of limitations lies in the risk metric. The risk metric in this work is a *heuristic*, that is, a pragmatic but simplified approach to quantifying risk. Risk is a combination of the severity and probability of occurrence of undesired events [96], and therefore inherently probabilistic. In contrast, the heuristic only takes into account the current state and does not consider probabilities of future events. While the use of risk metrics has been demonstrated to be effective in the experiments, there may be other situations where the risk metric does not accurately represent the actual risk which is associated to a given situation, and therefore misguides the search. Furthermore, if the risk metric does not correspond well to the actual risk of a given situation, this can lead to an undesired effect known as *reward hacking*. Reward hacking is a phenomenon where the agent succeeds in maximizing its reward (i.e., maximizing the risk metric), but fails to achieve the intended goal of the user (i.e., the discovery of unsafe states) due to a poor correspondence between the reward and the user's actual goal [182]. This issue has been illustrated by the case study in Section 5.3.4.

Finally, deployment effort can also be a limiting factor. Although risk- and hazard analyses are important to ensure safety of people and protect companies from liability and long-term economic risk, they do not provide short-term economic benefits. In an industrial environment characterized by time and cost pressures, novel hazard analysis methods may be viewed with a certain degree of skepticism, particularly if they are very time-consuming. Informal and semi-formal hazard analysis methods require none or relatively simple qualitative models of the system. They are therefore quick and cheap to deploy. In contrast, the techniques presented here are are more time consuming to deploy. The additional deployment effort is only justified when systems are complex or highly critical.

8.3. Future Work

Future work should target integration of more detailed digital human models. In this work, simplified human models were used because the focus was on proof of concepts rather than detail. For real-world application, this issue needs to be resolved. One possible solution is the use of

more detailed human body models with more degrees of freedom along with motion capture data. Details regarding this concept are laid out in [88, 51]. Furthermore, programming the human model needs to be as simple as possible and should not require excessive training. To that end, a graphical user interface (GUI), which allows users to conveniently define human movements, would be helpful. This GUI should allow users to define sequences of atomic actions (e.g., walking, sitting down/standing up, bending the upper body, or performing reaching motions) which can be parameterized with respect to specific properties such as path or goal coordinates. A possible concept for this is shown for instance in [51].

With a more detailed human model, it will also be possible to perform spatial reach-ability analyses, that is, calculation of the volumes which are reachable by different human body parts in a given layout of a robot cell. If the robot's path and velocity are known, one can allocate a potential a collision criticality to each point in that volume and thereby create a spatial safety map. The map can help to determine potentially critical collision points and required safety distances. However, this is only possible for static safeguards, because the map only considers static reachability without dynamic aspects. Besides spatial reachability, there are *state-space reachability* analyses, which calculate a set of possible states that a system can reach given an initial state, a set of possible control actions, and a model of the system' dynamics. In contrast to spatial-only reachability analysis, state-space reachability also considers dynamic effects. Furthermore, exhaustive state-space reachability analysis could potentially provide exhaustive proof of a system's safety. This would alleviate one of the major limitations of this work. In some special cases, reachability analysis has already been applied for safety purposes [9, 154, 136]. However, reachability analysis techniques require analytical modeling of system dynamics which is often intractable for complex 3D simulations or systems with black-box components. This is the reason why these techniques were not used in this work. However, as the state of the art progresses and the capabilities of these techniques evolve, they could be considered for future work.

Due to the previously mentioned limitation that the analysis is usually non-exhaustive, integration of coverage metrics (see Section 3.3.2) should also be considered. Apart from commonly used metrics such as code coverage, future work may explore new coverage criteria for agent-based testing. Another interesting addition could be a *similarity metric* for agent behaviors, which quantifies the degree of similarity between two given agent

behaviors. Such a similarity metric could be used to measure how far a given agent behavior deviates from a nominal expected behavior which can offer insights as to how likely such a behavior would be. A discussion of similarity metrics and their potential use can be found in [89].

Postprocessing capabilities could be necessary in the future, especially in cases where the analysis finds a large number of hazardous behaviors. In such cases, it is too time-consuming for the user inspect all simulation traces that lead to unsafe states. Postprocessing techniques such as clustering could help to organize the output into groups of similar behaviors to aid the human user in identifying the underlying causes of the hazards. This is especially important because even for a single underlying cause, there can be a large number of different unsafe behaviors which trigger to the same hazardous situation (as seen for instance in the experiments from Chapter 6).

Finally, reducing modeling efforts is an important area for future work, since high deployment efforts can hinder practical adoption of the developed techniques. Possible approaches could involve model libraries with preexisting components that can be tailored to a specific use-case (e.g., a generic formal model of the human operator such as in [191]), automatic generation of simulation models or digital twins [138, 23, 37], extraction of formal models from control code [50], or learning of automata models from simulation [179, 62].

8.4. Transfer to other Application Domains

This book has demonstrated the developed methods in the context of industrial HRC applications. Industrial HRC was chosen as an application area for two reasons: First, because there are well defined standards and regulations for industrial HRC systems and second, because industrial applications are usually associated with relatively structured environments and clearly defined tasks, which makes modeling easier. However, the presented agent-based testing approaches could also be transferred to other safety-critical domains. Safety-critical human-robot interactions can also take place in other contexts such as service, assistance, delivery or healthcare. While transferring the techniques to these domains is possible in principle, it can present additional challenges. In industrial human-robot collaboration, there is typically a structured environment and a clearly defined task. This provides a relatively clear framework for the modeling

of test scenarios and agents. Furthermore, the robot will typically interact with a single operator, or a known set of operators and the set of events that need to be considered when modeling the agents is limited. Other application areas may not provide such clearly defined boundaries. Consider, for instance, a mobile robot outside the industrial domain, such as a delivery or assistance robot. Such a robot will encounter unstructured environments where it will encounter a much larger range of different of agents, events, and environmental conditions than in an industrial setting. This makes it more difficult to define meaningful test agents, action spaces, and simulation scenarios. Therefore, users who want to apply simulation-based testing methods need to have a good initial understanding about the agents and events that are particularly safety-relevant, so they can tailor their test scenarios accordingly. Semi-formal hazard analysis methods like STPA could serve as a first step towards identifying critical agents and events to take into account. On this basis, it would be possible to build agent models and simulation-based test scenarios which can the be used for a more detailed hazard analysis as well as for early verification of robot control code.

8.5. Recent Advances in Artificial Intelligence

Another topic that merits a brief discussion are the recent advances in machine learning and artificial intelligence (AI). Although machine learning has been an active research area for a long time, it has gained additional interest with the advance of generative models. Generative models are pre-trained machine learning models which are capable of generating outputs based on prompts. These outputs can have different modalities, such as texts, images, videos, or they can be multimodal [70]. A particularly prominent class of generative models are Large Language Models (LLMs), which provide textual outputs such as natural language or program code [144].

In the context of this book, the recent advances in AI are interesting from two different perspectives, which can be summarized as (i) *AI for hazard analysis* and (ii) *hazard analysis for AI*.

The first aspect, *AI for hazard analysis*, refers to the question how AI and machine learning methods can be used for hazard analysis of safety-critical systems. With the proposed simulation- and agent-based approach,

this book has already given an answer to this question, at least on a conceptual level. In this book, methods from reinforcement learning, namely MCTS and PPO, have been used for this purpose. These methods primarily rely on random exploration coupled with heuristic optimization techniques. However, given more advanced AI algorithms, the proposed framework would also allow for the deployment of agents which have more advanced learning abilities, or even agents capable of semantical reasoning about potential hazards. This could significantly boost the performance of the techniques proposed in this book. Another interesting possibility is the use of LLMs as a supportive tool for semi-formal hazard analysis methods. Since semi-formal methods often rely on natural language to describe and analyse hazards (see Chapter 3), they can benefit from LLMs, which can interact with human analysts during the to possible hazards, provide suggestions, and point to potential hazards, albeit only in a supportive role. Such examples are provided by Thomas, who uses LLMs for FMEA [188], and by Qi et al. [163], who use LLMs for an STPA analysis.

The second aspect, *hazard analysis for AI*, refers to the question of how AI-based systems can be analyzed for potential hazards they may pose. Again,the concepts of this book (and those of falsification in general) can be seen as a partial answer to this question, as the simulation-based approach takes a black-box view of the SUT and thus also allows for the analysis of robot systems which are controlled by AI systems. However, this book is focused on physical safety (i.e., avoiding physical harm during human-robot interactions) whereas the scope of *AI safety* is much broader, encompassing various different ways in which AI systems may cause harm [11, 164]. Thus, the concepts of this book are not directly transferable to the field of AI safety as a whole. Yet, there are interesting parallels between this work and the procedures currently being developed for assessing the safety of AI systems. For instance, the concept of *red teaming*, which is applied to analyze the safety of LLMs, relies on a group of adversaries to find critical prompts which cause the LLM to provide harmful outputs, such as offensive or otherwise inappropriate language [69]. Although the application area is very different, there are clear parallels between this approach and the adversarial testing agents used throughout this book. These parallels suggest that there could be synergies between the two fields in the future.

8.6. Final Remarks

To conclude this book, some final remarks should be made. Regarding the practical value of the work, a potential point of criticism could be that the developed techniques are overly complex from a practical standpoint, and that current HRC applications are still simple enough so that human reasoning is adequate for hazard analysis. However, it is reasonable to assume that robots will continue to advance and achieve higher degrees of autonomy, and that the complexity of HRC applications will increase. At some point, this will likely necessitate the adoption of more sophisticated hazard analysis techniques, such as the simulation-based approach proposed here.

Furthermore, readers should be aware that the implementation and experiments in this book were focused on demonstration of conceptual aspects rather than on high performance and detail. Due to the conceptual nature of the presented research, the presented techniques are not yet ready for real-world use. On the other hand, one should note that the simulation scenarios are not too far from real-world use cases, and that the simulations were conducted on off-the-shelf personal computers for the consumer market. Given more computational resources along with more elaborate simulation models, it is reasonable to assume that scaling the developed techniques can achieve significant improvements beyond what was demonstrated in here.

Finally, it should be noted that a major point of this work was to demonstrate how to give computer-based hazard analysis methods some creativity and learning ability in the finding of hazards. A hazard analysis aims to uncover potentially dangerous situations at design time, i.e., before the system under test is put into operation or even implemented. In order to do that, human analysts need the ability to foresee and anticipate how a system might be used (or misused). Anticipating hazards requires a certain amount of *lateral* thinking and creativity. In other words, an analyst must run through a wide variety of "what if..." scenarios in a hazard analysis. Humans typically perform well at lateral thinking, that is, at considering such "what if" questions. Computer-based hazard analysis methods, however, perform well at evaluating systems characterized by complexity and large numbers of states and configurations, but as of yet, they only have a limited capability to create new and challenging testing conditions and uncover unforeseen hazards. By introducing a test agent that autonomously interacts with the SUT in simulation, has a learning ability, and receives

feedback in form of the risk metric, the risk-guided search aimed to recreate - or at least mimic - some of the lateral thinking and creativity of human analysts and coupling it with the computer's ability to efficiently simulate a large number of test cases. With current advances in machine learning, simulation- and agent-based testing can become a powerful to uncover safety flaws and hazards in robotic systems, ultimately contributing to the development of safer robot systems and applications.

Bibliography

[1] AALTONEN, I., AND SALMI, T. Experiences and expectations of collaborative robots in industry and academia: Barriers and development needs. *Procedia Manufacturing 38* (2019), 1151–1158.

[2] ABB AG. GoFa: Go far. Go Faster. Go further (product website). https://cobots.robotics.abb.com/en/robots/gofa/. Accessed: 2022-10-10.

[3] ABB AG. YuMi - IRB 14000 (Product Website). https://new.abb.com/products/robotics/collaborative-robots/irb-14000-yumi. Accessed: 2022-10-10.

[4] ADRIAENSEN, A., PINTELON, L., COSTANTINO, F., DI GRAVIO, G., AND PATRIARCA, R. An stpa safety analysis case study of a collaborative robot application. *IFAC-PapersOnLine 54*, 1 (2021), 534–539.

[5] AGERER, M. S. Gesima product website (german language). http://www.maschinen-sicherheit.net/07-seiten/7550-software-risikobeurteilung.php. Accessed: 2019-11-11.

[6] ÅKESSON, K., FLORDAL, H., AND FABIAN, M. Exploiting modularity for synthesis and verification of supervisors. *IFAC Proceedings Volumes 35*, 1 (2002), 175–180.

[7] ALAGI, H., ERGUN, S., DING, Y., HUCK, T. P., THOMAS, U., ZANGL, H., AND HEIN, B. Evaluation of On-Robot Capacitive Proximity Sensors with Collision Experiments for Human-Robot Collaboration (HRC). In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2022).

[8] ALARIE, S., AUDET, C., GHERIBI, A. E., KOKKOLARAS, M., AND LE DIGABEL, S. Two decades of blackbox optimization ap-

plications. *EURO Journal on Computational Optimization 9* (2021), 100011.

[9] ALTHOFF, M. *Reachability analysis and its application to the safety assessment of autonomous cars*. PhD thesis, Technische Universität München, 2010.

[10] ALTHOFF, M., FREHSE, G., AND GIRARD, A. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems 4*, 1 (2021).

[11] AMODEI, D., OLAH, C., STEINHARDT, J., CHRISTIANO, P., SCHULMAN, J., AND MANÉ, D. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[12] ANTONELLI, D., AND STADNICKA, D. Predicting and preventing mistakes in human-robot collaborative assembly. *IFAC-PapersOnLine 52*, 13 (2019), 743–748.

[13] ARAIZA-ILLAN, D., PIPE, A. G., AND EDER, K. Intelligent agent-based stimulation for testing robotic software in human-robot interactions. In *Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering* (2016), pp. 9–16.

[14] ARAIZA-ILLAN, D., WESTERN, D., PIPE, A., AND EDER, K. Coverage-driven verification— an approach to verify code for robots that directly interact with humans. In *Hardware and Software: Verification and Testing: 11th International Haifa Verification Conference, HVC 2015, Haifa, Israel, November 17-19, 2015, Proceedings 11* (2015), Springer, pp. 69–84.

[15] ARAIZA-ILLAN, D., WESTERN, D., PIPE, A., AND EDER, K. Model-based, coverage-driven verification and validation of code for robots in human-robot interactions. *arXiv preprint arXiv:1511.01354* (2015).

[16] ARAIZA-ILLAN, D., WESTERN, D., PIPE, A. G., AND EDER, K. Systematic and realistic testing in simulation of control code for robots in collaborative human-robot interactions. In *Annual Conference Towards Autonomous Robotic Systems* (2016), Springer.

[17] ASKARPOUR, M., MANDRIOLI, D., ROSSI, M., AND VICENTINI, F. SAFER-HRC: Safety analysis through formal verification in

human-robot collaboration. In *35th International Conference SAFECOMP* (2016).

[18] ASKARPOUR, M., MANDRIOLI, D., ROSSI, M., AND VICENTINI, F. A human-in-the-loop perspective for safety assessment in robotic applications. In *11th International Andrei P. Ershow Informatics Conference* (2017).

[19] ASKARPOUR, M., MANDRIOLI, D., ROSSI, M., AND VICENTINI, F. Modeling operator behaviour in the safety analysis of collaborative robotic applications. In *36th International Conference SAFECOMP* (2017).

[20] ASKARPOUR, M., ROSSI, M., AND TIRYAKILER, O. Co-simulation of human-robot collaboration: From temporal logic to 3D simulation. In *1st Workshop on Agents and Robots for Reliable Engineered Autonomy, AREA 2020* (2020), vol. 319, Open Publishing Association, pp. 1–8.

[21] AWAD, R., FECHTER, M., AND VAN HEERDEN, J. Integrated risk assessment and safety consideration during design of HRC workplaces. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (2017).

[22] BABIKIAN, A. A. Automated generation of test scenario models for the system-level safety assurance of autonomous vehicles. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (2020).

[23] BARTH, M., AND FAY, A. Automated generation of simulation models for control code tests. *Control Engineering Practice 21*, 2 (2013), 218–230.

[24] BAUER, A., WOLLHERR, D., AND BUSS, M. Human–robot collaboration: a survey. *International Journal of Humanoid Robotics 5*, 01 (2008), 47–66.

[25] BAUER, W., BENDER, M., BRAUN, M., RALLY, P., AND SCHOLTZ, O. Lightweight robots in manual assembly—best to start simply. *Frauenhofer-Institut für Arbeitswirtschaft und Organisation IAO, Stuttgart* (2016).

[26] BDIWI, M. Intuitive Roboterprogrammierung und intelligente Werkzeuge. *JOT Journal für Oberflächentechnik 62*, 8 (2022), 18–19.

[27] BEHRENS, R., AND ELKMANN, N. Study on meaningful and verified thresholds for minimizing the consequences of human-robot collisions. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), IEEE, pp. 3378–3383.

[28] BEHRENS, R., AND PLISKE, G. Human-robot collaboration: Partial supplementary examination [of pain thresholds] for their suitability for inclusion in publications of the DGUV and standardization, 2020.

[29] BENSACI, C., ZENNIR, Y., AND POMORSKI, D. A comparative study of STPA hierarchical structures in risk analysis: The case of a complex multi-robot mobile system. In *2018 2nd European Conference on Electrical Engineering and Computer Science (EECS)* (2018), IEEE, pp. 400–405.

[30] BOBICK, A. F. Movement, activity and action: the role of knowledge in the perception of motion. *Philosophical Transactions of the Royal Society of London (Biological Sciences) 352*, 1358 (1997).

[31] BOBKA, P., GERMANN, T., HEYN, J. K., GERBERS, R., DIETRICH, F., AND DRÖDER, K. Simulation platform to investigate safe operation of human-robot collaboration systems. In *6th CIRP Conference on Assembly Technologies and Systems (CATS)* (2016), vol. 44, pp. 187 – 192.

[32] BOHRER, R., TAN, Y. K., MITSCH, S., SOGOKON, A., AND PLATZER, A. A formal safety net for waypoint-following in ground robots. *IEEE Robotics and Automation Letters 4*, 3 (2019), 2910–2917.

[33] BOSONIC STUDIOS. Monte Carlo Tree Search (MCTS) algorithm tutorial and its explanation with Python code. `https://ai-boson.github.io/mcts/`. Accessed: 2022-09-10.

[34] BURDUK, A. Assessment of risk in a production system with the use of the fmea analysis and linguistic variables. In *International Conference on Hybrid Artificial Intelligence Systems* (2012), Springer, pp. 250–258.

[35] BÄCK, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 02 1996.

[36] CACCIABUE, P. C. *Guide to applying human factors methods: Human error and accident management in safety-critical systems*. Springer Science & Business Media, 2004.

[37] CAMPOS, J. G., LÓPEZ, J. S., QUIROGA, J. I. A., AND SEOANE, A. M. E. Automatic generation of digital twin industrial system from a high level specification. *Procedia Manufacturing 38* (2019), 1095–1102.

[38] CARROS, F., SCHWANINGER, I., PREUSSNER, A., RANDALL, D., WIECHING, R., FITZPATRICK, G., AND WULF, V. Care workers making use of robots: Results of a three-month study on human-robot interaction within a care home. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (2022), pp. 1–15.

[39] CASSANDRAS, C. G., AND LAFORTUNE, S. *Introduction to discrete event systems*. Springer, 2008.

[40] CHANCE, G., GHOBRIAL, A., LEMAIGNAN, S., PIPE, T., AND EDER, K. An agency-directed approach to test generation for simulation-based autonomous vehicle verification. *arXiv preprint arXiv:1912.05434* (2019).

[41] CHEN, Y.-L., AND LIN, F. Modeling of discrete event systems using finite state machines with parameters. In *Proceedings of the 2000. IEEE International Conference on Control Applications. Conference Proceedings (Cat. No. 00CH37162)* (2000), IEEE, pp. 941–946.

[42] CHOI, B. J., PARK, J., AND PARK, C. H. Formal verification for human-robot interaction in medical environments. In *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction* (2021), pp. 181–185.

[43] CLARKE, E. M., HENZINGER, T. A., VEITH, H., BLOEM, R., ET AL. *Handbook of model checking*, vol. 10. Springer, 2018.

[44] CLIFTON, J., AND LABER, E. Q-learning: Theory and applications. *Annual Review of Statistics and Its Application 7* (2020), 279–301.

[45] CORDERO, C. A., CARBONE, G., CECCARELLI, M., ECHAVARRI, J., AND MUNOZ, J. L. Experimental tests in human–robot collision evaluation and characterization of a new safety index for robot operation. *Mechanism and machine theory 80* (2014), 184–199.

[46] CORSO, A., DU, P., DRIGGS-CAMPBELL, K., AND KOCHENDERFER, M. J. Adaptive Stress Testing with Reward Augmentation for Autonomous Vehicle Validation. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (2019), IEEE.

[47] CORSO, A., LEE, R., AND KOCHENDERFER, M. J. Scalable autonomous vehicle safety validation through dynamic programming and scene decomposition. *arXiv preprint arXiv:2004.06801* (2020).

[48] CORSO, A., MOSS, R., KOREN, M., LEE, R., AND KOCHENDERFER, M. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research 72* (2021).

[49] CRONRATH, C., HUCK, T. P., LEDERMANN, C., KRÖGER, T., AND LENNARTSON, B. Relevant safety falsification by automata constrained reinforcement learning. In *2022 IEEE International Conference on Automation Science and Engineering (CASE)* (2022), IEEE.

[50] DA SILVA, L. D., DE ASSIS BARBOSA, L. P., GORGÔNIO, K., PERKUSICH, A., AND LIMA, A. M. N. On the automatic generation of timed automata models from function block diagrams for safety instrumented systems. In *2008 34th Annual Conference of IEEE Industrial Electronics* (2008), IEEE, pp. 291–296.

[51] DAI, F., KLOSE, S., HUCK, T. P., STUHLENMILLER, F., AND LEDERMANN, C. Human model in a simulation-assisted risk assessment tool for safe robot applications. In *ISR Europe 2023; 55th International Symposium on Robotics* (2023), VDE.

[52] DE MELO, M. S. P., DA SILVA NETO, J. G., DA SILVA, P. J. L., TEIXEIRA, J. M. X. N., AND TEICHRIEB, V. Analysis and com-

parison of robotics 3D simulators. In *2019 21st Symposium on Virtual and Augmented Reality (SVR)* (2019), IEEE, pp. 242–251.

[53] DIANATFAR, M., HESHMATISAFA, S., LATOKARTANO, J., AND LANZ, M. Feasibility analysis of safety training in human-robot collaboration scenario: Virtual reality use case. In *International Conference on Flexible Automation and Intelligent Manufacturing* (2023), Springer, pp. 246–256.

[54] DIMITROKALLI, A., VOSNIAKOS, G.-C., NATHANAEL, D., AND MATSAS, E. On the assessment of human-robot collaboration in mechanical product assembly by use of virtual reality. *Procedia Manufacturing 51* (2020), 627–634.

[55] DING, W., CHEN, B., XU, M., AND ZHAO, D. Learning to collide: An adaptive safety-critical scenarios generating method. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), IEEE.

[56] DITTRICH, F., AND WOERN, H. Robot activity adaptation for safe human-robot collaboration based on probabilistic risk modeling. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)* (2016), IEEE, pp. 1–6.

[57] DOCUFY GMBH. "DOCUFY Machine Safety" (product website). `https://www.docufy.de/en/products/docufy-machine-safety`. Accessed: 2019-11-11.

[58] DU, P., AND DRIGGS-CAMPBELL, K. Adaptive failure search using critical states from domain experts. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 38–44.

[59] DUECK, G. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational physics 104*, 1 (1993), 86–92.

[60] EDER, K. Gaining confidence in the trustworthiness of robotic and autonomous systems. In *Software Engineering for Robotics*. Springer, 2021, pp. 139–164.

[61] EUROPEAN UNION. DIRECTIVE 2006/42/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 17 May 2006 on machinery, and amending Directive 95/16/EC (recast), 2006.

[62] FAROOQUI, A., FALKMAN, P., AND FABIAN, M. Towards automatic learning of discrete-event models from simulations. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)* (2018), IEEE, pp. 857–862.

[63] FERRER, B. R., AHMAD, B., LOBOV, A., VERA, D. A., LASTRA, J. L. M., AND HARRISON, R. An approach for knowledge-driven product, process and resource mappings for assembly automation. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)* (2015), IEEE.

[64] FRANÇOIS-LAVET, V., HENDERSON, P., ISLAM, R., BELLEMARE, M. G., PINEAU, J., ET AL. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning 11*, 3-4 (2018), 219–354.

[65] FRAUNHOFER IFF. Cobot Planer - Design Safe HRC Applications Quickly and Easily (Online Tool). `https://www.cobotplaner.de/preambel`, 2023. Accessed: 2023-01-02.

[66] FULTON, N., MITSCH, S., QUESEL, J.-D., VÖLP, M., AND PLATZER, A. Keymaera x: An axiomatic tactical theorem prover for hybrid systems. In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25* (2015), Springer, pp. 527–538.

[67] FURIA, C. A., MANDRIOLI, D., MORZENTI, A., AND ROSSI, M. *Modeling time in computing*. Springer Science & Business Media, 2012.

[68] GAEDE, C., RANZ, F., HUMMEL, V., AND ECHELMEYER, W. A study on challenges in the implementation of human-robot collaboration. *Journal of Engineering, Management and Operations Vol. I* (2020), 29.

[69] GANGULI, D., LOVITT, L., KERNION, J., ASKELL, A., BAI, Y., KADAVATH, S., MANN, B., PEREZ, E., SCHIEFER, N., NDOUSSE, K., ET AL. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858* (2022).

[70] GARCÍA-PEÑALVO, F., AND VÁZQUEZ-INGELMO, A. What do we mean by genai? a systematic mapping of the evolution, trends, and techniques involved in generative ai.

[71] GELLY, S., KOCSIS, L., SCHOENAUER, M., SEBAG, M., SILVER, D., SZEPESVÁRI, C., AND TEYTAUD, O. The grand challenge of computer go: Monte carlo tree search and extensions. *Communications of the ACM 55*, 3 (2012), 106–113.

[72] GOPINATH, V., AND JOHANSEN, K. Risk assessment process for collaborative assembly–a job safety analysis approach. *Procedia CIRP 44* (2016), 199–203.

[73] GROZA, B. Traffic models with adversarial vehicle behaviour. *arXiv preprint arXiv:1701.07666* (2017).

[74] GRZESKOWIAK, F., GONON, D., DUGAS, D., PAEZ-GRANADOS, D., CHUNG, J. J., NIETO, J., SIEGWART, R., BILLARD, A., BABEL, M., AND PETTRÉ, J. Crowd against the machine: A simulation-based benchmark tool to evaluate and compare robot capabilities to navigate a human crowd. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), IEEE, pp. 3879–3885.

[75] GTE INDUSTRIEELEKTRONIK GMBH. Product flyer "cobosafe" (online). `https://www.gte.de/material/325-2811-001_EN11_Flyer_CoboSafe_CBSF-75-Basic.pdf`. Accessed: 2022-07-06.

[76] GUIOCHET, J. Hazard analysis of human–robot interactions with HAZOP–UML. *Safety Science 84* (2016), 225 – 237.

[77] GUIOCHET, J., DO HOANG, Q. A., KAANICHE, M., AND POWELL, D. Model-based safety analysis of human-robot interactions: The MIRAS walking assistance robot. In *2013 IEEE 13th International Conference on Rehabilitation Robotics (ICORR)* (2013), IEEE.

[78] HADDADIN, S., ALBU-SCHAEFFER, A., AND HIRZINGER, G. Requirements for safe robots: Measurements, analysis and new insights. *The International Journal of Robotics Research 28*, 11-12 (2009), 1507–1527.

[79] HADDADIN, S., ALBU-SCHAFFER, A., FROMMBERGER, M., AND HIRZINGER, G. The role of the robot mass and velocity in physical human-robot interaction-Part II: Constrained blunt impacts. In *2008 IEEE International Conference on Robotics and Automation* (2008), IEEE, pp. 1331–1338.

[80] HADDADIN, S., ALBU-SCHÄFFER, A., AND HIRZINGER, G. Safety evaluation of physical human-robot interaction via crash-testing. In *Robotics: Science and systems* (2007), vol. 3, Citeseer, pp. 217–224.

[81] HADDADIN, S., ALBU-SCHAFFER, A., AND HIRZINGER, G. The role of the robot mass and velocity in physical human-robot interaction-Part I: Non-constrained blunt impacts. In *2008 IEEE International Conference on Robotics and Automation* (2008), IEEE, pp. 1331–1338.

[82] HANNA, A., LARSSON, S., GÖTVALL, P.-L., AND BENGTSSON, K. Deliberative safety for industrial intelligent human–robot collaboration: Regulatory challenges and solutions for taking the next step towards industry 4.0. *Robotics and Computer-Integrated Manufacturing 78* (2022), 102386.

[83] HAWKINS, H., AND ALEXANDER, R. Situation coverage testing for a simulated autonomous car–an initial case study. *arXiv preprint arXiv:1911.06501* (2019).

[84] HOLLNAGEL, E. The phenotype of erroneous actions: Implications for HCI design. *Human-computer interaction and complex systems* (1991), 73–121.

[85] HOLZMANN, G. J. The model checker SPIN. *IEEE Transactions on Software Engineering 23*, 5, 279–295.

[86] HORNUNG, L., AND WURLL, C. Human-robot collaboration: a survey on the state of the art focusing on risk assessment. In *Berichte aus der Robotik - Robotix-Academy Conference for Industrial Robotics (RACIR) 2021* (Sep. 2021), pp. 10–17.

[87] HUCK, T. P., KAISER, M., CRONRATH, C., LENNARTSON, B., KRÖGER, T., AND ASFOUR, T. Reinforcement learning for safety testing: Lessons from a mobile robot case study. *arXiv preprint arXiv:2311.02907* (2023).

[88] HUCK, T. P., LEDERMANN, C., KLOSE, S., DAI, F., MATTHIAS, B., AND BYNER, C. Development of a simulation-based risk assessment tool for HRC applications. In *ISR Europe 2022; 54th International Symposium on Robotics* (2022), VDE, pp. 1–8.

[89] HUCK, T. P., LEDERMANN, C., AND KRÖGER, T. Testing robot system safety by creating hazardous human worker behavior in simulation. *IEEE Robotics and Automation Letters 7*, 2 (2021), 770–777.

[90] HUCK, T. P., LEDERMANN, C., AND KRÖGER, T. Simulation-based testing for early safety-validation of robot systems. In *2020 IEEE Symposium on Product Compliance Engineering* (2020), IEEE.

[91] HUCK, T. P., LEDERMANN, C., AND KRÖGER, T. Virtual adversarial humans finding hazards in robot workplaces. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), IEEE.

[92] HUCK, T. P., MÜNCH, N., HORNUNG, L., LEDERMANN, C., AND WURLL, C. Risk assessment tools for industrial human-robot collaboration: Novel approaches and practical needs. *Safety Science 141* (2021).

[93] HUCK, T. P., SELVARAJ, Y., CRONRATH, C., LEDERMANN, C., FABIAN, M., LENNARTSON, B., AND KRÖGER, T. Hazard analysis of collaborative automation systems: A two-layer approach based on supervisory control and simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2023), IEEE, pp. 10560–10566.

[94] IBF GMBH. Safexpert Software for CE Marking (product website). `https://www.ibf.at/en/ce-software-safexpert`, 2019. Accessed: 2019-11-11.

[95] INKULU, A. K., BAHUBALENDRUNI, M. R., AND DARA, A. Challenges and opportunities in human robot collaboration context of Industry 4.0 - A state of the art review. *Industrial Robot: the international journal of robotics research and application 49*, 2 (2022), 226–239.

[96] INTERNATIONAL ELECTROTECHNICAL COMMISSION. IEC 61508-1:2010-1 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements, 2006.

[97] INTERNATIONAL ELECTROTECHNICAL COMMISSION. IEC 61882:2016: Hazard and operability studies (HAZOP studies) - application guide, 2016.

[98] INTERNATIONAL FEDERATION OF ROBOTICS (IFR). World robotics 2022 report, 2022.

[99] INTERNATIONAL ORGANISATION FOR STANDARDIZATION. Iso deliverables: The different types of iso publications (online). `https://www.iso.org/deliverables-all.html`, 2024. Accessed: 2024-08-10.

[100] INTERNATIONAL ORGANISATION FOR STANDARDIZATION. Iso/iec awi ts 22440-1 artificial intelligence — functional safety and ai systems). `https://www.iso.org/standard/89535.html`, 2024. Accessed: 2024-08-27.

[101] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 12100:2010 Safety of machinery - General principles for design - Risk assessment and risk reduction, 2010.

[102] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 13855:2010 Safety of machinery - Positioning of safeguards with respect to the approach speeds of parts of the human body, 2010.

[103] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 10218-2:2011 Robots and robotic devices - Safety requirements for industrial robots - Part 2: Robot systems and integration, 2011.

[104] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/TR 14121-2:2012 Safety of machinery — Risk assessment — Part 2: Practical guidance and examples of methods, 2012.

[105] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 13842:2014 Robots and robotic devices - Safety requirements for personal care robots, 2014.

[106] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 13849-1:2015: Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design, 2015.

[107] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO TS 15066:2016 Robots and robotic devices - Collaborative robots, 2016.

[108] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 13854:2017 Safety of machinery - Minimum gaps to avoid crushing of parts of the human body, 2017.

[109] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 26262-3:2018: Road vehicles — Functional safety - Part 3: Concept phase, 2018.

[110] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 13857:2019 Safety of machinery - Safety distances to prevent hazard zones being reached by upper and lower limbs, 2019.

[111] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/PAS 21448:2019-01: Road vehicles - Safety of the intended functionality, 2019.

[112] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/IEC TR 5469:2024 Artificial intelligence - Functional safety and AI systems, 2024.

[113] KAPINSKI, J., DESHMUKH, J. V., JIN, X., ITO, H., AND BUTTS, K. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine 36*, 6 (2016), 45–64.

[114] KARUNAKARAN, D., WORRALL, S., AND NEBOT, E. Efficient falsification approach for autonomous vehicle validation using a parameter optimisation technique based on reinforcement learning. *arXiv preprint arXiv:2011.07699* (2020).

[115] KLISCHAT, M., AND ALTHOFF, M. Generating critical test scenarios for automated vehicles with evolutionary algorithms. In *2019 IEEE Intelligent Vehicles Symposium (IV)* (2019), IEEE.

[116] KOREN, M., ALSAIF, S., LEE, R., AND KOCHENDERFER, M. J. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)* (2018), IEEE, pp. 1–7.

[117] KOREN, M., CORSO, A., AND KOCHENDERFER, M. J. The adaptive stress testing formulation. *arXiv preprint arXiv:2004.04293* (2020).

[118] KOUSKOULAS, Y., RENSHAW, D., PLATZER, A., AND KAZANZIDES, P. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In *Proceedings of the 16th international conference on Hybrid systems: computation and control* (2013), pp. 263–272.

[119] KOVINCIC, N., GATTRINGER, H., MÜLLER, A., AND BRANDSTÖTTER, M. A boosted decision tree approach for a safe human-robot collaboration in quasi-static impact situations. In *International Conference on Robotics in Alpe-Adria Danube Region* (2020), Springer, pp. 235–244.

[120] KUKA AG. LBR iiwa (Product Website). `https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa`. Accessed: 2022-10-10.

[121] KUKA AG. Sensitives Fügen von Kegelrädern im Mensch-Roboter-Kollaboration (MRK) Betrieb (Promotional Video). `https://www.youtube.com/watch?v=02TzqIvWiso&t=37s`. Accessed: 2020-06-01.

[122] KULIĆ, D., AND CROFT, E. A. Real-time safety for human–robot interaction. *Robotics and Autonomous Systems 54*, 1 (2006), 1–12.

[123] LASOTA, P. A., FONG, T., SHAH, J. A., ET AL. *A survey of methods for safe human-robot interaction*. Now Publishers, 2017.

[124] LEE, J., JEON, W., KIM, G.-H., AND KIM, K.-E. Monte-carlo tree search in continuous action spaces with value gradients. In *Proceedings of the AAAI conference on artificial intelligence* (2020), vol. 34, pp. 4561–4568.

[125] LEE, R., KOCHENDERFER, M. J., MENGSHOEL, O. J., BRAT, G. P., AND OWEN, M. P. Adaptive stress testing of airborne colli-

sion avoidance systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)* (2015), IEEE.

[126] LEE, R., MENGSHOEL, O. J., SAKSENA, A., GARDNER, R. W., GENIN, D., SILBERMANN, J., OWEN, M., AND KOCHENDERFER, M. J. Adaptive stress testing: Finding likely failure events with reinforcement learning. *Journal of Artificial Intelligence Research 69* (2020).

[127] LEE, S.-D., KIM, B.-S., AND SONG, J.-B. Human–robot collision model with effective mass and manipulability for design of a spatial manipulator. *Advanced Robotics 27*, 3 (2013), 189–198.

[128] LENNARTSON, B., FABIAN, M., AND AKESSON, K. Introduction to discrete event systems (lecture notes), 2009.

[129] LESAGE, B. M. J.-R., AND ALEXANDER, R. SASSI: Safety Analysis using Simulation-based Situation Coverage for Cobot Systems. In *Proceedings of SafeComp 2021* (2021), York.

[130] LESTINGI, L., ASKARPOUR, M., BERSANI, M. M., AND ROSSI, M. Statistical model checking of human-robot interaction scenarios. *arXiv preprint arXiv:2007.11738* (2020).

[131] LEVESON, N. *Engineering a safer world: Systems thinking applied to safety*. MIT Press, 2011.

[132] LIU, Y., HABIBNEZHAD, M., JEBELLI, H., AND MONGA, V. Worker-in-the-loop cyber-physical system for safe human-robot collaboration in construction. In *Computing in Civil Engineering 2021*. 2022, pp. 1075–1083.

[133] LOYOLA-GONZALEZ, O. Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE access 7* (2019), 154096–154113.

[134] MAJUMDAR, A., AND PAVONE, M. How should a robot assess risk? Towards an axiomatic theory of risk in robotics. In *Robotics Research*. Springer, 2020, pp. 75–84.

[135] MALIK, R., AKESSON, K., FLORDAL, H., AND FABIAN, M. Supremica-An efficient tool for large-scale discrete event systems. *IFAC-PapersOnLine 50*, 1 (2017), 5794 – 5799. 20th IFAC World Congress.

[136] MANZINGER, S., PEK, C., AND ALTHOFF, M. Using reachable sets for trajectory planning of automated vehicles. *IEEE Transactions on Intelligent Vehicles 6*, 2 (2020), 232–248.

[137] MARTIN-GUILLEREZ, D., GUIOCHET, J., POWELL, D., AND ZANON, C. A UML-based method for risk analysis of human-robot interactions. In *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems* (2010), pp. 32–41.

[138] MARTÍNEZ, G. S., SIERLA, S., KARHELA, T., AND VYATKIN, V. Automatic generation of a simulation-based digital twin of an industrial process plant. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society* (2018), IEEE, pp. 3084–3089.

[139] MARVEL, J. A., FALCO, J., AND MARSTIO, I. Characterizing task-based human–robot collaboration safety in manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 45*, 2 (2014).

[140] MATSAS, E., VOSNIAKOS, G.-C., AND BATRAS, D. Prototyping proactive and adaptive techniques for human-robot collaboration in manufacturing using virtual reality. *Robotics and Computer-Integrated Manufacturing 50* (2018), 168–180.

[141] MATTHIAS, B., OBERER-TREITZ, S., STAAB, H., SCHULLER, E., AND PELDSCHUS, S. Injury risk quantification for industrial robots in collaborative operation with humans. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)* (2010), VDE, pp. 1–6.

[142] MAYRING, P. Qualitative content analysis. *A companion to qualitative research 1*, 2004 (2004), 159–176.

[143] METZNER, M., UTSCH, D., WALTER, M., HOFSTETTER, C., RAMER, C., BLANK, A., AND FRANKE, J. A system for human-in-the-loop simulation of industrial collaborative robot applications. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)* (2020), IEEE, pp. 1520–1525.

[144] MINAEE, S., MIKOLOV, T., NIKZAD, N., CHENAGHLU, M., SOCHER, R., AMATRIAIN, X., AND GAO, J. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).

[145] MITKA, E., AND MOUROUTSOS, S. G. Applying the stamp system safety engineering methodology to the design of a domestic robot. *International Journal of Applied Systemic Studies 6*, 1 (2015), 81–102.

[146] MITSCH, S., GHORBAL, K., AND PLATZER, A. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24-June 28, 2013* (2013).

[147] MOHAJERANI, S., MALIK, R., AND FABIAN, M. A framework for compositional synthesis of modular nonblocking supervisors. *IEEE Transactions on Automatic Control 59*, 1 (2013), 150–162.

[148] MYERS, G. J., SANDLER, C., AND BADGETT, T. *The art of software testing*. John Wiley & Sons, 2011.

[149] NAJMAEI, N., LELE, S., KERMANI, M., AND SOBOT, R. Human factors for robot safety assessment. In *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics* (2010), IEEE, pp. 539–544.

[150] NÖRDINGER, S. Den Mensch im Blick: Wie Sensoren Cobots absichern. `https://www.produktion.de/technik/den-mensch-im-blick-wie-sensoren-cobots-absichern-290.html`. Accessed: 2019-05-22.

[151] NURCHALIFAH, D., BLUMENTHAL, S., LO IACONO, L., AND HOCHGESCHWENDER, N. Analysing the safety and security of a UV-C disinfection robot. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2023), IEEE.

[152] OBERER-TREITZ, S. *Abschätzung der Kollisionsfolgen von Robotern zur Bewertung des sicheren Einsatzes in der Mensch-Roboter-Kooperation*. Stuttgart: Fraunhofer Verlag, 2018.

[153] PANDEY, D., AND PANDEY, P. Approximate Q-learning: An introduction. In *2010 second international conference on machine learning and computing* (2010), IEEE, pp. 317–320.

[154] PEREIRA, A., AND ALTHOFF, M. Overapproximative human arm occupancy prediction for collision avoidance. *IEEE Transactions on Automation Science and Engineering 15*, 2 (2017), 818–831.

[155] PILZ GMBH & CO. KG. The first safe camera system SafetyEYE opens up new horizons for safety & security. https://www.automate.org/news/the-first-safe-camera-system-safetyeye-opens-up-new-horizons-for-safety-and-security. Accessed: 2022-07-06.

[156] PILZ GMBH & CO. KG:. Human-robot collaboration 4.0: Standard-compliant HRC with the collision measurement set PRMS. https://www.pilz.com/en-DE/company/news/articles/200588. Accessed: 2021-11-19.

[157] PILZ GMBH & CO. KG:. Overview of basic standards. https://www.pilz.com/en-DE/support/knowhow/law-standards-norms/iso-standards. Accessed: 2021-11-19.

[158] PLATBROOD, F., AND GORNEMANN, O. *Safe robotics - Safety in collaborative robot systems*. SICK AG White Paper, 2017.

[159] PLATZER, A. *Logical foundations of cyber-physical systems*. Springer, 2018.

[160] POWELL, M. J. Direct search algorithms for optimization calculations. *Acta numerica 7* (1998), 287–336.

[161] PRADELLA, M. A user's guide to Zot. *arXiv preprint arXiv:0912.5014* (2009).

[162] PROETZSCH, M., BERNS, K., SCHUELE, T., AND SCHNEIDER, K. Formal verification of safety behaviours of the outdoor robot RAVON. In *ICINCO-RA (1)* (2007).

[163] QI, Y., ZHAO, X., KHASTGIR, S., AND HUANG, X. safety analysis in the era of large language models: a case study of stpa using chatgpt. *arXiv preprint arXiv:2304.01246* (2023).

[164] RAJI, I. D., AND DOBBE, R. Concrete problems in ai safety, revisited. *arXiv preprint arXiv:2401.10899* (2023).

[165] RAMADGE, P. J., AND WONHAM, W. M. The control of discrete event systems. *Proceedings of the IEEE 77*, 1 (1989), 81–98.

[166] RATHMAIR, M., LUCKENEDER, C., HASPL, T., REITERER, B., HOCH, R., HOFBAUR, M., AND KAINDL, H. Formal verification of safety properties of collaborative robotic applications including variability. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)* (2021), IEEE, pp. 1283–1288.

[167] RETHINK ROBOTICS GMBH. Sawyer Black Edition (Product Website). `https://www.rethinkrobotics.com/sawyer`. Accessed: 2022-10-10.

[168] ROBLA-GÓMEZ, S., BECERRA, V. M., LLATA, J. R., GONZALEZ-SARABIA, E., TORRE-FERRERO, C., AND PEREZ-ORIA, J. Working together: A review on safe human-robot collaboration in industrial environments. *IEEE Access 5* (2017), 26754–26773.

[169] ROHMER, E., SINGH, S. P. N., AND FREESE, M. CoppeliaSim (formerly V-REP): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)* (2013).

[170] RÜCKERT, P., WOHLFROMM, L., AND TRACHT, K. Implementation of virtual reality systems for simulation of human-robot collaboration. *Procedia Manufacturing 19* (2018), 164–170.

[171] RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[172] RUTENBAR, R. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine 5*, 1 (1989), 19–26.

[173] SANDERUD, A., THOMESSEN, T., OSUMI, H., AND NIITSUMA, M. A proactive strategy for safe human-robot collaboration based on a simplified risk analysis. In *Modeling, Identification and Control, Vol. 36, No. 1, 2015,* (2015), pp. 11–21.

[174] SCHLOSSER, P., AND LEDERMANN, C. Using diverse neural networks for safer human pose estimation: Towards making neural networks know when they don't know. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), IEEE, pp. 10305–10312.

[175] SCHLOSSER, P., AND LEDERMANN, C. Achieving hard real-time capability for 3d human pose estimation systems. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), IEEE, pp. 3772–3778.

[176] SCHLOTZHAUER, A., STOTZ, T., AWAD, R., AND KRAUS, W. Virtual validation of power and force limiting setups in human-robot-collaboration. *Procedia CIRP 107* (2022), 845–850.

[177] SCHRÖDER, J. Basics of stochastic automata theory. *Modelling, State Observation and Diagnosis of Quantised Systems* (2003), 13–35.

[178] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[179] SELVARAJ, Y., FAROOQUI, A., PANAHANDEH, G., AHRENDT, W., AND FABIAN, M. Automatically learning formal models from autonomous driving software. *Electronics 11*, 4 (2022), 643.

[180] SHIN, H., CHOI, J., CHOI, J., AND RHIM, S. Physical safety analysis of robot considering impactor shape. In *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)* (2017), IEEE, pp. 1–5.

[181] SICK AG. safeVisionary2Safe 3D environment perception opens up new dimensions (Product Website. `https://www.sick.com/de/en/catalog/produkte/safety/sichere-kamerasysteme/safevisionary2/c/g568562`. Accessed: 2023-11-06.

[182] SKALSE, J., HOWE, N. H., KRASHENINNIKOV, D., AND KRUEGER, D. Defining and characterizing reward hacking. *arXiv preprint arXiv:2209.13085* (2022).

[183] SKOLDSTAM, M., AKESSON, K., AND FABIAN, M. Modeling of discrete event systems using finite automata with variables. In *2007 46th IEEE Conference on Decision and Control* (2007), pp. 3387–3392.

[184] SMITH, D. J., AND SIMPSON, K. G. L. *Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety,*

IEC 61508 (2010 Edition) and Related Standards, Including Process IEC 61511 and Machinery IEC 62061 and ISO 13849, 3rd ed. Elsevier, Oxford, UK, 2010.

[185] SONG, C. S., AND KIM, Y.-K. The role of the human-robot interaction in consumers' acceptance of humanoid retail service robots. *Journal of Business Research 146* (2022), 489–503.

[186] STANTON, N. A. Hierarchical task analysis: Developments, applications, and extensions. *Applied ergonomics 37*, 1 (2006), 55–79.

[187] SVARNY, P., ROZLIVEK, J., RUSTLER, L., AND HOFFMANN, M. 3D collision-force-map for safe human-robot collaboration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), IEEE, pp. 3829–3835.

[188] THOMAS, D. Revolutionizing failure modes and effects analysis with chatgpt: unleashing the power of ai language models. *Journal of Failure Analysis and Prevention 23*, 3 (2023), 911–913.

[189] UNIVERSAL ROBOTS A/S. Cobots by Universal Robots (Product Website). `https://www.universal-robots.com/products/`. Accessed: 2023-07-26.

[190] UTHER, W. *Markov Decision Processes. In: Encyclopedia of Machine Learning*. Springer US, Boston, MA, 2010, pp. 642–646.

[191] VICENTINI, F., ASKARPOUR, M., ROSSI, M. G., AND MANDRIOLI, D. Safety assessment of collaborative robotics through automated formal verification. *IEEE Transactions on Robotics 36*, 1 (2019).

[192] WANG, D., TAN, D., AND LIU, L. Particle swarm optimization algorithm: an overview. *Soft computing 22* (2018), 387–408.

[193] WEBSTER, M., DIXON, C., FISHER, M., SALEM, M., SAUNDERS, J., KOAY, K., AND DAUTENHAHN, K. Formal verification of an autonomous personal robotic assistant. *Formal Verification and Modeling in Human-Machine Systems* (2014).

[194] WEBSTER, M., WESTERN, D., ARAIZA-ILLAN, D., DIXON, C., EDER, K., FISHER, M., AND PIPE, A. G. A corroborative approach to verification and validation of human–robot teams. *The International Journal of Robotics Research 39*, 1 (2020), 73–99.

[195] WEKA MEDIA GMBH & CO. KG. Weka manager product website. `https://www.weka.de/ps/weka-manager-ce-english`. Accessed: 2019-11-12.

[196] WHALEN, M. W., RAJAN, A., HEIMDAHL, M. P., AND MILLER, S. P. Coverage metrics for requirements-based testing. In *Proceedings of the 2006 international symposium on Software testing and analysis* (2006), pp. 25–36.

[197] WIGAND, KRÜGER, WREDE, STUKE, AND EDLER. *Report on the Application of the COVR Toolkit and its Protocols for the Certification of Cooperative Robot Workstations*. University of Bielefeld, 2020.

[198] WINANDS, M. H. M. *Monte-Carlo Tree Search*. Springer International Publishing, Cham, 2015, pp. 1–6.

[199] WONHAM, W. M., AND RAMADGE, P. J. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization 25*, 3 (1987), 637–659.

[200] YEE, T., LISÝ, V., BOWLING, M. H., AND KAMBHAMPATI, S. Monte carlo tree search in continuous action spaces with execution uncertainty. In *IJCAI* (2016), pp. 690–697.

9. Appendix

A. Experiment Details

This section contains some additional details about the experiments which have been omitted from the main body of the book for sake of brevity.

A.1. Experiments from Chapter 5

The descriptions and images in this subsection are taken from [91].

The agent's action space in the first experiment series is defined as follows:

$$A = A_{\text{Walking}} \times A_{\text{UpperBody}} \tag{9.1}$$

with A_{Walking} and $A_{\text{UpperBody}}$ being defined as follows:

$$\begin{aligned} A_{\text{Walking}} = \{&(\text{stop}), (\text{step forward}), (\text{step left } 45°),\\ &(\text{step left } 90°), (\text{step right } 45°), (\text{step right } 90°)\} \end{aligned} \tag{9.2}$$

$$\begin{aligned} A_{\text{UpperBody}} = \{&(\text{upright}), (\text{bend forward}), (\text{bend left}),\\ &(\text{bend right}), (\text{bend forward and right}),\\ &(\text{bend forward and left})\} \end{aligned} \tag{9.3}$$

The reward for the MCTS algorithm is defined as follows:

$$R = \begin{cases} R_E & \text{if } \text{spec}(s) = \text{false} \\ -\frac{1}{r_{max}} & \text{if } \text{spec}(s) = \text{true} \wedge k = n \\ 0 & \text{otherwise} \end{cases} \tag{9.4}$$

Where k denotes the current step of the action sequence and n the maximum action sequence length. R_E is a nonnegative constant to reward the

discovery of unsafe states and $-1/r_{max}$ is a negative penalty which is given if the action sequence reaches its maximum length (i.e., $k = n$) without discovering an unsafe state. The value r_{max} is the maximum risk metric value that has occured over the course of the action sequence. The risk metric is defined similarly to to Eq. (5.3), although slight alterations have been made due to implementation-specific details (see [91]). Note that the lower r_{max} the more penalty is incurred. This encourages the algorithm to find *unsafe* behaviors.

Furthermore, note that Eq. (9.4) differs slightly from other reward formulations used in this work. This is because, as already mentioned in Sec. 5.3.2, the MCTS implementation of Lee et al. [126] was used for this experiment, which required a particular reward formulation.

Details about the scenarios are given below.

Scenario 5-A

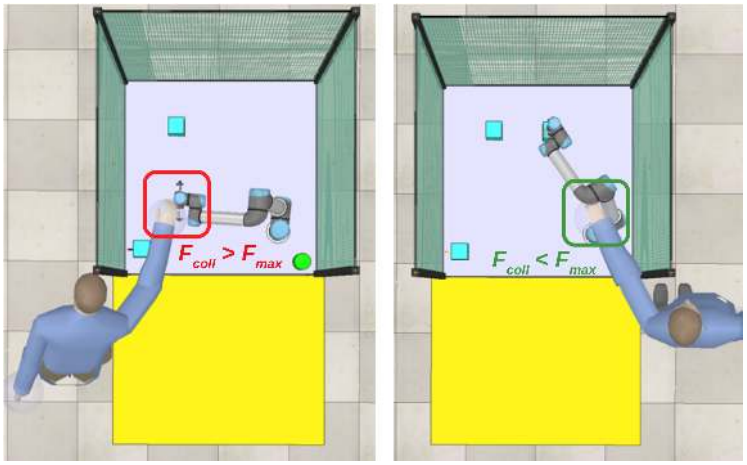


Figure A.1.: The human worker model walks around the safety mat and reaches around the fence, thus avoiding the safety measures and causing collisions.
Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Task and layout: The cell is seen in Figure A.1. The robot performs a pick-and-place task in a cell which is encased by fences on three sides. The front of the cell has no fencing to allows access for human workers.

Safety measures: To avoid collisions, a sensor mat in front of the cell (yellow area in Figure A.1). The sensor mat stops the robot when the worker steps on it.

Safety flaw/hazard: The safety flaw in this scenario is that the mat only covers the front of the cell, but not the right and left side. This allows for potential collisions when the human walks around the mat and reaches around the fence into the cell. Note that a collision is possible both on the left side and on the right side of the cell. However, only on the left side, the cartesian velocity of the robot is high enough to cause a critical collision. The collision on the right side represents only a local optimum w.r.t. the risk metric (i.e., a collision is possible, but the risk metric is not sufficiently high for this state to be counted as an unsafe state).

Scenario 5-B

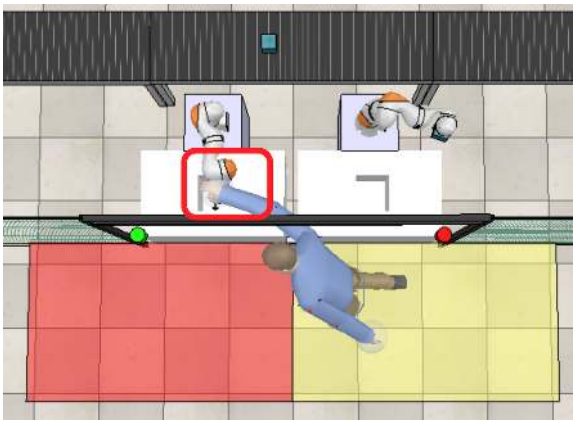


Figure A.2.: The agent enters through the right laser scanner field (yellow area), but then turns to and reaches for the left robot, which is still running, since the left scanner field (red area) is not triggered.
Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Task and layout: Two KUKA iiwa robots are working at an assembly line besides each other. Each robot is located between a table and a conveyor, performing pick-and-place movements to grab objects from the table and put them onto the conveyor.

Safety measures: Each of the two robots monitors the floor area for approaching humans with a laser scanner. The laser scanner fields of the two robots are indicated in Figure A.2 (red for the left robot and yellow for the right robot). Entering a field triggers a safety stop of the respective robot.

Safety flaw/hazard: The laser scanner fields are not sufficiently large to detect the human approaching. In particular, the fields, the fields should overlap in the middle, which is not the case. Without this overlap, it is

possible for the human to provoke a collision with the robot. As seen in Figure A.2, the human enters through the right laser scanner field and then turns towards the left robot. Due to the missing overlap of the detection areas, the left robot is within human reach before the left laser scanner field can trigger a safety stop (or vice versa, from left to right).

Scenario 5-C

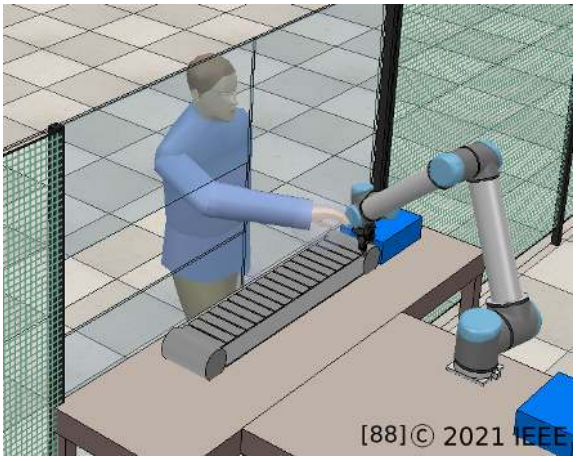


Figure A.3.: The agent reaches through the feed opening into the robot's path, provoking a collision.
Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Task and layout: A Universal Robots UR10 robot operates on a table with a conveyor belt. Both robot and conveyor belt are behind a plexiglass safety barrier. The safety barrier has a feed opening through which human workers can put workpieces on a conveyor belt.

Safety measures: The safety barrier is the main safety measure. It is intended to avoid human-robot collisions. No further safety measures are in place.

Safety flaw/hazard: The safety distance between the feed opening and the robot is insufficient, allowing the human worker to reach into the robot's path, which results in a critical collision that exceeds collision force limits (see Figure A.3).

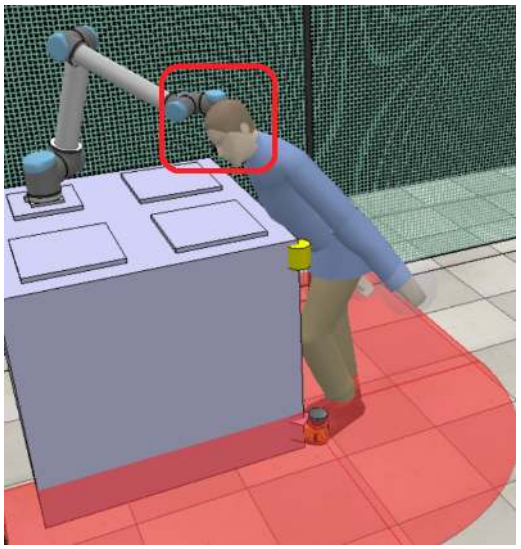


Figure A.4.: The agent leans forward into the robot's path, resulting in a critical collision with the head.
Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Scenario 5-D

Task and layout: A Universal Robots UR10 robot performs a pick-and-place task on a table. The table is closed off on two sides by a safety fence. The remaining two sides are open.

Safety measures: A laser scanner monitors the open sides of the robot cell. A reduction of the robot velocity is triggered when a human approaches. With the reduction, collision forces are sufficiently reduced so that collisions with arms and upper body are uncritical.

Safety flaw/hazard: Due to the height of the table, the robot can collide not only with arms and upper body, but also with the head (compare Section 2.1.5). Due to more restrictive collision force limits for the head, a head collision exceeds the threshold despite the reduced robot velocity (see Figure A.4).

Scenario 5-E

Task and layout: A Universal Robots UR10 works in an enclosed cell which is accessible through a light curtain.

Safety measures: The light curtain is intended to trigger a safety stop of the robot as soon as the agent enters the cell.

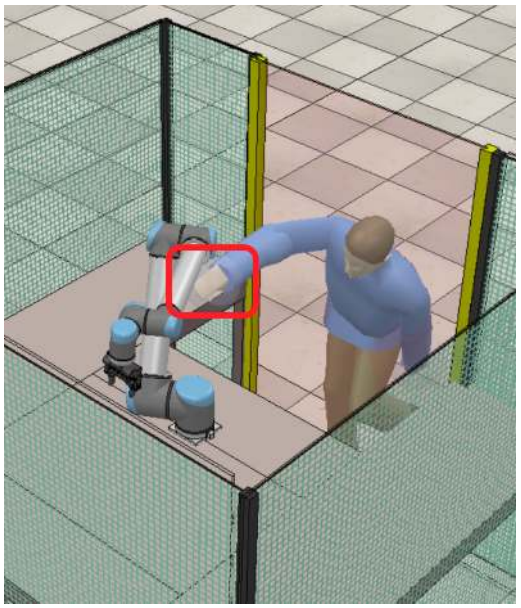


Figure A.5.: Due to a prolonged stopping time, the agent can reach the robot's elbow joint before the robot has slowed down sufficiently to avoid critical collision forces.
Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Safety flaw/hazard: The combined response time of the laser scanner and the robot stopping time (i.e., the total time that passes from the entering of the worker into the cell until the robot has reached a safely low velocity) is too high. A collision hazard may be present, depending on the current robot position of the robot and the point in time when the worker enters the cell. If the robot is in a certain position, then the robot's elbow joint protrudes far enough so that a collision is possible before the robot has slowed down sufficiently (see Figure A.5).

Scenario 5-F

Task and layout: A KUKA iiwa robot is placed in a workcell and separated from the agent by a light curtain. To grasp a workpiece, the robot temporarily moves outside the workcell and then takes the workpiece inside.

Safety measures: Inside the cell, where protection by the light curtain is given, the robot works with full velocity. When reaching outside the cell, the velocity is limited to keep collision forces below the acceptable force limits.

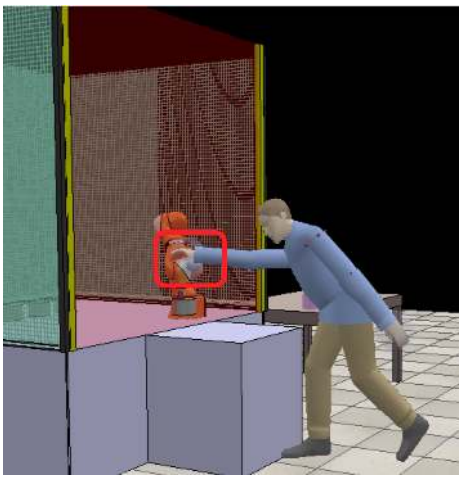


Figure A.6.: The agent reaches for the robot after it has picked up the workpiece, but before it has retreated behind the light curtain.
Figure from [91]. ©2021 IEEE. Reprinted with permission from IEEE.

Safety flaw/hazard: The collision forces are only sufficiently low until the robot has picked up the workpiece. As soon as the robot has grasped the workpiece, its mass is added to the effective robot mass, which increases the collision forces above the critical threshold (compare Eq. (2.4)). Thus, a critical collision is possible if the worker approaches the robot with such a timing that the workpiece has already been picked up, but the robot has not yet fully retreated behind the light curtain.

A.2. Experiments from Chapter 6

The descriptions and images in this subsection are taken from [93] (with the exception of figures denoted as "author's own figure").

Scenario 6-A

Scenario 6-A has already been described in Examples 7 - 9 (see sections 6.1-6.3). For the sake of completeness, the explanation is briefly repeated below.

Layout and Task: A figure of the cell layout is shown in Example 7 (see section 6.1). The task proceeds as follows: The human worker walks from the center area to the storage table on the left, where he retrieves a part from the table. The worker then walks back to the center and places the

Action	Explanation
t_1	Walking between area A and B
t_2	Walking between area A and C
u_S	Picking up the part at the storage table
d_S	Putting down the part at the storage table
u_S	Picking up the part at the robot station
d_S	Putting down the part at the robot station
b_1	Pressing button to activate robot
b_2	Pressing button to activate robot in safety override mode
b_3	Pressing button to stop robot
r	Retracting hand after reaching motion

Table A.1.: Agent action space in scenario 6A

part in front of the robot. The worker then walks back to the center where he activates the robot at the control panel. The robot performs a procedure (e.g., drilling) on the part until it is stopped by the worker at the control panel. Finally, the worker retrieves the part from the robot station and places it back onto the storage table.

Agent Model: The agent's discrete action space is given by table A.1.

Constraints on the agent's behavior are expressed by the EFA from Example 9. The SUPREMICA implementation of this EFA is shown in Figure A.7:

The continuous parameters of the agent behavior are defined as follows:

$$\underline{\theta} = (v_H, \Delta x, \Delta y) \tag{9.5}$$

where v_H denotes the human worker's walking speed, and $\Delta x, \Delta y$ denotes the longitudinal and lateral displacement of the position where the worker puts down the workpiece.

Safety Measures: A safety laser scanner is used to monitor the area around the robot station and trigger a safety stop if the worker approaches the robot station. The safety zone within which a safety stop is triggered is highlighted by the red area in the figure. For inspection purposes, a safety override mode can be triggered by the worker, which overrides the safety stop and allows the worker to inspect the processing of the workpiece from close distance (note: from a functional safety perspective, such a override mode should be of course avoided. Here it was only introduced to create additional hazards for the sake of the experiment).

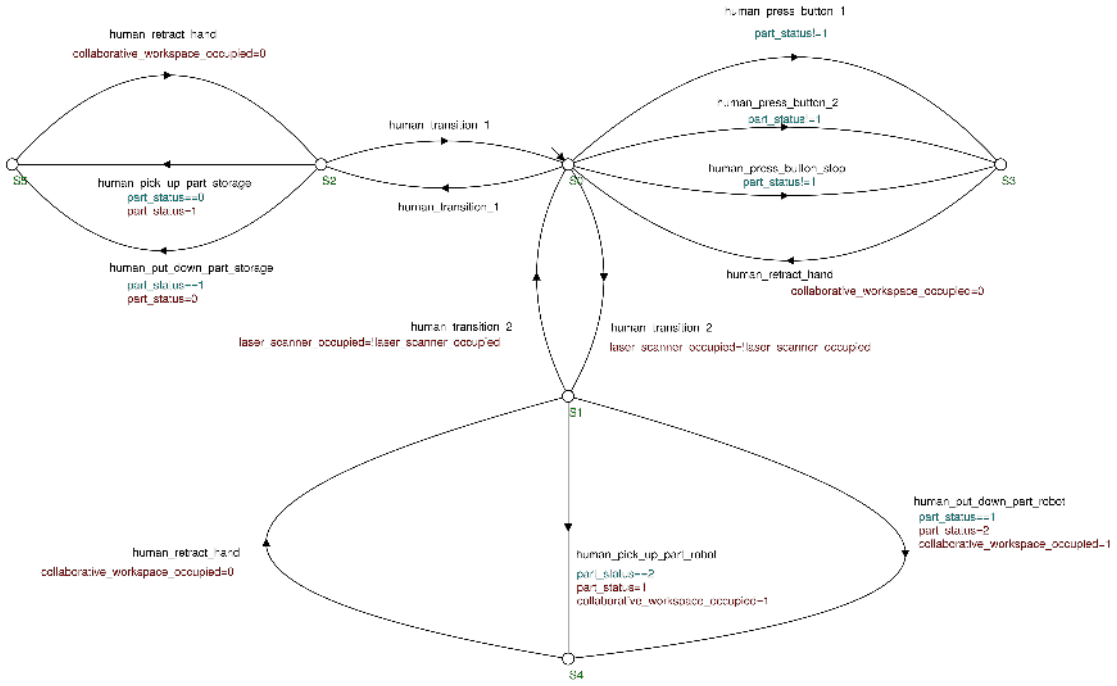


Figure A.7.:
Supremica implementation of the agent EFA from scenario 6-A. (Author's own figure.)

Safety Flaws/Hazards: Possible hazards in this scenario are (1) misuse of the safety override button by the agent (e.g., starting the robot in safety override mode and later inserting/removing a workpiece without reverting back to safe mode) and (2) fast collisions due to prolonged stopping time of the robot combined with fast approaches of the agent.

Scenario 6-B and 6-C

Scenarios 6-B and 6-C share the same task, and therefore also the same action space, parameters, and EFA models as given below. The difference between both scenarios is only in the safety measures.

Layout and Task: The robot cell layout is shown in Figure A.8. This scenarios has already been explained in Chapter six. The explanation is repeated for the sake of completeness. In this scenario human and robot collaboratively assemble a gearbox. In a normal assembly sequence, the worker first retrieves parts (bearings) from a shelf (A). One of the parts is inserted into the housing of the gearbox (B). Afterwards, the worker presses a button to activate the robot which then moves over the housing and presses the gearwheel (E) into the housing in a downward motion.

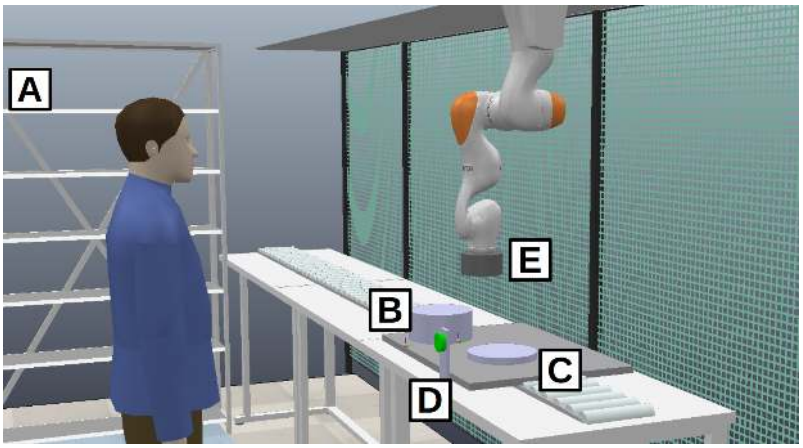


Figure A.8.: Cell layout in scenario B. Figure from [93]. ©2023 IEEE. Reprinted with permission from IEEE.

Action	Explanation
t	Walk between area A and B
r_P	Reach for parts from shelf
r_H	Reach into housing
r_C	Reach into cover
p_B	Press button (activate robot)
r	Retract hands (after reaching motion)
m_C	Mount cover on housing

Table A.2.: Agent's action space from Scenario 6-B and 6-C.

Meanwhile, the worker inserts the second bearing into the cover (C). Finally, the worker mounts the cover onto the housing.

Agent Model: The agent's discrete action space is given by table A.9. The SUPREMICA implementation of the agent EFA is shown in Figure A.9

Safety Measures: This is the only aspect where scenarios 6-B and 6-C differ. While scenario 6-B has no particular safety measures despite a limitation of the robot's speed, scenario 6-C features an additional fence and light curtain in front of the workstation (see Figure 6.1b). This light curtain detects the human hand and trigger a safety stop of the robot. However, there is a certain delay associated to the safety stop due to the latency of the light curtain and the stopping time required by the robot.

Safety Flaw/Hazard: Hazards in this scenario include potential collisions between the human hand and the robot, or the possibility of the human hand getting trapped between gearwheel and housing. Both situations can appear if the agent deviates from the nominal workflow (e.g., by switching

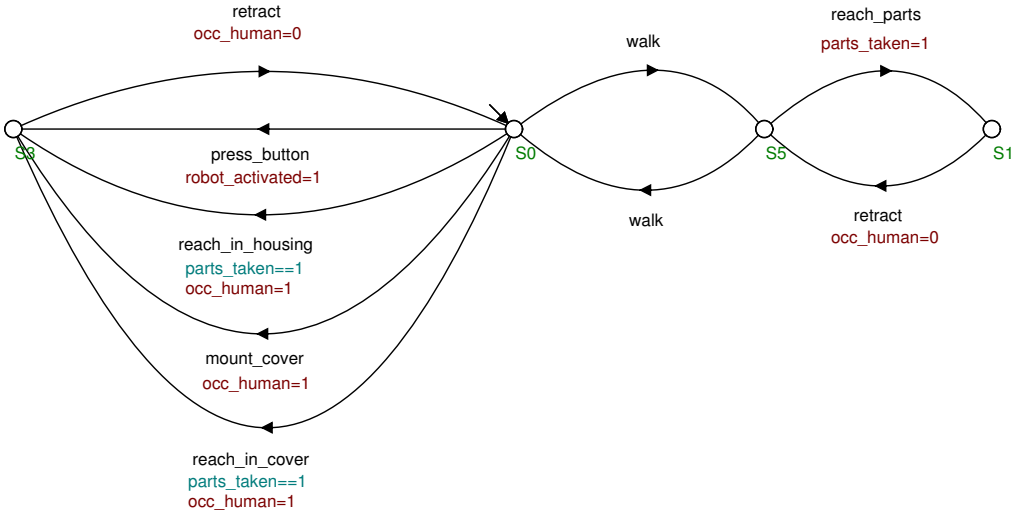


Figure A.9.: SUPREMICA implementation of the agent EFA in scenario B and C. (Author's own figure.)

the order of assembly actions such that the agent reaches into the housing while the robot is inserting the gearwheel). Furthermore, a collision between the robot and the human head is possible, if the human leans to far forward over the table while reaching into the housing or mounting the cover.

A.3. Experiments from Chapter 7

The experiments in chapter 7 are based on the same scenarios from chapter 6 (i.e., scenarios 6-A, 6-B and 6-C). The difference is that the agent EFA model is extended by further EFA models for SUT and safety specification. These additional models are shown below.

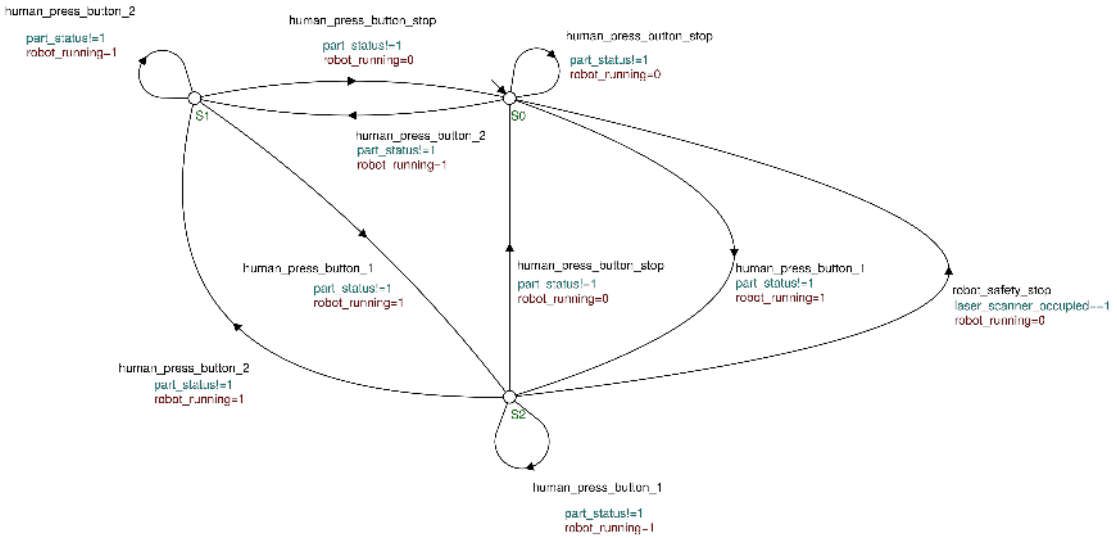


Figure A.10.: EFA model of the SUT from scenario 6-A. (Author's own figure.)

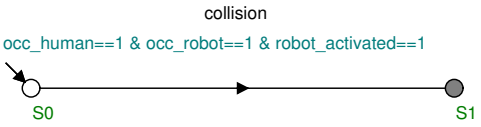
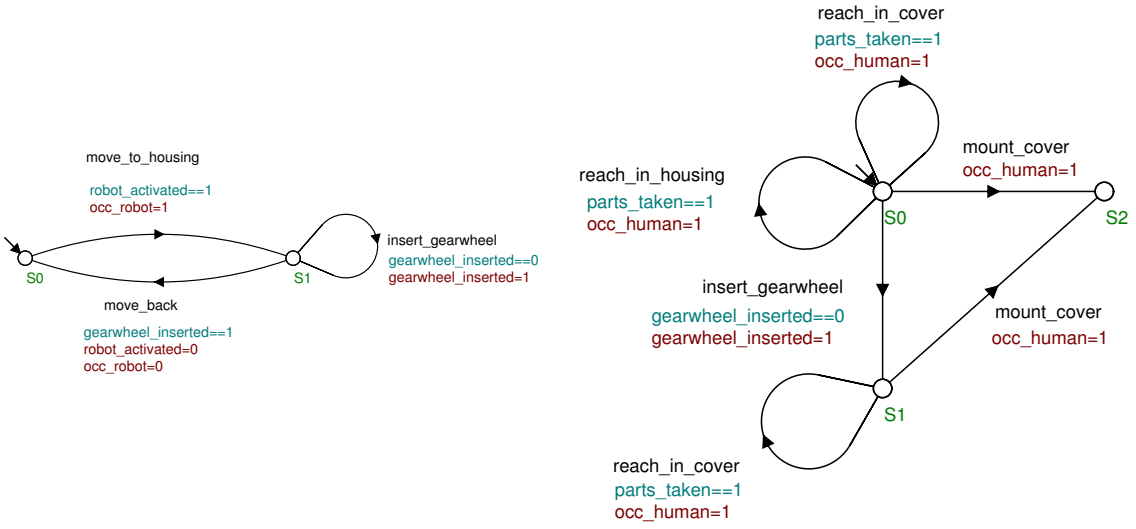


Figure A.11.: EFA model of the safety specification from scenario A. (Author's own figure.)



B. Algorithms

The standard MCTS algorithm was already introduced in Chapter 2. In this book, the algorithm is used in slightly adapted forms. Below, Al-

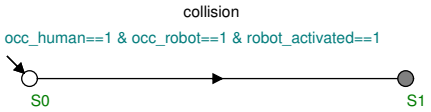


Figure A.13.: EFA model of the safety specification from scenario 6-B and 6-C. (Author's own figure.)

gorithm 2 shows pseudocode for the adaptation of MCTS for risk-guided search (see Chapter 5), and Algorithm 3 shows pseudocode for the adaptation of MCTS for automata-constrained risk-guided search (see Chapter 6). Plase note that these algorithms are only intended to explain the principles and that the actual implementation may deviate from the pseudocode (for details, the reader is referred to the further publications that have been referenced in the respective chapters).

Algorithm 2 MCTS Adaptation for Risk-Guided Search (unconstrained)

unsafeSequences = { };
while not simulationBudgetExceeded **do**
    currentSequence = { }; currentNode = rootNode;
    simulation.init(); // set simulator to initial state;
    while fullyExpanded(currentNode) **do**
        // Traverse tree until reaching a node that is not fully expanded
        currentNode, action = SELECTBESTCHILD(currentNode)
        simulation.step(action);// update simulation state;
        currentSequence.append(action);
    end while
    currentNode, action = EXPAND(currentNode);
    simulation.step(action);
    currentSequence.append(action)
    while not length(currentSequence) == n **do**
        // Perform rollout and backpropagate reward;
        action = randomSample(A);
        simulation.step(action);
        currentSequence.append(action);
    end while
    reward = sim.getRisk();
    BACKPROPAGATION(currentNode, reward);
    if reward > riskThreshold **then**
        unsafeSequences.append(currentSequence);
    end if
end while
function SELECTBESTCHILD(currentNode)
    // iterate over children of current node, return the one maximizing the UCT criterion
    bestChildNode = argmax $\left(\frac{\text{node}.q}{\text{node}.n} + c\sqrt{\cdot \frac{log(\text{parentNode}.n)}{\text{childNode}.n}} \right)$;
    bestAction = bestChildeNode.incomingAction // action leading to child node;
    return bestChildNode, bestAction;
end function

function EXPAND(currentNode)
    a = randomSample(A);
    childNode = Node(parent=currentNode, incomingAction = a, q = 0, n = 0);
    currentNode.child = childNode;
    return childNode, a;
end function

function BACKPROPAGATION(currentNode,reward)
    currentNode.$n \leftarrow$ currentNode.$n + 1$;
    currentNode.$q \leftarrow$ currentNode.q+reward;
    if currentNode == rootNode **then**
        return;
    else
        BACKPROPAGATION(currentNode.parent, reward);
    end if
end function

Algorithm 3 MCTS Adapatation for Automata-Constrained Risk-Guided Search

hazardousSequences = { };
while not simulationBudgetExceeded **do**
    currentSequence = { }; currentNode = rootNode;
    $\underline{\theta}$ = randomSample($\underline{\theta}_{min}$, $\underline{\theta}_{max}$);
    simulation.init($\underline{\theta}$); automaton.init(); // set simulator and automaton to initial state;
    while fullyExpanded(currentNode) **do**// Tree traversal
        currentNode, action = SELECTBESTCHILD(currentNode)
        simulation.step(action); automaton.transition(action); // update states
        currentSequence.append(action);
    end while
    currentNode, action = EXPAND(currentNode);
    simulation.step(action); automaton.step(action);
    curentSequence.append(action)
    while not length(currentSequence) == n **do**// Rollout/backpropagation reward
        feasibleActions = automaton.getFeasibleActions();
        action = randomSample(feasibleActions);
        simulation.step(action); automaton.step(action);
        curentSequence.append(action);
    end while
    reward = sim.getRisk();
    BACKPROPAGATION(currentNode, reward);
    if reward > riskThreshold **then**
        hazardousSequences.append(currentSequence);
    end if
end while

function SELECTBESTCHILD(currentNode)// find node maximizing UCT criterion
    bestChildNode = argmax $\left(\frac{\text{node}.q}{\text{node}.n} + c\sqrt{\cdot \frac{log(\text{parentNode}.n)}{\text{childNode}.n}} \right)$;
    bestAction = bestChildeNode.incomingAction // action leading to child node;
    return bestChildNode, bestAction;
end function

function EXPAND(currentNode)
    feasibleActions = automaton.getFeasibleActions();
    a = randomSample(feasibleActions);
    childNode = Node(parent=currentNode, incomingAction = a, q = 0, n = 0);
    currentNode.child = childNode;
    return childNode, a;
end function

```
40:
41: function BACKPROPAGATION(currentNode,reward)
42:     currentNode.n ← currentNode.n + 1;
43:     currentNode.q ← currentNode.q+reward;
44:     if currentNode == rootNode then
45:         return;
46:     else
47:         BACKPROPAGATION(currentNode.parent, reward);
48:     end if
49: end function
```

C. Modeling of HRC Systems with Supremica

In chapters 6 and 7, a detailed discussion of the modeling procedure for obtaining EFA models was omitted for reasons of brevity. Instead, it was assumed that a model is already given. This part of the appendix discusses the modeling procedure in more detail and aims to provide some basic modeling guidelines. However, it should be emphasized that the procedure presented here is only one possible way of modeling HRC systems. Several aspects of the procedure such as the desired level of detail or the model granularity are decisions which are to be made by the user on a case-by-case basis and cannot be prescribed here in general terms. Furthermore, note that EFA modeling involves a certain redundancy. Recall that the state-space of an EFA is given by the cartesian product of the set of locations and the variable space:

$$S = L \times V$$

Thus, aspects of the system state can be expressed either by introducing variables or locations, or through a combination of both. Although some general guidelines are given below, it is ultimately a decision of the user how much they want to rely on locations or variables to express system states. Thus, depending on modeling styles, there may be multiple different models expressing the same behavior.

C.1. Modeling

General Modeling Approach

The hazard analysis problem is modeled by three components:

- The *system under test* (SUT) models the behavior of the system whose safety is to be analyzed (e.g., a collaborative robot system).
- The *agent* models the behavior of the entity/entities which are interacting with the SUT (e.g., a human that is collaborating with the robot, or another robot in a multi-robot system).
- The *safety specification* classifies if a given state of the joint model of agent and SUT is safe or unsafe.

Each model component is represented by an EFA, where $\mathcal{A}$ denotes the EFA of the agent, $\mathcal{SUT}$ the EFA of the SUT, and $\mathcal{SP}$ the EFA of the safety specification. Note that $\mathcal{A}$ and $\mathcal{SUT}$ encode the actual behavior of the system, while $\mathcal{SP}$ is an abstraction of a safety specification which may be given in a textual or other form.

Modeling of Agent and SUT

The behavior of agent and SUT are described by sequences of events from the respective EFA models. A given behavior is encoded as a sequence of events. The set of feasible event sequences is thus restricted by the structure of the EFA. As discussed above, in contrast to ordinary Finite Automata (FA) which only feature states but no variables, the state-space of EFA is given by the cartesian product of the set of possible locations and variables assignments. It is therefore difficult start with EFA modeling by defining the state-space from the outset.

Instead, it is more intuitive to approach the modeling problem from the perspective of the action- rather than the state-space. The modeler should first define an action space which is suitable to describe the behavior of Agent or SUT, respectively. Then, the modeler should reason about the feasibility of actions and action sequences under certain conditions and define a location- and transitions-structure which expresses these constraints. Afterwards, additional constraints which are not captured by the location/-transition structures of the submodels can be expressed by introducing variables, guards, and update rules. With this approach, the state-space does not have to be defined explicitly from the outset, but follows implicitly from the locations and variables that are introduced along the way. More specifically, the modeling could proceed as follows:

1. **Define the event spaces** (i.e., the sets of possible events) for agent and SUT, respect-ively: What events can occur? What activities can

the agent and SUT perform? Decide if activities should be assumed as instantaneous, or if they should extend over time. If an activity is assumed as instantaneous, it is modeled by a single event. For activities which extend over time, two events are defined: One for starting, and one for ending the activity. Note that the sets of events for agent and SUT need not be disjoint. If a certain event affects the behavior of both agent and SUT, it may appear in both event spaces. This is called a *shared event*.

2. **Build a location/transition structure.** The following procedure should be performed separately for Agent and SUT. In principle, it does not matter whether one starts with modeling the agent or the SUT. We generally recommended to start with the agent, since the agent is often the pro-active part of a system and the SUT the re-active part of the system (compare the discussions in Chapter 4). In other words, the agent's behavior will generally depend less on the SUT behavior than vice-versa. Hence, it may be easier to start with modeling the agent.

   a) Start with the initial location. This represents the system's initial state.

   b) Look at the event space and select those events that are feasible in the current location. The subset of currently feasible events shall be denoted by Σ_C.

   c) For each event $e_i \in \Sigma_C$, do the following:

      - Create a transition labeled by e_i which starts at the current location.

      - Think about what events are feasible after performing e_i: If the occurrence of e_i does not change Σ_C, let the transition point to the current location (i.e., create a self-loop). If performing e_i changes Σ_C, then introduce a new location and let the transition point to that new location.

   d) Go to each new created location that was created in the previous step and repeat the procedure starting from step 2b).

   By performing above procedure, the location/transition structure will gradually ex-pand. The expansion is complete when all locations are fully expanded, i.e., when there is no location left from which new

transitions can be added. In some cases, the expansion of the location/transition structure may lead to redundant locations. Two locations are redundant if the have exactly the same set of outgoing transitions. Redundant locations can be consolidated into one location to limit the number of locations and make models more compact. This consolidation, however, is optional. In some cases, the unconsolidated structure may be more intuitive and easier to read. This should be decided on a case-by-case basis.

3. **Model additional dependencies between agent and SUT.** So far, the location/transition structures of agent and SUT have been modeled separately. In collaborative systems, however, there are dependencies between the behaviors of agent and SUT. Some of these dependencies may already be captured through shared events. However, there may also be additional inter-dependencies and/or constraints which are not yet captured in the model. These can be expressed by introducing additional variables and guards. The procedure for introducing such constraints is as follows:

   a) For each action in the action space, determine if the action is subject to additional constraints which are not captured by the location/transition structure.

   b) For each event that is subject to additional constraints, do the following:

      - Determine the condition on which the feasibility of the event depends. Introduce new variables v_i into the variable space V to track the information that is needed to decide the condition (if they are not already represented in V). Note: You may only introduce integer variables of finite range.
      - Write the condition as a logical formula over the previously introduced variables and add this formula as a guard to the transitions which are associated with the respective event. The event can now only be executed if the formula holds true.
      - Determine for each event if the execution of the respective event affects the values of the previously introduced variables. If this is the case, create an according update rule for this event. The update rule will be executed each time the event is executed.

Modeling of Safety Specifications

Safety specifications are usually given in textual form. These textual specifications need to be formalized. To that end, they are translated into EFA form. The EFA representing the safety specification shall be denoted by $\mathcal{SP}$. This EFA uses the notion of marked and unmarked states to differentiate between safe and unsafe states: *safe* states are associated with *un*marked states, whereas *unsafe* states are associated with *marked* states.

To build $\mathcal{SP}$, first create an initial location. The initial location is unmarked. Next, translate the textual safety specification into a logical formula over the variable space. This formula should evaluate to *true* if the safety specification is violated, and to *false* if the safety condition is not violated. If the variables introduced in the previous steps are not sufficient to express the safety specification, intrdouce additional variables introduced (and do not forget to introduce appropriate update rules for the newly introduced variables as well).

Next, create a marked location. This marked location represents an unsafe system state. Create a transition from the initial (unmarked) location to the marked location. This transition represents the occurrence of an unsafe event. Finally, create a guard statement for the newly introduced transition. The guard statement should correspond with the logical formula from the safety specification. Hence, the transition that represents the occurence of an unsafe event may only occur if the guard statement evaluates to true (and, consequently, if the safety specification is violated). If there are multiple unsafe conditions, one can introduce a marked location and corresponding transition for each condition. Although it would be also possible to have a single marked location and transition that is guarded by a logical disjuction of the multiple criteria, having separate locations/transitions may be easier to read.

The result is an EFA with one unmarked location and one or multiple marked locations where transitions from unmarked to marked locations can only occur if the specification is violated. For now, this may seem like a strangely convoluted way of expressing the safety specification. However, this structure has the advantage that it enables a safety analysis based on supervisor synthesis, which is described in more detail later.

Remarks

Please note the following remarks on the procedure presented above:

- **Refine iteratively.** it is important to note that a manual modeling procedure such as the one presented above is naturally error-prone. Furthermore, it may not be possible to find suitable agents spaces and model structures in the first go. Retroactive changes may be necessary (e.g. the modeler may decide to introduce new events at a later stage after defining the initial action space). Therefore, the model should be refined iteratively by repeating above procedure until the model is consistent.
- **Beware of unintentional constraints.** A possible source of errors is that modelers may unintentionally restrict the model's behavior in ways they are not aware of. This is particularly critical from a safety analysis point of view, since the restrictions can lead to potentially safety-critical behaviors being omitted from the model. Typical errors which can lead to unintentional restrictions are, for instance:
  - An outgoing transition is erroneously omitted from a location. It is therefore important to carefully think about feasibility of all events in step 2b). Errors of this kind can also occur if an event which has previously not been part of the event space is added retroactively to a location of a model without checking if that event should also be feasible in other locations of that same model.
  - Variables are not updated properly, or update rules are forgotten. This can lead to guard statements evaluating to *false* although they should actually be *true*, thus preventing the execution of a transition that should actually be feasible. To avoid this, it is necessary to carry out step 3b) of the procedure particularly carefully.
- **Consider introducing additional variables.** Recall that there is a certain redundan-cy in EFA as a modeling formalism. More specifically, constraints on a system's behavior may be expressed either through the location/transition structure, or through variables, guards, and update rules. Of course, it is difficult to prescribe explicitly when one should use a location and when one should use a variable instead. As a general rule, however, the location/transition structure

is easier to read and more intuitive for modelers to understand than the interplay between variables, guards, and update rules. Therefore, the procedure above uses the transition/location structure as a primary modeling formalism. Variables, guards, and update rules are then added in a secondary modeling step after building the location/-transition structure. In some cases, however, the location/transition structure can be come very extensive. To simplify the structure, modelers may consider to remove some locations and instead introduce additional variables to capture particular aspects of the state-space.

- **Consider a modular approach:** modularity can also be a helpful tool to mitigate complexity in the modeling procedure. In the procedure above, the system is broken down into interacting subcomponents, namely agent and SUT. This modular approach can be taken even further by breaking down the models of agent and SUT into further sub-components, which can be modeled using the same principles as described above. For instance, if the SUT consists of several robots, an EFA for each robot can be introduced. In some cases, it can also be useful to introduce EFA models for passive components such as workpieces. Although these components do not actively create events, their configuration can have an impact on the feasibility of events in other submodels. For instance, in an assembly task, the current configuration of a workpiece changes with the occurrence of certain worksteps. The configu-ration of the workpiece, in turn, may influence what worksteps (events) are currently feasible. Instead of attempting to encode this in the agent and SUT EFA models, it may be worthwhile to maintain a separate EFA for the workpiece (consider also the following example).

- **Avoid marked locations in the agent and SUT models:** The safety specification should be the only of the three EFA for which marked locations are defined. The reason for this is that in the analysis procedure described below, the notion of marked and unmarked locations is used to differentiate between safe and unsafe states. To keep the model clear and readable, and to avoid accidental misclassification of safe or unsafe states, the safety specification should be the only EFA that introduces marked states into the model, whereas the other EFA should describe what system behaviors are possible, but remain agnostic with respect to the safety of these behaviors.

C.2. Synthesis

The final step is the synthesis. In this step, we use methods from supervisory control theory (SCT) – more specifically, supervisor synthesis – to identify possible system behaviors that violate the safety specification.

The system as a whole is described by $\mathcal{A} \parallel \mathcal{SUT}$, that is, the synchronization of the two submodels. Note that $\mathcal{A} \parallel \mathcal{SUT}$ in itself is also an automaton whose event space is the union of both submodel's event spaces. A sequence of events resulting from feasible transitions in this automaton thus represents one possible behavior of the system. Consequently, the language $L(\mathcal{A} \parallel \mathcal{SUT})$ of the automaton (i.e., the set of all feasible event sequences) represents all possible system behaviors. The challenge is now to extract from $L(\mathcal{A} \parallel \mathcal{SUT})$ the subset of behaviors that violate the safety specification. This is where supervisor synthesis is applied: Supervisor synthesis restricts the behavior of a controlled system (called *plant*) in such a way that it complies to a specification. The resulting supervisor is *minimally restrictive* and *non-blocking*. In this context, *non-blocking* means that all resulting behaviors will always reach a marked state. We can therefore synthesize unsafe behaviors by performing a supervisor synthesis for the system $\mathcal{A} \parallel \mathcal{SUT}$ with respect to the specification $\mathcal{SP}$. Since we modeled the safety specification in such a way that the unsafe states are marked states, the synthesized behaviors will always result in an unsafe state, whereas safe behaviors that do not result in marked states will be discarded by the synthesis. *Minimally restrictive* means that the synthesis only discards the blocking behaviors, but no more than that. Therefore, the set of remaining behaviors is complete w.r.t the unsafe behaviors. The result of performing a supervisor synthesis in SUPREMICA is another automaton, whose language $L(\mathcal{K})$ corresponds to the unsafe system behaviors. This automaton can be easily exported and parsed to extract the unsafe event sequences.

C.3. Example

The following example illustrates the modeling procedure. Scenario 6-B from Section 7.5 is chosen as an exemplary scenario to be modeled.

Agent and SUT Model

We start by modeling the agent. According to step 1, we first define the event space of the agent, which corresponds to the action space of the human worker (i.e., the set of all actions that the human worker can perform in the collaborative assembly procedure):

$$\{walk, press\_button, retrieve\_parts, reach\_housing, reach\_cover, mount\_cover, retract\}$$

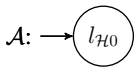
here, $walk$ denotes walking between the workplace and the shelf where the parts are stored, p_B is pressing the button for activating the robot, $reach\_housing$ and $reach\_cover$ represent reaching into the housing and cover to mount a part, respectively, $mount\_cover$ denotes mounting the cover onto the housing[1]. Finally, $retract$ denotes retracting the hands after a reaching motion has been completed. Recall that, according to the guideline, an action which does not happen instantaneously should be modeled not only by an event for initializing the action, but also one for ending the action. This is also the case here: the event $retract$ denotes retracting the hand at the end of a reaching and therefore represents the end of the various actions which are initiated by the following events:

$$retrieve\_parts, reach\_housing, reach\_cover, mount\_cover$$

Note that there need not be a dedicated end-event for each action. Instead, a single event can represent a termination for multiple actions if this makes sense in the particular use-case. In contrast, $walking$ is modeled as single instantaneous event (this is a deliberate modeling simplification since walking is not safety-critical in the context of this example).

[1] For reasons of simplicity in this example we only consider one-directional assembly (i.e., no disassembly or un-doing of assembly steps)

We proceed with step 2: building the location/transition structure. According to step 2a), we create an initial location l_{H0}, which represents the worker being located in front of the robot station:

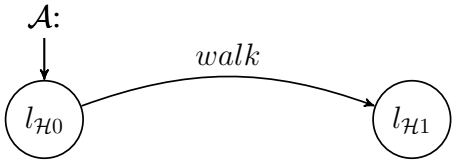


In this initial location, the worker is idle (i.e., not performing any activities). Next, according to step 2b), we define the set of feasible events in this state:

$$\{walk, press\_button, reach\_housing, reach\_cover, mount\_cover\}$$

meaning that the worker can walk away from the station, press the button, or perform the various reaching motions to manipulate the workpiece. The event $retrieve\_parts$ is infeasible since the worker is not located in front of the shelf, and $retract$ is infeasible because the worker has not yet started any reaching motion.

Now, we proceed with 2c): For each event in the feasible set, we determine if the execution of that event impacts the feasible set. We start with the $walk$ event: Clearly, executing $walk$-event impacts the feasible set. Once the worker has walked to the shelf, the button and workpiece are no longer reachable and the related events become infeasible. However, the event r_P (retrieve parts from the shelf) becomes feasible. Since the event affects the feasible set, a transition to a new location is created:

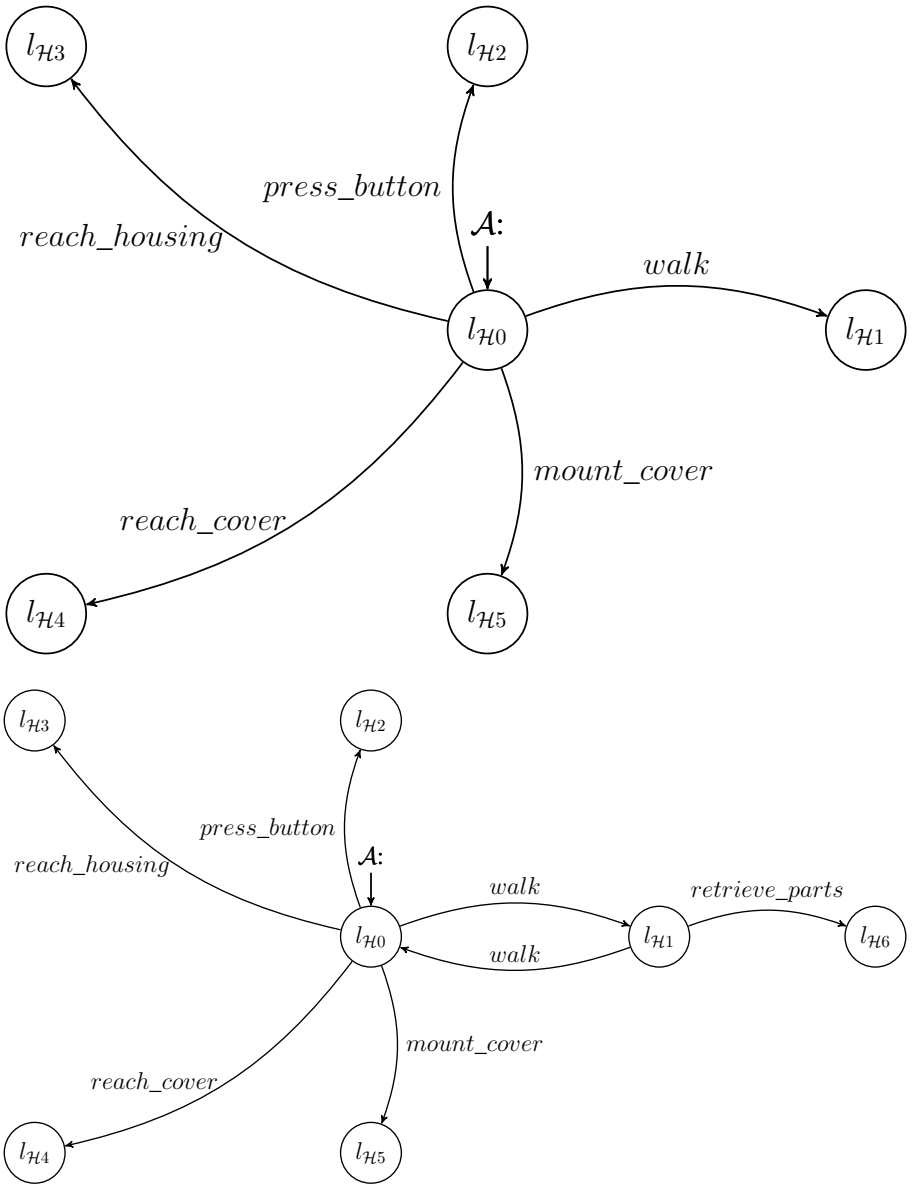


The same is repeated also for the other events in the feasible set. After fully expanding the initial state, the model looks as follows:

We now repeat the expansion for each newly created location, starting at location $l_{\mathcal{H}\infty}$: After walking from the robot station to the shelf, the worker can reach for parts from the shelve or walk back. Hence, the feasible set here in this location is:

$$\{retrieve\_parts, walk\}$$

With the newly created transitions, the structure looks as follows:



By performing the expansion procedure also for the remaining nodes, we finally arrive at the following location/transition structure:

Note that it is possible to consolidate the structure: As noted above, multiple locations can be consolidated into one if they have exactly the same outgoing transitions. Here, this is the case for the locations l_{H2} to l_{H5}. Therefore, we can create a single location l'_{H2} that consolidates $l_{H2} \ldots l_{H5}$ (see Figure C.15).

Next, we repeat steps 1 and 2 of the modeling procedure for the SUT. In this case, we decompose the SUT into two further models: One for the robot, and one for the workpiece (the latter being important because the assembly state of the workpiece influences the feasible events for both worker and robot). For brevity, we do not show the whole modeling pro-

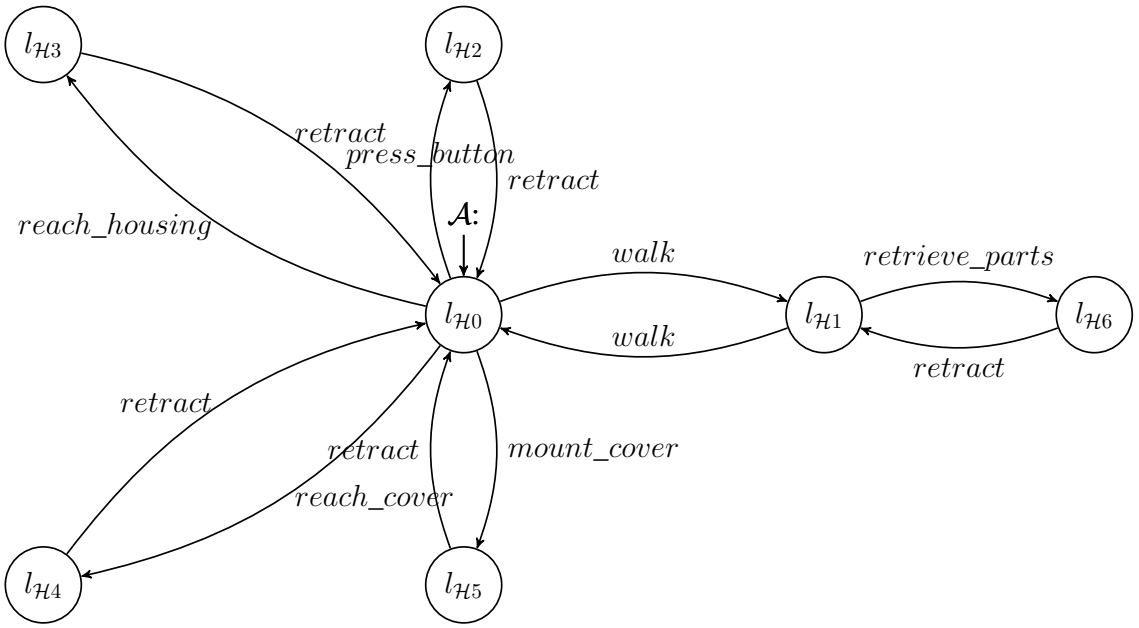


Figure C.14.: Location/transition structure of the agent model

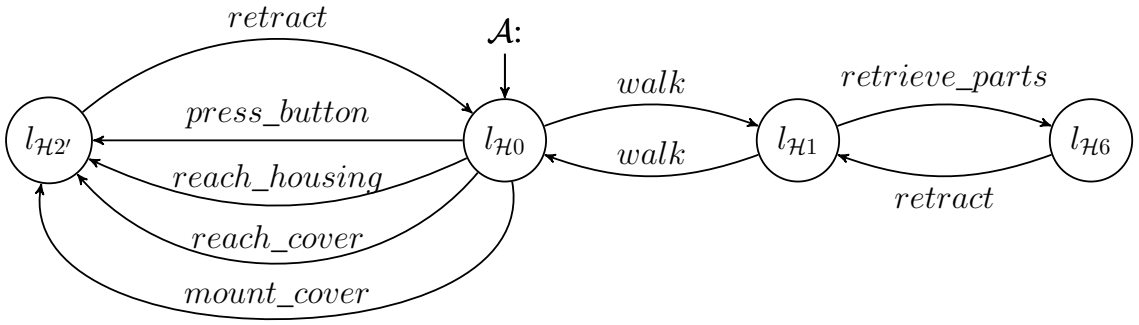
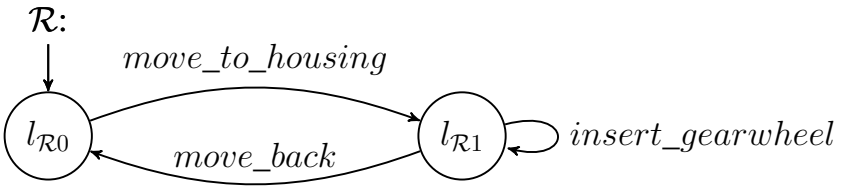


Figure C.15.: Consolidated location/transition structure of the agent model

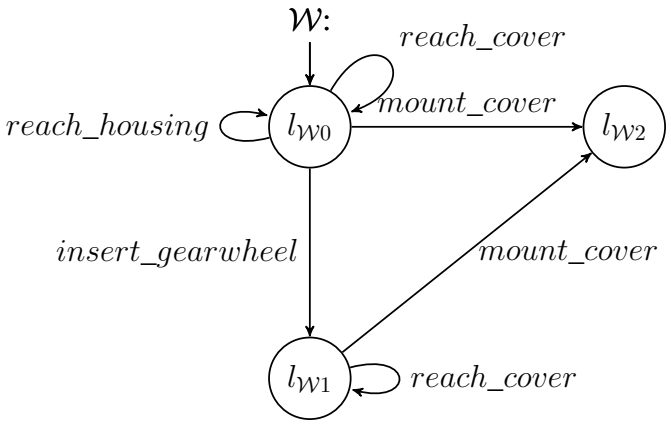
cedure step by step again and only give the resulting location/transition structures. Event space and structure of the robot model are as follows:

$$\{move\_to\_housing, insert\_gearwheel, move\_back\}$$



Here, the initial location represents the robot being in idle mode. From there, the robot can move to the housing of the workpiece. When located over the housing, the robot can insert a gearwheel into the housing and/or move back to idle mode (note that this structure does not enforce that the robot *must* insert a gearwheel before moving back, it may also move back without having inserted it).

The location/transition structure for the workpiece model is as follows:



Here, the initial location l_{W0} represents a state where the workpiece is not assembled (i.e., the gearwheel is not inserted into the housing and the cover is not mounted onto the housing). In this initial location, all assembly actions are feasible (i.e., the worker can reach into both housing and cover and mount the cover onto the housing, and the robot can insert the gearwheel into the housing). After inserting the gearwheel, however, the housing is blocked and the worker cannot reach into the housing anymore (see l_{W1}). After mounting the cover, no further assembly action is possible (see l_{W2}). With respect to the workpiece model, the following remarks should be pointed out:

- The workpiece model only features shared events which also appear the agent- and robot event spaces. It does not introduce new events, because the workpiece is a passive component which cannot change its state spontaneously. Thus, the model merely tracks the events from the other models and changes its state accordingly.

- Shared events can only occur if they are simultaneously feasible in all models in which they are part of the event space: Thus, by restricting the feasibility of assembly actions in the workpiece model,

we also restrict their feasibility in the agent- and robot model, respectively. Besides shared variables (which we will introduce in the next step), such shared events are one of the mechanisms by which dependencies between interactive systems is expressed.

- The workpiece model only models the *feasibility* of events. It does only restrict what sequences of events are feasible, but ot does not specify in any way in what order the assembly sequence should be carried out.

Finally, we arrive at step 3 of the procedure. In this step, we introduce variables, guards, and update rules to model additional dependencies between the submodels which are not yet captured by the location/transition structure. According to step 3a), we start by going through all events and determining for each event if it is subject to additional constraints that are not captured by the location/transition structure. According to step 3b we introduce variables, guards, and update rules as shown in the following table:

Event	**Variable(s)**	**Initial value**	**Guard(s)**	**Updated on event(s)**
$reach\_housing$	$parts\_taken \in \{0,1\}$	0	$parts\_taken == 1$	$retrieve\_parts$
$reach\_cover$	$parts\_taken \in \{0,1\}$	0	$parts\_taken == 1$	$retrieve\_parts$
$move\_to\_housing$	$robot\_activated \in 0,1$	0	$robot\_activated == 1$	$press\_button$, $move\_back$
$move\_back$	$gearwheel\_inserted \in 0,1$	0	$gearwheel\_inserted == 1$	$insert\_gearwheel$

Table C.3.: Variables, guards, and update rules to capture additional dependencies

Additional constraints have been introduced for the following actions:

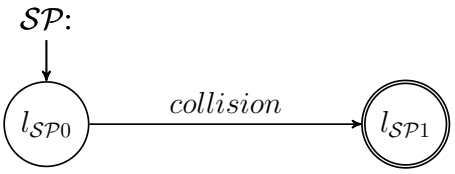
$$reach\_housing, reach\_cover, move\_to\_housing, move\_back$$

For $reach\_housing$ and $reach\_cover$ it is required that the worker first retrieves parts from the shelf (otherwise it would make no sense for the worker to reach into housing or cover because the worker does not have any parts to mount). Therefore, a variable is introduced to track if any parts have been taken. This variable is updated upon occurrence of the event $retrieve\_parts$. Furthermore, the event $move\_to\_housing$ requires that the robot is activated first. This is captured by a variable $robot\_activated$ which is set to 1 when the worker presses the activation button, and set

back to 0 when the robot has completed its procedure and moved back into idle mode. Finally, we want to enforce that the robot can only move back to idle mode after the gearwheel has been inserted. Thus, we introduce a variable $gearhweel\_inserted$ which is set to 1 when the gearwheel is inserted.

Safety Specification

Next, the safety specification $\mathcal{SP}$ is created. We start with an unmarked initial location to represent safe states, and a marked location to represent unsafe states. We assume that in our case, the safety specification specifies that no human-robot collision should occur. Thus, the transition from safe (unmarked) to unsafe (marked) is associated with a human robot collision, for which we create a new transition labeled $collision$.



The condition for a human-robot collision to occur is that both human and robot simultaneously occupy the collaborative workspace. To track the occupancy, we introduce two additional variables occ_human and occ_robot to track the occupancy of the collaborative workspace worker and robot, respectively. Finally, a guard statement is introduced, requiring that both worker and robot must occupy the collaborative workspace for the collision to occur:

Event	Variable(s)	Initial value	Guard(s)	Updated on event(s)
$collision$	$occ\_human \in \{0,1\}$, $occ\_robot \in \{0,1\}$	0	$occ\_human == 1$ $\wedge$ $occ\_robot == 1$	$reach\_housing, reach\_cover,$ $mount\_cover, retract$ $move\_to\_housing$ $move\_to\_housing$

Table C.4.: Additional variables, guards, and update rules for safety specification

The modeling procedure is now completed. The developed models are implemented with the software tool SUPREMICA. The SUPREMICA implementation of the resulting models is shown in Figures C.16-C.18.

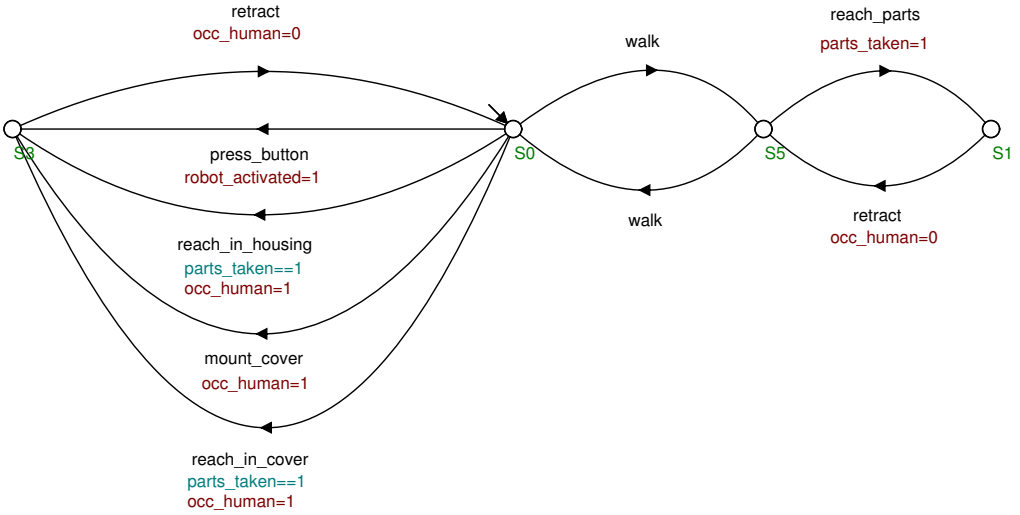


Figure C.16.: Agent model.

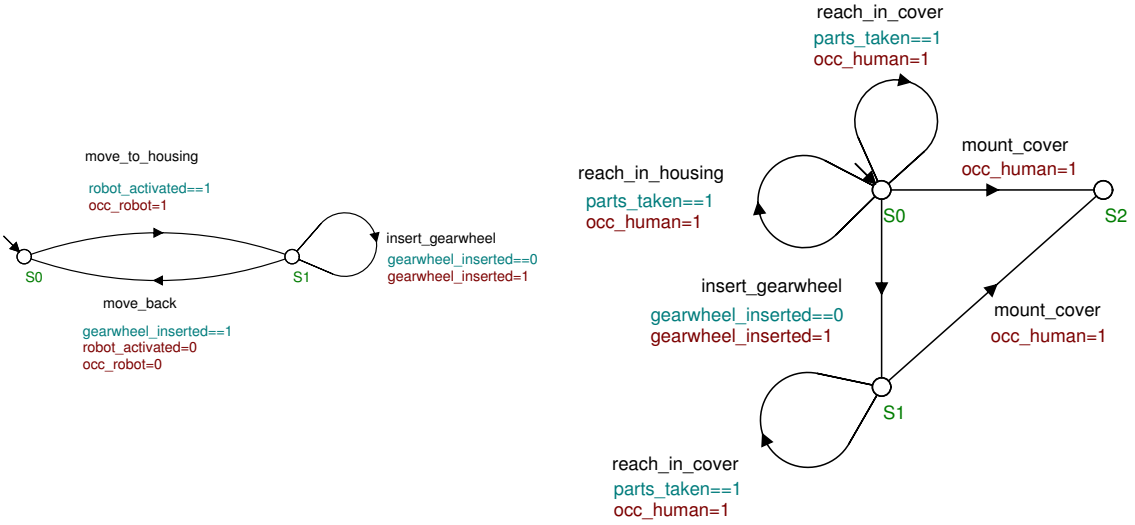


Figure C.17.: SUT model, consisting of the submodels robot (left) and workpiece (right).

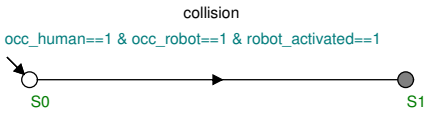


Figure C.18.: Safety Specification.

Synthesis

Finally, we perform the Synthesis in SUPREMICA. In order to do that, we go to the "Analyzer" section in the left upper pane, select all models (including the safety specification), right-click, and select "Synthesize". We perform a synthesis with default settings (see Figure C.19). Another

model named "Supervisor" will appear in the upper left pane. This is the supervisor automaton. By right-clicking on the supervisor and selecting "export", the automaton can be exported in XML format. The XML file can be easily parsed to extract the unsafe event sequences. Note, however, that the language of the automaton may be infinite. This can be the case if the automaton contains loops. Thus, the extraction of event sequences should be restricted to a subset of finite length. In some cases, it may be

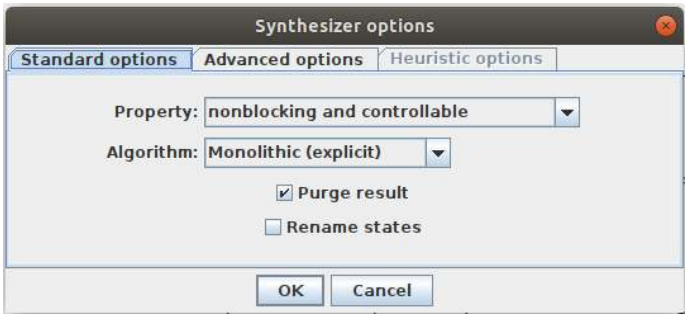


Figure C.19.: Settings for the supervisor synthesis in Supremica.

desirable to exclude certain events from the language (such as in Chapter 7, where we are only interested in events related to the agent). This can be achieved by hiding events before exporting the automaton (right-click on "Supervisor", then select "Hide events"). Note that the hiding of events may create substitute events (labeled "tau"). These can be removed by minimizing the automaton (right-click on "Supervisor", then select "Minimize"). Afterwards, perform the export procedure again. For instance, in the experiments in Chapter 7, the events related to the robot were hidden, the automaton minimized and exported, and the resulting XML file was parsed to extract all unsafe event sequences up to a length of ten events, which resulted in 83 sequences (see Section 7.5). One example of an unsafe sequences that was synthesized is as follows:

$$\{walk, reach\_parts, retract, walk, press\_button, retract, reach\_in\_housing, collision\}$$

Contrary to the nominal assembly procedure, where the worker should activate the robot *after* reaching into the housing, the worker in this sequence switches two actions and presses the activation button before reaching into the housing, which results in a collision with the robot.

List of Figures

List of Tables

List of Algorithms