



HELMUT SCHMIDT
UNIVERSITÄT

Universität der Bundeswehr Hamburg

On Diagnosing Cyber-Physical Systems

Der Fakultät für Maschinenbau und Bauingenieurwesen
der Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegte

DISSERTATION

von ALEXANDER DIEDRICH

aus Steinheim Westf.

Hamburg 2023

Gutachter

Prof. Dr. rer. nat. Oliver Niggemann
Prof. Dr.-Ing. habil. Jürgen Beyerer

Tag der mündlichen Prüfung

27.06.2023

Abstract

Cyber-physical systems are a class of technical systems that integrate mechanical components with intelligent, adaptable control devices and software. Nowadays, this integration enables high-performance, modular, and parameterized systems with high complexity, but low operating cost. Typical examples of cyber-physical systems are production machinery, cars, aeroplanes, and smart home appliances. In this thesis, the focus is on diagnosing faults within cyber-physical systems used in industrial production contexts. Faults occurring during production quickly lead to degrading product quality or production stops, which can be costly and may endanger human lives. Existing approaches to automated fault diagnosis are mostly defined on narrow use-cases or require a significant amount of expert knowledge. In this thesis, three different algorithms to automatically identify faults in cyber-physical systems are presented to mitigate these drawbacks. Therefore, this thesis makes four main contributions: (i) It introduces a novel diagnosis algorithm HySD to find faults in cyber-physical systems. (ii) It presents a new uninformed algorithm DDRC to learn diagnosis models from process data, using correlations in time-series data. (iii) It presents the new algorithm DDGD, which learns diagnosis models from time-series data supervised, using Granger Causality. (iv) It provides a novel theory to describe fault propagation in cyber-physical systems. More precise, the algorithm HySD uses satisfiability modulo linear arithmetic to combine process data with traditional symbolic consistency-based diagnosis algorithms. However, the algorithm heavily relies on models formulated by experts. Therefore, the algorithms DDRC and DDGD are introduced to learn diagnosis models from process data automatically. All algorithms build on the foundation of the theory of fault propagation. The algorithms were evaluated on internationally accepted benchmarks of tank systems, the well-known Tennessee Eastman Process, and two industrial use-cases. Throughout all empirical results, the algorithms exhibit good performance in learning suitable models and in diagnosing faults in synthetic and real fault scenarios.

Kurzfassung

Die automatische Diagnose von Fehlern ist eine wichtige Fähigkeit moderner cyber-physischer Systeme. Cyber-physische Systeme wie Produktionsanlagen, Fahrzeuge, Smart Home Geräte und viele andere zeichnen sich durch eine hohe Komplexität, Modularität und Parametrisierbarkeit aus. Diese Komplexität lässt jedoch den Aufwand, Fehler manuell zu suchen und zu beheben, ständig steigen. Daher werden intelligente Algorithmen benötigt, welche ein cyber-physisches System so analysieren können, dass Fehler automatisch diagnostiziert werden können. Ziel dieser Arbeit ist es, solch einen Algorithmus für cyber-physische Systeme im Bereich der industriellen Produktion zu entwickeln. Dabei soll der Algorithmus traditionelle Diagnosealgorithmen aus dem Bereich der Schaltkreisdiagnose nutzen und diese an heterogene Prozessdaten anbinden. Weiterhin soll der Algorithmus in der Lage sein, Modelle von Systemen selbstständig zu lernen und zu verwenden. Um dieses Ziel zu erreichen, werden in dieser Arbeit die folgenden Beiträge vorgestellt: Erstens, ein neuartiger Diagnosealgorithmus für cyber-physische Systeme (HySD). Der Algorithmus nutzt hierbei Satisfiability Modulo Linear Arithmetic, um Prozessdaten mit bekannten Algorithmen der konsistenzbasierten Diagnose zu kombinieren. Zweitens, ein von Expertenwissen weitgehend unabhängiger Algorithmus (DDRC), welcher auf Korrelationserkennung beruht, um Diagnosemodelle aus Daten zu lernen. Drittens, ein auf dem Ansatz der Granger Causality beruhender Algorithmus (DDGD), um Diagnosemodelle mithilfe von Fehlerannotationen zu lernen. Viertens, eine neuartige Theorie, um Fehlerfortpflanzung in cyber-physischen Systemen zu behandeln und zu beschreiben. Insgesamt wird mit diesen Algorithmen eine Methode beschrieben, mit der automatisch Diagnosemodelle aus Prozessdaten gelernt und dem Algorithmus HySD zur Verfügung gestellt werden können. HySD liest aktuelle Prozessdaten ein, vergleicht sie mit der Modellvorhersage und ermittelt bei Diskrepanzen zwischen Modell und Prozessdaten eine Menge von möglichen Fehlerursachen. Alle entwickelten Algorithmen wurden mit Hilfe von frei verfügbaren Datensätzen von Tanksystemen, dem bekannten Tennessee Eastman Process, sowie zwei industriellen Anwendungsfällen getestet. Dabei zeigt sich sowohl in der Fehlerdiagnose, als auch im Modelllernen eine gute Performance.

Acknowledgements / Danksagung

This thesis would not exist without the valuable support of many people and organisations. First and foremost I would like to thank Prof. Dr. rer. nat. Oliver Niggemann who has not only suggested the first ideas behind my thesis topic, but has guided my research through many years now. Further, I would like to thank my second examiner for this thesis Prof. Dr.-Ing. habil. Jürgen Beyerer for his input during regular status presentations.

My introduction into the topic of consistency-based diagnosis was made possible mainly through my research experience at Xerox PARC, CA, USA under the supervision of Alexander Feldman, PhD. His introduction into this interesting research field, and especially the DX community, resulted in many interesting ideas and papers. Additionally, he is quite good in recommending great vacations! The overall research stay at PARC would not have been possible without the encouragement and support of Prof. Dr. rer. nat. Oliver Niggemann and Prof. Dr.-Ing. Jürgen Jasperneite, as well as the significant financial support of the Fraunhofer society, Technische Hochschule Ostwestfalen-Lippe, Studienfonds OWL, and Hans und Clara Lenze Stiftung. Prof. Jasperneite is also responsible that I entered the academic career path in the first place.

Another form of significant financial support has come from the Phoenix Contact Stiftung in the course of the main part of this thesis. Their valuable support has enabled me to complete my dissertation faster and with (hopefully) better quality. In addition, it has enabled me to even perform some research outside of the domain of my thesis.

Außerdem möchte ich auch meiner Familie danken. Meiner Mutter, die mir lange Zeit während meiner Ausbildung und meines Studiums finanziell und zeitlich den Rücken freigehalten hat. Und vor allem auch meinem Vater, der mein Interesse an technischen Dingen und vor allem am Basteln mit Elektronik geweckt hat. Außerdem möchte ich meinen Kollegen und Freunden danken: Jens Eickmeyer für seine konstruktiven Gedanken, Netzwerkfähigkeiten und viel Spaß. Kaja Balzereit für die coole Zusammenarbeit, Spaß und fachliche Diskussionen im Rahmen unserer beiden Dissertationen. Sowie den anderen Mitarbeitern vom Fraunhofer, speziell Marta Fullen, Franziska Zelba, Katharina Giese und Daniel Schneider für viele fachliche Diskussionen und die coole, wenn auch manchmal stressige Zeit. Außerdem danke ich Andreas Bunte für fachliche Diskussionen und coole Konferenzbesuche an weit entfernten Orten.

Schließlich danke ich meiner Freundin Silke Merkelbach für Ihren signifikanten Aufwand für das Review dieser Arbeit und meinen Kollegen Niels Grüttemeier und Lukas Moddemann für Reviews und fachliche Diskussionen.

Contents

Abstract	iii
Kurzfassung	v
Acknowledgements / Danksagung	vii
Glossary	xiii
I. Motivation	1
1. Introduction	3
1.1. Problem Description	6
1.2. Research Questions	9
1.3. Scientific Contribution	12
1.4. Running Example	15
1.5. Research Articles	17
1.6. Outline	19
II. Background	21
1. Foundations	23
1.1. Propositional Logic	23
1.2. Predicate Logic	25
1.3. Satisfiability Modulo Theory	27
1.4. Reasoning	30
1.5. Causality	32
1.6. Consistency-based diagnosis	37
1.7. Consistency-based diagnosis algorithms	41
1.8. Interfacing consistency-based diagnosis algorithms	43
1.9. Creating System Descriptions from Data	44
2. State of the Art	47
2.1. Consistency-based Diagnosis	47
2.2. Fault Detection and Isolation	49
2.3. Causality And Dependencies	51
2.4. Satisfiability Modulo Theory	53
2.5. Industrial Approaches	54
2.6. Major Related Work	56
2.7. Summary and Research Gaps	56

III. Solution	59
1. Solution Approach	61
2. A New Algorithm for Consistency-based Diagnosis for Cyber-Physical Systems	65
2.1. Algorithm HySD	66
2.2. Using HySD	71
2.3. Example: HySD	73
3. A New Algorithm to Learn System Descriptions from Process Signals	77
3.1. Algorithms DDRC and DDA-IM	78
3.2. Example: DDRC and DDA-IM	80
4. A New Algorithm to Learn System Descriptions from Residuals and Component Health States	83
4.1. Algorithm DDGD	84
4.2. Example: DDGD	86
5. A Novel Theory of Fault Propagation	89
5.1. Component and Connection Models in Practice	90
5.2. Definitions	90
5.3. Structure Patterns using Signal Values	92
5.4. Discussion of signal-based fault propagation	93
5.5. Structure Patterns using Residual Values	95
5.6. Discussion of residual-based fault propagation	96
5.7. Example: Causal Models	97
IV. Evaluation	99
1. Overview	101
2. Methodology	103
3. Empirical Results	107
3.1. Description of data-sets	107
3.2. Algorithm HySD	109
3.3. Checking model validity - Algorithm CheckDiag	114
4. Theoretical Analysis	121
4.1. Computational Complexity	121
4.2. Running-time Evaluation	122
V. Conclusion and Future Work	125
1. Discussion	127
2. Conclusion	129
3. Outlook	131

VI. Appendix	133
1. System Descriptions of the investigated Systems	135
2. Code Availability	147
Bibliography	149

Glossary

Symbol	Description	Definition
m	A measured physical value	Def.: 33
\mathbf{x}	State vector	Eq.: 1.1
\mathbf{y}	Output vector	Eq.: 1.1
\mathbf{u}	Input vector	Eq.: 1.1
α	A single observation	Def.: 34
OBS	The set of observations	Def.: 34
$d(m)$	Discretisation function for a measured value	Def.: 34
SD	The system description	Def.: 37
Φ	A set of logical expressions, such as SD	Chap.: 1, Part II
COMPS	The set of component names	Def.: 37
$(\Phi, COMPS, OBS)$	A diagnostic system	Def.: 38
ω	A set of diagnosed faulty components	Def.: 39
ω'	The minimum cardinality diagnosis	Def.: 40
\mathcal{V}	Set of component variables	Def.: 21
\mathcal{V}_i	Set of input variables for a component	Def.: 21
\mathcal{V}_o	Set of output variables for a component	Def.: 21
\mathcal{V}_s	Set of state variables for a component	Def.: 21
\mathcal{F}	Set of all component model functions	Def.: 21
name	The symbolic name for the component	Def.: 21
CON	The connection model	Def.: 17
$p_{e,i}$	An effort connective in a connection model	Def.: 23
$p_{f,i}$	A flow connective in a connection model	Def.: 23
$h(\mathbf{x}, \mathbf{u}, \mathbf{y}, \tau)$	A health function calculating a residual	Eq.: 1.1
τ	A threshold	Eq.: 1.1
$\hat{\alpha}$	A model prediction	Chap.: 1, Part II
$n(\mathbf{x}, \mathbf{u})$	The output model of a component	Def.: 24
Ξ	Set of parameters for error terms	Def.: 25
$f_m(\Xi_m)$	Multiplicative fault error term	Def.: 25
$f_a(\Xi_a)$	Additive fault error term	Def.: 25
G_S, G_J, G_R	Structure patterns	Sec.: 5.2
G_M, G_D, G_C	Structure patterns	Sec.: 5.2
r	A residual value	Chap.: 4, Part III
φ	A generic set of logical expressions	Chap.: 1, Part II
CM	The set of component model functions \mathcal{F}	Chap.: 1, Part II
M	A set of process data values	Chap.: 3, Part III
T	A time series	Chap.: 3, Part III
\tilde{T}	The signal names of T	Chap.: 4, Part III
\tilde{S}	The signal names of S	Chap.: 4, Part III
\tilde{M}	The signal names of M	Chap.: 4, Part III
S	A set of quality signals	Chap.: 3, Part III
F	A set of health states $F \subset COMPS$	Chap.: 4, Part III
ρ	The Spearman Correlation	Eq. 1.3

List of Figures

1.	Contributions of this thesis	13
2.	The running example of a one-tank system	15
3.	An example of a four-tank system	16
4.	The causal graph for the running example	33
5.	Structure of a single component	35
6.	Structure of an exemplary connection model	36
7.	Interfaces to diagnosis algorithms	44
8.	Architecture of the major algorithms in this thesis	62
9.	The solution approach	63
10.	Different fault propagation structure patterns	91
11.	Generation of a residual	95
12.	Expert defined system model of the one-tank running example	104
13.	Running time in milliseconds of algorithm HySD compared to the problem size	123
14.	Architecture of the evaluation system.	147

List of Tables

1.	Major articles with highest impact	17
2.	Main articles as first author	18
3.	Co-authored articles	19
4.	Truth table for a negation (\neg)	24
5.	Truth table for a conjunction (\wedge)	24
6.	Truth table for a disjunction (\vee)	24
7.	Truth table for an implication (\rightarrow)	24
8.	Truth table for an equivalence (\leftrightarrow)	25
9.	Signal division into sets S and M	78
10.	Correlation matrix ρ of algorithm DDRC	80
11.	Expert defined components related to Boolean residuals	84
12.	Expert defined components related to real-valued residuals	86
13.	Expert defined components related to Boolean residuals (transformed)	86
14.	Comparison between the calculation of absolute errors and residuals	97
15.	Experimental results using HySDand DDGD	111
16.	The results for the injection molding machine using DDRC.	113
17.	The results for system S_3 using DDRC.	114
18.	The results for system S_3 using DDRC.	114
19.	Example of a perfectly diagnosable logical system description	117
20.	Example of a logical system description with hidden components $\{a, b\}$	118
21.	Model validity for all diagnosed systems	119
22.	Model validity for systems of the ISCAS-85 benchmark	119
23.	Comparison between rules generated through signal-based and component-based Granger Causality	119
24.	Comparison of the computational complexity and running time of the algorithms developed in this thesis	121

Part I.

Motivation

1. Introduction

Within the life cycle of any technical system, some of its constituent parts may at some point break down and need to be replaced for the system to keep its intended functionality. Automatically finding those broken components is the task of the research field of fault diagnosis. This thesis presents a novel and practical method to diagnose faults in cyber-physical systems. Cyber-physical systems [1] are a subset of technical systems and include intelligent control software in addition to the traditional electronic and mechanical components such as levers, conveyors, valves etc. of older technical systems. While the presented approach is applicable to most cyber-physical systems, particular focus in this thesis lies on cyber-physical production systems, which are necessary for the manufacturing of most modern products. Companies tend to prefer cyber-physical production systems over purely mechanical or mechatronic systems due to their higher flexibility, easier adjustment, and better networking. For example, cyber-physical production systems can more easily adapt to changes in demand, allow product customisation, enable modularity, and extend overall reporting and control functionality [2]. But all this flexibility comes with drawbacks: the technical complexity of these systems increases significantly, not only in hardware, but also in software. The high complexity, particularly in the number of system components and their associated interdependencies, are hard for humans to grasp. The more components exist, the higher the probability that some component may fail. The increase in complexity and the higher fault probability thus lead to the requirement of using automated diagnosis to find faults faster and keep downtimes at a minimum [3, 4, 5, 6].

The necessity to develop efficient fault diagnosis methods for cyber-physical systems and their integration into existing processes has been identified within the industry [7], governmental [8] and non-governmental bodies [9, 10, 11], and in the academic fields of control theory [12] and artificial intelligence [13]. But so far, in the research field of artificial intelligence, industrial solutions are either proprietary implementations adapted to a single use-case [14], or are knowledge-based systems requiring a significant amount of human input [15, 16]. In the research field of control theory, fault diagnosis methods are developed under the label of *fault detection and isolation* [12]. But those methods require an accurate understanding of the underlying physical effects and often require first-principles models in the form of (differential) equations. For many cyber-physical systems, obtaining such equations and models is very costly. However, one class of algorithms can be identified, which lend themselves to general applicability for fault diagnosis: consistency-based diagnosis algorithms. These have been researched mainly for Boolean circuit diagnosis [17], but since they rely on a highly flexible knowledge base specified in logic, as well as highly adaptable reasoning, it is a natural intuition to adapt them to other domains. Such an adaptation is the aim of this thesis.

There is a long history of applying consistency-based diagnosis algorithms to practical problems. The research fields concerned with fault diagnosis go back at least until the 1970s [18]. Since then, progress has always been gradual and limited to narrow fields. For example, quite some progress has been made in developing better diagnosis algorithms.

The first formal algorithms were developed by De Kleer [19] and Reiter [17]. But many other algorithms have been developed by, for example, Stern et al. [20], Feldman et al. [21, 22], Zhao et al. [23], and many others. At the same time, modelling systems through equations has progressed from early approaches as those mentioned by Isermann et al. [24], over graph-based approaches by Biswas et al. [25], to elaborate systems of equations for diagnosis [12]. All of these works present robust and usable methods to perform diagnosis for many systems. What they all rely on, however, is a common understanding of deterministic fault effects, which is often termed fault propagation. Further, many diagnosis algorithms are only able to process binary signals. But in practice this is often not the case, as data in cyber-physical systems is more heterogeneous and includes discrete and continuous values. To illustrate, assume some Boolean circuit: when one component fails, its effect will deterministically propagate from the source of the fault to the end (the output) of the circuit. The different ways in which the propagation will occur can be counted, although this can be computationally expensive. In complex cyber-physical systems this determinism does not exist. In a biological reactor, for example, if some input valve fails and releases a wrong dosage of some input, it is often unknown how the output may behave.

Additionally, cyber-physical systems are often not static. Depending on the produced product, their setup may change or some parameters are changed to optimise production. All of this requires that a diagnosis algorithm needs to reason about how faults propagate and how system components behave. It must also be able to respond to those changes by allowing changes to its knowledge base. In the past, several approaches such as those by de Kleer [26], Forbus [27], or Kortoglu [28] have attempted to capture fault propagation. Overall, for any diagnosis algorithm that is used in cyber-physical systems it is necessary to have an accurate model of how effects propagate between system components. But again, obtaining these models is hard and so far has often been done by experts for narrowly defined use-cases. The research behind fault propagation can be traced back to Schopenhauer's principle of sufficient reason [29], McDermott's non-monotonic reasoning [30, 31], and Reiter's and De Kleer's theories of diagnosing faults [17, 19]. Schopenhauer's principle of sufficient reason describes the philosophical aspects of cause and effect relations in the real world. Most importantly, he states the necessity that causes and effects are always due to changes over time and space. McDermott's theory of non-monotonic reasoning has established a logical framework which allows the retraction of assumptions. Without such retractions modern consistency-based fault diagnosis would be impossible, as inconsistent knowledge bases would be prohibited. Finally, Reiter's and De Kleer's theories of fault diagnosis, as well as De Kleer's and Forbus's ideas on fault propagation [32, 33], build on the philosophical aspects of Schopenhauer, McDermott, and many others to apply non-monotonic reasoning to practical problems in fault diagnosis.

This thesis presents an entirely novel and integrated approach for automatically diagnosing faults in cyber-physical systems. More precisely, it is an approach to develop a novel and practical method to apply traditional consistency-based diagnosis algorithms to heterogeneous, modular cyber-physical systems. The approach is able to learn parts of its model from time-series data, if available, and from experts otherwise, allows the augmentation of the model through experts, provides functionality to diagnose systems with discrete and continuous values, and allows the usage of many dedicated diagnosis algorithms such as GDE (General Diagnosis Engine) [19], SAFARI (Stochastic Fault Diagnosis Algorithm) [22], Reiter's Hitting Set-tree (HS-Tree) [17, 34], and other compatible algorithms [35]. At the same time, it provides an integrated knowledge base, formulated in satisfiability modulo

linear arithmetic (SMT- \mathcal{LRA} , a special kind of logic that can reason about real numbers) which is explainable [36] and compatible to many established diagnosis algorithms by translation into propositional logic. This is different to many previous diagnosis approaches in industrial contexts such as automata [37] and other discrete event systems [38] which can only process discrete signals. Grastien has shown that using SMT- \mathcal{LRA} can provide advantages to deal with continuous and discrete signals [39]. Thus far, however, SMT- \mathcal{LRA} was not applied to broadly defined real-world systems.

In addition to performing diagnosis using SMT- \mathcal{LRA} , this thesis also discusses the construction of suitable diagnosis models that are practical to learn and can accurately represent a cyber-physical system. This is necessary such that experts do not have to specify each model of some cyber-physical system manually using logical SMT- \mathcal{LRA} expressions. For this, a model of the system is divided into a connection model and a behaviour model [40]. While both of these are explained later in detail, so far it is important to note that the connection model describes connections between parts of the system and the behaviour model describes the behaviour of those different parts. In the thesis, the connection model will be specified as a graph and through logical expressions, while the component models will be specified as functions. On top of the behaviour model, residual generation equations are specified through observer-patterns, thus taking inspiration from the fields of artificial intelligence and control theory. Previous approaches [41, 42] have used residual generation with a unique formulation to perform diagnosis, but do not adapt well between different systems.

All of this leads to a novel and generalized approach to automated fault diagnosis in cyber-physical systems. System behaviour is predicted through behaviour and connection models, which in turn are formulated in the algorithm agnostic SMT- \mathcal{LRA} logic. Then common diagnosis algorithms are applied to determine those components which have caused a fault.

1.1. Problem Description

This section shall analyse and discuss the underlying problems in this thesis from several aspects. First, we will provide a theoretical definition of the diagnosis problem itself and then describe how the diagnosis problem can be generalised. Second, we will look into several problems related to using a diagnosis method in practice.

The diagnosis problem is defined as follows: Given a set of observations, as well as a behaviour and connection model of a system, find those components that might have caused a fault. In practice, the observations are time-series data and the behaviour and connection models are electronically readable engineering documents [18, 43], some suitable representation of their content, or similar data. For Boolean circuits this problem has been initially solved in the 1980s [17, 19], and has been improved since. The main goal in traditional circuit diagnosis was to find a function which takes time-series data (or a single datum) as its input and, provided the data contains faults, outputs a set of possibly faulty components. Formally this can be denoted as

$$\omega = \text{diag}(\mathbf{X}, \gamma) \quad (1.1)$$

where $\mathbf{X} \in \mathbb{B}^{n \times m}$ is some time-series of size $n, m \in \mathbb{N}$, ω is a set of faulty components, γ is some model, and $\text{diag}() : \mathbb{B}^{n \times m}, \gamma \rightarrow \omega$ is some suitable diagnosis function [35]. Please note that for notation we write \mathbb{R}^n to denote the size of the vector of real numbers. In the case of sets, we also use the notation \mathbb{R}^V , instead of $\mathbb{R}^{|V|}$, for some hypothetical set of variables V . The model γ is usually implemented through propositional logic expressions (derived from logic gate truth tables). Finding and improving function $\text{diag}()$ was for many years one of the main tasks in the research field of consistency-based diagnosis [44].

But for cyber-physical systems the diagnosis function and the models need to be extended. Since these systems contain not only Boolean values, but usually real values as well, the problem can be formulated as finding a function $\text{diag_cont}()$ to satisfy

$$\omega = \text{diag_cont}(\mathbf{X}, \gamma) \quad (1.2)$$

where $\mathbf{X} \in \mathbb{R}^{n \times m}$ is some real-valued time-series of size $n, m \in \mathbb{N}$ and $\text{diag_cont}() : \mathbb{R}^{n \times m}, \gamma \rightarrow \omega$ is some diagnosis function. Determining such a function and finding suitable model formulations are the main objectives in this thesis.

In many industrial use-cases, however, γ cannot be provided a-priori. Another problem to be solved in this thesis is therefore, how to avoid the explicit specification of some model γ and instead find a function $\text{diag_cps}()$ which satisfies

$$\omega = \text{diag_cps}(\mathbf{X}, \mathcal{L}) \quad (1.3)$$

where \mathcal{L}^n is a set of labels to annotate faulty episodes within \mathbf{X} , and $\text{diag_cps}() : \mathbb{R}^{n \times m}, \mathcal{L} \rightarrow \omega$ is some diagnosis function.

Usually, a function such as $\text{diag_cps}()$ can be implemented through calls to a function such as $\text{diag_cont}()$, and $\text{diag_cont}()$ can be implemented through calls to some function $\text{diag}()$. Now, the aim for this thesis is first to find a method which primarily adapts $\text{diag_cont}()$ to $\text{diag}()$, as the Boolean diagnosis problem of $\text{diag}()$ has already been solved in the past. For most known algorithms $\text{diag}()$ uses for its model some propositional logic formulations. So some function $\text{diag_cont}()$ needs to adapt the real valued sensor data for those Boolean

diagnosis algorithms. In particular, it needs to provide a mapping $\mathbb{R}^{n \times m} \rightarrow \mathbb{B}^{n \times m}$. At the same time it must be able to use γ in such a way, so that it models how components usually behave (their normal working behaviour) and how they are connected. In Boolean circuit diagnosis this is done through truth tables [45]. For cyber-physical systems this is still an open question, although several promising approaches exist [12, 46, 47]. The main complication of models for cyber-physical systems is their heterogeneity. While Boolean circuits can be modelled through a small number of fundamental truth tables (logic gates), the diverse number of possible components in cyber-physical systems such as tanks, pumps, levers, electrical components, proprietary devices, etc. makes it impossible to find an enumerable minimum set of common behaviour models. Instead, some method needs to be found which enables practitioners to provide a model capturing a system's main characteristics and connections. To summarise, a diagnosis algorithm for cyber-physical systems needs to rely on existing diagnosis algorithms defined on Boolean input data, while processing real-valued input and adapting models accordingly.

Further, in order to avoid the explicit specification of models, a method such as *diag_cps()* is necessary which can learn (at least) an approximation of γ from a set of known faulty periods. This enables practitioners to avoid the costly specification of elaborate models. One reason to learn models automatically is that modern cyber-physical systems may change their architecture over their lifetime. Approaches to adapt a diagnosis algorithm to modular cyber-physical production systems exist [48, 49]. But all of them require the existence of a correct diagnosis model for each possible change in constituent modules. In practice, this requirement leads to a combinatorial explosion of manually created models. Additionally, in many controlled domains, where fault diagnosis algorithms are currently successfully applied, the system behaviour as well as its faulty behaviour are known. This means in those cases an expert is able to create a model for normal operation as well as describe all possible faults and the effects of these faults [50]. But those controlled domains are rarely encountered in real production environments. Some types of faults occur only once and were unforeseen by the system designers. This is particularly surprising in the case of critical systems which are subjected to a failure mode effects analysis (FMEA) [51] or other risk-analysis schemes. The goal of these analyses is to systematically analyze possible fault modes such that the probability of their occurrence can be minimized. However, many faults such as those due to human error, deviations in input material, or even structural deficiencies are sometimes not accounted for (otherwise there would be no major incidents at complex and dangerous production facilities such as refineries or spacecraft [52]).

Overall, consistency-based diagnosis algorithms and their models solve a philosophically hard question: They reason against the flow of causality. Given a fault, they determine what might have caused the fault according to some model. In causal research this type of reasoning can lead to intractable problems. For example, Taleb [53] has given the example of an ice-cube. If an ice-cube melts and creates a puddle one cannot determine whether the water was originally an ice cube or if someone has spilled the water from a cup. For a diagnostic system to understand what has happened it is therefore necessary to have an accurate model. The model can be supplied with current observations from a cyber-physical system and the diagnostic algorithm can reason what might have happened, given the current observations. Pearl [54], Halpern [55], Bochman [56], and others have made advances to model causal behaviour in the real world during the last decade. The developed formalisms, such as Pearl's structural equation models [54], were not intended to be used for diagnosing cyber-physical systems, but instead were developed to reason about causality. It is thus necessary to adapt those formalisms in order to make them applicable

to industrial processes. The most pressing problem is that Pearl and others assume a real-world ground truth of causality exists and that it can be explicitly formulated. But complex industrial processes can often not be formulated in such a way that a ground-truth exists. Instead, it is necessary to learn dependencies through system observations or through some hybrid approach using system observations and expert knowledge. This leads to trade-offs: Pearl and other researchers of causality correctly state that causality cannot be learned through observations alone. But if a diagnosis algorithm requires a (causal) model which can only be obtained through observations, a trade-off must be found that risks to get some causality / dependencies wrong, but one that is also practical to learn and apply. Thus, in what follows, we will refer to causality in a strict sense: There exists a ground truth of what cause leads to which effect. Whenever this ground-truth is unknown, we will refer only to dependencies between components.

Latest research still shows significant gaps between theoretical research and a generally applicable diagnosis method for cyber-physical systems. Recently, new diagnosis algorithms have been developed, but these are evaluated on binary circuits [34, 23, 13]. Others have introduced a ternary logic to diagnose hybrid systems, but do not capture all of the dependencies with their approach [57]. Still other approaches are new algorithms for structural analysis of systems [12]. Having discussed the limitations of current approaches to fault diagnosis in cyber-physical systems we will now go on and set them into a research context.

1.2. Research Questions

Looking at the challenges identified in the last section, several research directions for this thesis can be derived. It was identified that no general approach exists for diagnosing faults in cyber-physical systems. Additionally, it was shown how different approaches from the research fields of artificial intelligence [19], machine learning [58], control theory [12], and engineering [59, 60] all offer approaches to diagnose specific systems. Approaches to fuse several of these research fields exist, but have so far not been applied to cyber-physical systems or other complex setups [61, 62]. The industry describes a system's ability to automatically diagnose itself as self-repair [7]. But to realise the full potential of self-repair technologies, it is necessary to find a generally applicable diagnosis approach. Such an approach should be adaptable between different systems and needs to be adaptable to changes to the system's architecture. However, the system also needs to be equipped with technologies which facilitate diagnosis: i) It must be known which components are located within the system and how those components are interconnected. ii) The instrumentation of the systems must be such that the *important* behaviour can be observed.

In this thesis three research questions will be answered. They will describe how traditional consistency-based diagnosis algorithms will be adapted for cyber-physical systems, how faults propagate in systems, and how diagnosis models can be specified and learned automatically.

Since no general approach exists yet to diagnose a broad class of cyber-physical systems, the first research question guides the investigation towards a suitable method.

RQ 1: How can consistency-based diagnosis be used in cyber-physical systems?

RQ 1.1: How can a consistency-based diagnosis algorithm be integrated into cyber-physical systems?

RQ 1.2: How can observations be represented by continuous values and how can these values be integrated into symbolic logic?

A consistency-based diagnosis method for cyber-physical systems needs to be able to process binary and continuous sensor values, must be able to integrate external (expert) knowledge, and must be able to use derivations of those models, which are usually used during construction and while operating cyber-physical systems. In binary circuit diagnosis, many algorithms exist for this task. The goal in this thesis is to adapt those algorithms to work with cyber-physical systems. A diagnosis algorithm for cyber-physical systems must be able to track fault propagation and still diagnose the root-cause. So far, only approaches for narrowly defined domains [14, 63] exist. But especially cyber-physical systems in production environments are more modular and less well-defined than many other systems. Further, fault modes are usually represented in literature using strong-fault models [20]. These models require information about all faulty behaviour. But enumerating all possible fault modes and creating strong-fault models cannot be done practically for many cyber-physical systems. Instead, it is necessary to rely on weak-fault models [20], which only model the normal behaviour of the system. Many of these questions shall be answered in this thesis by introducing the right model formalism. Most current model representations use some kind of logic. Binary circuit diagnosis uses propositional logic. Grastien suggested using a special predicate logic (Satisfiability Modulo Theory) [64]. Pill et al. [65] have suggested using Linear Temporal Logic [66] to formulate timing behaviour. But temporal logic, which is useful for discrete systems and diagnosing software, is

not expressive enough for cyber-physical systems. Apart from the timing behaviour, for cyber-physical system diagnosis a method needs to be found which translates the signals obtained from the system, the behaviour model, and the connection information into one diagnosis model. This model must be transformable into propositional logic in order to interface it with consistency-based diagnosis algorithms. One method for translating sensor values into logic is by using residual values (referred to as residuals). A residual value is a real number that is usually equal to zero, when the underlying sensor shows normal behaviour and is different from zero (given some margin), if the sensors show some abnormal value. Most research assumes that the equations to generate residual values are given. But for diagnosing cyber-physical systems this assumption cannot hold. Therefore, a general application of a method must be found, which can generate residual values either automatically or with a minimum of expert knowledge.

At the center of each diagnosis problem whether in industry, binary circuit diagnosis, or in control theory lies an accurate model. Model predictions are compared to real observations, and discrepancies between prediction and reality are used to perform diagnosis. In other words, a model usually describes which component (if faulty) may cause which observation to be non-normal. This boils down to the question: Which cause leads to which symptom? To answer this question a model of the system's dependencies is necessary which describes how components are connected and how each single component behaves. Using a weak-fault model, every deviation from this normal behaviour is treated as an anomaly and fault diagnosis is started. Overall, a model-formalism is required which describes the normal behaviour of each component and describes how the output of each component relates to every other component within the system. Then, if a fault occurs, an algorithm can determine the cause of the fault by analysing the fault propagation between components. This leads to the second research question.

RQ 2: How can fault propagation in cyber-physical systems be described?

RQ 2.1: How can dependencies be defined in cyber-physical systems?

RQ 2.2: How can a model be adapted to changing and modular systems?

Value propagation is at the heart of the research field of causality as defined by Pearl [54]. But what is missing is a definition of causality in cyber-physical systems that is applicable to perform diagnosis. In cyber-physical systems it is often impossible to determine true, genuine causality in the same sense as Pearl has done due to missing information. Instead, it makes more sense to refer to dependencies between components. But as with the integration of consistency-based diagnosis approaches (RQ 1), dependencies still need to be set into the context of cyber-physical production systems. While many works exist that analyse causal dependencies in everyday life [54, 55], others have attempted to generally define causality for data-driven approaches [67]. Further, each possible approach to define causality in these systems must also adapt to changing system architectures. Thus far, such an approach has not been developed.

The information about how components within a cyber-physical system behave and how they are connected needs to be represented such that a consistency-based diagnosis algorithm can be applied [22, 34]. In binary circuit diagnosis this information is contained within a propositional logic knowledge base [62]. In control theory this information is contained within the formulation of differential equations [41]. To adapt consistency-based diagnosis algorithms to cyber-physical systems, a model formalism needs to be created

which allows the integration of heterogeneous data from production systems, as well as the connection and behaviour models. The third research question can thus be formulated.

RQ 3: What is a suitable model to perform consistency-based diagnosis of cyber-physical systems?

RQ 3.1: How is it possible to learn a diagnosis model from data?

RQ 3.2: How diagnosable are learned models?

So far, no approaches exist yet to learn diagnosis models from data. But in order to cope with modular cyber-physical systems it is a necessity to learn at least parts of the model from available process data, as manually specifying models is far too expensive. However, learning these models leads to the challenge of evaluating whether the learned model is correct. Since models can become quite large, it is infeasible and expensive to let an expert check those models. Thus, an automated method is required which can (at least heuristically) determine whether a data-driven model is suitable for a diagnosis task.

Overall, answers to these research questions shall determine how to design a consistency-based diagnosis methodology using traditional algorithms combined with automatically learned models based on a common logical representation.

1.3. Scientific Contribution

To answer the research questions above, this thesis makes four main contributions. Together, they describe a novel and practical approach to perform diagnosis for cyber-physical systems. In particular, they answer how faults propagate in cyber-physical systems through a new theoretical framework of fault propagation and, using the framework, provide new methods to diagnose faults. In addition, novel algorithms are introduced which show how diagnosis models are learned from data and how those diagnosis models can be evaluated regarding their fitness for diagnosis tasks.

The central goal of the thesis is to leverage previous works in the research field of consistency-based diagnosis to provide a novel method for fault diagnosis by integrating a model formulated in SMT- \mathcal{LRA} logic (RQ 1). As a result, the diagnosis algorithm *Hybrid System Diagnosis* (HySD) is presented which facilitates automated fault diagnosis of cyber-physical systems. The algorithm is able to convert an expert-defined system model (also called system description) into SMT- \mathcal{LRA} expressions, integrate and evaluate real-valued residuals, and perform diagnosis using Reiter's HS-tree [17]. Additionally, two light-weight algorithms HySD_simple and DDA-IM are introduced, which each contain a subset of the functionality of HySD. These two latter algorithms are used when a system description already exists and does not need to be converted into SMT- \mathcal{LRA} . In particular, algorithm DDA-IM is used to assign observations from a system to logical expressions, and HySD_simple is used for diagnosis once a suitable system model already exists. Therefore, the first contribution can be stated as:

Contribution 1: This thesis introduces a novel algorithm HySD and theoretical extensions for consistency-based diagnosis using SMT- \mathcal{LRA} logic. This allows the adaptation of traditional, well-known diagnosis algorithms to heterogeneous cyber-physical systems.

Using SMT- \mathcal{LRA} requires the evaluation of sensor signals against some model (RQ 1). As part of this thesis it is evaluated how those signal values can best be used. Many approaches, especially in control theory, use the method of calculating residual values [42, 68] through applying different kinds of observer patterns. Taking the idea of using residual values and observer patterns as a baseline, the thesis provides a more general formalisation to track fault propagation. For this, a theoretical foundation describing why using residual values is superior to using sensor values is presented (RQ 2). This is also a theoretical justification for many previous approaches in the literature. To model fault propagation and residuals, the thesis introduces minimal system topologies that are termed structure patterns. These formally describe fault propagation between a minimum number of components. For all of these structure patterns, the thesis compares two types of fault propagation approaches: 1) An approach that measures signal amplitudes and analyses how these amplitudes propagate through the system. This is usually the case in heuristic and phenomenological diagnosis approaches, where signal values or features generated from signals are fed into some machine learning algorithm. 2) An approach where residuals are generated for each signal. Looking at both approaches, it is studied how calculating residuals changes the way fault propagation can be analysed for cyber-physical systems. This is summarised in the second contribution:

Contribution 2: This thesis provides a novel theoretical and light-weight foundation of fault propagation in cyber-physical systems and presents a novel theoretical justification of the benefits of calculating residual values for fault diagnosis.

Based on the theory of fault propagation, this thesis introduces an approach to learn diagnosis models from data unsupervised (RQ 3.1). For this, the novel algorithm *Data-driven Diagnosis Rule Creation* (DDRC) was developed which learns the influences of production signals on a set of quality signals by computing the correlation between signals from production and quality measurements. The output of this algorithm consists of a set of propositional logic expressions which associate multiple production signals to one quality signal each. This set of propositional logic expressions is then used as the diagnosis model for traditional consistency-based diagnosis algorithms. The uninformed and correlation-based approach is therefore summarised in the third contribution:

Contribution 3: This thesis introduces a novel algorithm DDRC to create propositional logic diagnosis models data-driven, based on correlations.

Taking the correlation-based approach as a baseline, a more informed algorithm was developed to generate diagnosis models. The novel algorithm *Data-driven Granger Causality Detection* (DDGD) uses Granger Causality to learn a diagnosis model supervised. The algorithm evaluates the causality between a matrix of known component health states and associates these to observable process signals. The output of the algorithm is a set of propositional logic expressions associating system components to observable signals. This is an improvement over the correlation-based approach, as diagnoses include specific components instead of signal-names. Algorithm DDGD is summarised in the last contribution:

Contribution 4: This thesis introduces a novel algorithm DDGD, which uses Granger Causality to create propositional logic diagnosis models.

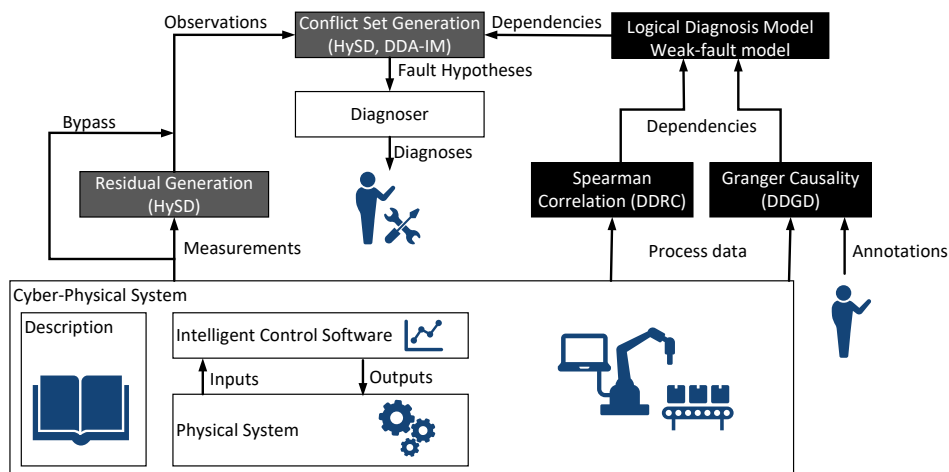


Figure 1.: Contributions of this thesis. Black boxes contain novel approaches, grey boxes describe changes to existing techniques, and white boxes show existing technologies.

Figure 1 summarizes the contributions in this thesis. The underlying assumption is that a cyber-physical system is available and is sufficiently instrumented to observe its functionality. Given process data in the form of time-series measurements, a system description can be learned automatically, either by using correlations through using algorithm DDRC (Contribution 3), or through the use of Granger Causality using algorithm DDGD (Contribution 4) on right-hand side of Figure 1. It must be noted that DDRC requires training data of only normal production and DDGD requires annotated training data of faulty behaviour

in the form of component health states. Once trained, both of these algorithms output sets of dependencies specified in propositional logic expressions. Either, these expressions are used as the model for algorithm HySD, or some expert-defined SMT- \mathcal{LRA} expressions need to be provided. Algorithm HySD itself (left-hand side of Figure 1) uses actual process data and compares it to the provided system description. If the actual process data is inconsistent with the provided system description HySD employs non-monotonic reasoning to identify faulty components that might cause the inconsistency. The output consists of a set of possibly faulty components that need to be repaired or replaced. The functionality of HySD is encapsulated into algorithms DDA-IM for the residual generation and HySD_simple for the actual diagnosis. The concrete theory and implementation behind these algorithms is presented in Part III.

1.4. Running Example

To illustrate the challenges mentioned above a small demonstration use-case shall be introduced as a running example that will be used throughout the thesis. Figure 2 shows the basic components and connections within the system. The use-case consists of two pipes with valves connected to a tank, where water is pumped into pipe 0 and flows out of the tank through pipe 1. It is further assumed that the system is in a steady state such that the water level in the tank stays the same. For instrumentation we assume that the inflow, outflow, and water level within the tank can be observed through sensors and that no noise exists.

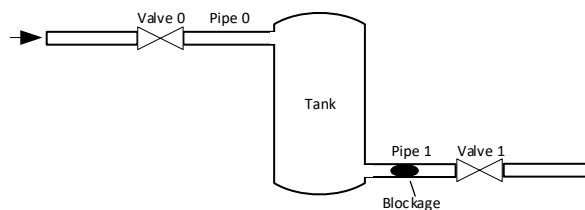


Figure 2.: The running example of a one-tank system

To do fault diagnosis for this simple system the three parts components, connections, and symptoms need to be defined. Components are represented by the names of the objects of the system, namely: *pipe0*, *pipe1*, *tank*, *valve0*, *valve1*. The connections can be formulated as simple strings such as *pipe0* -> *tank*. Analysing the possible observations and symptoms is more complex. A symptom is an indication of a fault and is interpreted as some function mapping from one set of real numbers to another set of real numbers, where each symptom denotes some numerical value indicating whether or not the symptom describes normal behaviour (component health). Describing the three constituents: components, connections, and symptoms is straightforward engineering and many different approaches exist, such as piping and instrumentation diagrams. Formulating these constituents in some logical representation is more difficult. It must be analysed what kind of logic would be suitable. Are binary expressions enough? Are ternary expressions required? Is there a need to analyse and abstract timing behaviour? How can the names of components, the connection graph, and real-valued symptoms be integrated into the same logical representation? Each of these questions can be answered through a different kind of logical model. The difficulty lies in finding a logical representation, which is compatible with established diagnosis algorithms, while ensuring that the representation can still be learned automatically (from process data). For answers to those questions the reader is referred to later chapters.

The one-tank system is used for the majority of illustrations within this thesis. However, for some examples a more complex system is required. For this, a four-tank system is introduced (Figure 3). The system exhibits more complex behaviour as it includes parallel connections, a bypass, and more tanks, and thus leads to more complex system models. The inflow is similar to the one-tank system. But after the first tank two connections lead

1. Introduction

to two intermediate tanks (*tank 1* and *tank 2*), and one connection is a bypass through *valve 3*. All three connections then lead into the fourth tank (*tank 3*), which is then connected to the outflow. The concrete objects within the four-tank system are *valve0*, *valve1*, *valve2*, *valve3*, *valve4*, *valve5*, *valve6*, *tank0*, *tank1*, *tank2*, and *tank3*. The pipes connecting the objects are not specified by name.

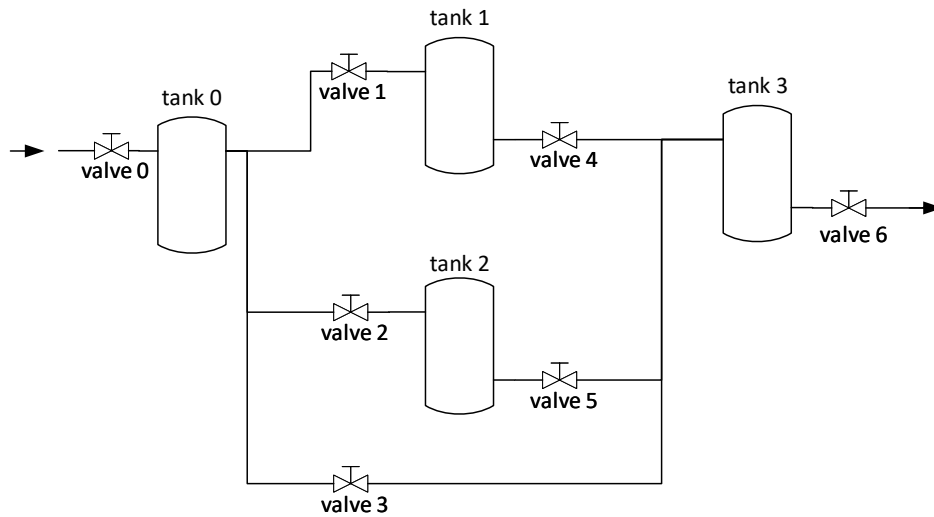


Figure 3.: An example of a four-tank system

1.5. Research Articles

The content of seventeen articles has contributed to this thesis. All mentioned articles have been subject to peer-review. For all articles the author has assumed the CRediT (Contributor Roles Taxonomy) roles: *Data curation, Resources, Software, and Writing - review & editing*. Additional CRediT roles were assumed for the major and first-author articles such as: *Conceptualization, Formal Analysis, Investigation, Validation, Visualization, and Writing - original draft*. This section is structured into three parts: i) Description of those articles with highest impact, ii) Description of the articles as first author, iii) Description of co-authored articles.

Table 1 shows the major articles which were produced for this thesis. The first article was published on the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19) in 2019 (one of two major artificial intelligence conferences) and contains a large part of the investigations of diagnosing cyber-physical systems using first-principles models [69]. In particular, it answers RQ 1 and parts of RQ 3. The publication contributed the formulation of the system description in SMT and showed how an expert-generated cyber-physical system can be diagnosed. The theory behind the causal analysis of cyber-physical systems (RQ 2), the fault propagation, and a comparison between signal-based and residual-based approaches (RQ 1) have been published in Elsevier’s Engineering Applications of Artificial Intelligence journal [70] (Impact Factor 7.802). The article contains most of the definitions and proofs used in this thesis, as well as the algorithm HySD.

Usage	Reference	Ref.
Chp. 5, Pt. II	Diedrich, A., & Niggemann, O. (2022). On Residual-based Diagnosis of Physical Systems. <i>Engineering Applications of Artificial Intelligence</i> , 109, 104636.	[70]
Chp. 2, Pt. III	Diedrich, A., Maier, A., & Niggemann, O. (2019, July). Model-based diagnosis of hybrid systems using satisfiability modulo theory. In <i>Proceedings of the AAAI Conference on Artificial Intelligence</i> (Vol. 33, No. 01, pp. 1452-1459).	[69]

Table 1.: Major articles with highest impact

Table 2 contains those articles that were published as first author. Chronologically, these start with a publication in the area of constraint satisfaction [71]. Though no definitions or content of this paper is used in this thesis, the ideas of this paper have influenced several of the later articles. Some first definitions of consistency-based diagnosis have first been introduced by the author in the context of analysing quantum circuits in 2016 [72]. The first theoretical foundations in fault diagnosis for cyber-physical systems were published on the International Workshop on Principles of Diagnosis in 2018 [73]. This article was also a precursor to the work later published at AAAI. Given those articles, an overview of some of the diagnosis work was presented at the Machine Learning for Cyber-physical Systems conference [74]. Generating causal hypotheses through uninformed approaches (RQ3) has been investigated in a conference paper published by the author on the Annual Conference of the Prognostics and Health Management community 2019 [75]. In this article, the algorithm DDRC was introduced and evaluated qualitatively. An expert system for diagnosis using Bayesian inference to suggest observations and to compute diagnoses in cyber-physical systems was recently published at the IEEE Conference on Industrial Cyber

Physical Systems (ICPS) [16]. The system itself was inspired by the well-known EUREKA system of Bobrow et al. [15] that was used for diagnosing Xerox printers in the 1990s. The most recent article describes a theory and algorithm to automatically learn causal dependencies from data using Granger Causality (RQ3) and includes a first attempt of a specificity criterion, as well as the algorithm DDGD.

Usage	Reference	Ref.
Chp. 4, Pt. III	Diedrich, A., Buchholz, F., & Niggemann, O. (2022). Learning a Causal System Description for Diagnosing Physical Systems. In Proceedings of the 33rd International Workshop on Principles of Diagnosis, Toulouse, France.	[76]
Chp. 3, Pt. V	Diedrich, A., Deutschmann, P., & Junker, C. (2022). ServiceNavigator - A Bayesian Assistance System for Diagnosing Industrial Production Systems. IEEE Conference on Industrial Cyber Physical Systems	[16]
Chp. 3, Pt. III	Diedrich, A., & Niggemann, O. (2021, November). Diagnosing Systems through Approximated Information. In Annual Conference of the PHM Society (Vol. 13, No. 1).	[75]
Chp. 1, Pt. II	Diedrich, A., Balzereit, K., & Niggemann, O. (2021). First Approaches to Automatically Diagnose and Reconfigure Hybrid Cyber-Physical Systems. In Machine Learning for Cyber Physical Systems (pp. 113-122). Springer Vieweg, Berlin, Heidelberg.	[74]
Chp. 2, Pt. III	Diedrich, A., & Niggemann, O. (2018). Diagnosing Hybrid Cyber-Physical Systems using State-Space Models and Satisfiability Modulo Theory. In International Workshop on Principles of Diagnosis, Denver.	[73]
Chp. 2, Pt. III	Diedrich, A., Feldman, A., Perdomo-Ortiz, A., Abreu, R., Niggemann, O., & de Kleer, J. (2016, October). Applying simulated annealing to problems in model-based diagnosis. In Proceedings of the 27th International workshop on principles of diagnosis, Denver, CO, USA (pp. 4-7).	[72]
NA	Diedrich, A., Böttcher, B., & Niggemann, O. (2016, February). Exposing Design Mistakes During Requirements Engineering by Solving Constraint Satisfaction Problems to Obtain Minimum Correction Subsets. In ICAART (2) (pp. 280-287).	[71]

Table 2.: Main articles as first author

Several articles contributed to this thesis where the author was one of several co-authors. Table 3 provides an overview over these articles. Chronologically, the first articles as co-author were published in the domain of quantum diagnosis [77, 78] and ontologies [79, 80]. Following this, more recently an article was published describing a reference architecture for digital twins for artificial intelligence applications [60]. A work in some aspects parallel to the author's publication at the conference on Prognostics and Health Management [75] was published by Balzereit et al. on the International Conference on Cyber-physical Systems [81]. This work also relies on correlation analysis to find dependencies in cyber-physical systems. Finally, to support many of the experiments in the author's publications, simulations of tank systems were created. Within the diagnosis literature tank systems are accepted validation mechanisms, but only few are publicly available. To remedy this, the author contributed to two papers which created and openly published a benchmark for others to use. One was mainly written by Jonas Ehrhardt and one by Kaja Balzereit [82, 83].

Usage	Reference	Ref.
Chp. 3, Pt. IV	Ehrhardt, J., Ramonat, M., Heesch, R., Balzereit, K., Diedrich, A., & Niggemann, O. (2022). An AI benchmark for Diagnosis, Reconfiguration & Planning. IEEE Conference on Emerging Technologies in Factory Automation (ETFA), Stuttgart, Germany.	[83]
Chp. 1, Pt. II	Balzereit, K., Diedrich, A., Kubus, D., Ginster, J., & Bunte, A. (2022). Generating Causal Hypotheses for Explaining Black-Box Industrial Processes. IEEE Conference on Industrial Cyber Physical Systems	[81]
Chp. 3, Pt. IV	Balzereit, K., Diedrich, A., Ginster, J., Windmann, S., & Niggemann, O. (2021, July). An Ensemble of Benchmarks for the Evaluation of AI Methods for Fault Handling in CPPS. In 2021 IEEE 19th International Conference on Industrial Informatics (INDIN) (pp. 1-6). IEEE.	[82]
Chp. 1, Pt. I	Niggemann, O., Diedrich, A., Kühnert, C., Pfannstiel, E., & Schraven, J. (2021, May). A Generic DigitalTwin Model for Artificial Intelligence Applications. In 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS) (pp. 55-62). IEEE.	[60]
NA	Perdomo-Ortiz, A., Feldman, A., Ozaeta, A., Isakov, S. V., Zhu, Z., O’Gorman, B., ... & Biswas, R. (2019). Readiness of quantum optimization machines for industrial applications. <i>Physical Review Applied</i> , 12(1), 014004.	[78]
NA	Bunte, A., Diedrich, A., & Niggemann, O. (2016, September). Integrating semantics for diagnosis of manufacturing systems. In 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA) (pp. 1-8). IEEE.	[80]
NA	Bunte, A., Diedrich, A., & Niggemann, O. (2016). Semantics enable standardized user interfaces for diagnosis in modular production systems. In <i>International Workshop on the Principles of Diagnosis (DX)</i> .	[79]

Table 3.: Co-authored articles

1.6. Outline

This thesis is structured into four parts: The background, the solution, the evaluation, and the conclusion. For the background, Chapter 1 in Part II introduces different kinds of logics and explains how they can be used in industrial contexts. It also introduces a notion of causality and model learning, and the theory of consistency-based diagnosis. Thus, the reader is referred here for questions such as: What is the syntax and semantic of a particular logic? What notion of causality is used in this thesis? What are the basics of consistency-based diagnosis algorithms? Apart from these foundations, Chapter 2 Part II contains an overview of related research and shows how this thesis fits into prior work. Primarily, the chapter presents the research gaps that this thesis addresses. Part III introduces the main algorithms developed within this thesis. It answers questions such as: What diagnosis algorithm do I need? How do I obtain system descriptions automatically? In Part III Chapter 2 introduces algorithm HySD which is used for diagnosing cyber-physical systems. Chapters 3 and 4 introduce algorithms DDRC, DDA-IM, and DDGD which learn a propositional logic system description from process data. Chapter 5 presents a theory of how fault propagation can be described in cyber-physical systems. Practitioners can use this information to construct suitable system descriptions manually. Chapter 3 Part

IV presents empirical results with synthetic and with real observations from a variety of use-cases. Questions answered in this part could be: How do I track and model fault propagation? How do the developed algorithms perform in real applications? At the end, Chapter 4 discusses the computational complexity and runtime of the developed algorithms. Concluding the thesis in part V, the results are discussed and an outlook into future work is provided. In the appendix the output of the different algorithms is presented, as well as a description of the source code that was used to execute experiments in this thesis.

Part II.

Background

1. Foundations

This chapter introduces common definitions from the research fields of formal logics, diagnostics, and causality. It provides a brief introduction into the relevant concepts of propositional and predicate logic, the specialties of SMT logic, causality, consistency-based diagnosis, and learning models from data, as they are used in the context of this thesis.

1.1. Propositional Logic

For propositional logic we rely on adaptations of the definitions of Moura et al. [84] and Russel and Norvig [85]. The smallest building block of each propositional formula is a variable (also called a propositional symbol), which is defined as a sequence of at least one letter possibly followed by numbers. Examples of variables are: a , x_1 , or $tank$. A variable or its negation are called a literal. The concrete set of variables used within some use-case is called the propositional signature. We will use the terms formula and logical expressions interchangeably.

Definition 1. (*Propositional signature*) The propositional signature Σ is a set of variables.

From the signature it is possible to create propositional formulae. An atomic formula consists of only a single variable or its negation (the literal)

Definition 2. (*Atomic formula*) An atomic formula φ is an expression consisting of a single variable or its negation.

Atomic formulae are the smallest building blocks of propositional logic expressions. But usually they are assembled into more complex expressions to form propositional logic formulae.

Definition 3 (Propositional formula). If φ_0, φ_1 are propositional formulae (or atomic formulae), then these formulae together with their connectives $\varphi_0 \wedge \varphi_1, \varphi_0 \vee \varphi_1, \varphi_0 \rightarrow \varphi_1, \varphi_0 \leftrightarrow \varphi_1$ are also propositional formulae.

Please note that, with slight abuse of notation, in the following φ denotes single logical expressions (such as propositional formulae) as well as sets of formulae.

The semantics of a propositional formula are defined through its model (also called the semantic interpretation). The model describes how the truth assignments to the formula are interpreted within some universe. The connectives are the usual operations $\neg, \wedge, \vee, \rightarrow$, and \leftrightarrow , interpreted as the logical operations: negation (Table 4), conjunction (Table 5), disjunction (Table 6), implication (Table 7), and equivalence (Table 8). The binding priority

is in the same order, so unnecessary brackets can be avoided. The model is defined through the assignment of truth values to all of the variables of a propositional formula. In the following the symbol \top will be used to denote the logical value *true* and \perp will be used to denote the logical value *false*. All of the tables (Tables 4, 5, 6, 7, 8) are based on their logic-gate counterparts. As such, the tables for conjunction and disjunction are displayed with the least inputs possible, but their number of inputs can be arbitrary. The negation inverts any input values. The conjunction results in \top , when both input values are \top , similar to an AND-gate. The disjunction represents the logical OR-gate and results in \top , when either input values is \top . The implication (Table 7) together with conjunction is used for most logical expressions within this thesis. Through the truth table it is possible to conclude, if A then B, but not vice versa. For example, the implication may state: when all components within a system are healthy, then we expect to measure no fault. The equivalence, or bi-implication, is valid in both directions. In this thesis it is used to assign some values to a logical expression.

Definition 4 (Model). A model is a truth assignment $M : \Sigma_\varphi \rightarrow \{\top, \perp\}$ for all variables within a formula φ , where Σ_φ is the propositional signature for formula φ .

Using the model, the truth assignment $[[\varphi]]_M$ assigns some Boolean value to each variable within φ . Usually, the assignment is due to some semantic interpretation within a specific universe. In consistency-based diagnosis in general and for this thesis in particular, the semantics are defined such that variables denote the *health* of a component. To illustrate, $tank = \top$ means the tank is healthy (does not have any faults). Conversely, $tank = \perp$ means the tank is in a faulty state (because it is leaky, for example).

A propositional formula is satisfiable, if at least one model (i.e. semantic interpretation) can be found for which the formula evaluates to \top . This corresponds to solving the computationally expensive (NP-complete) Boolean Satisfiability Problem [86] and is the central task of consistency-based diagnosis.

A	$\neg A$
\perp	\top
\top	\perp

Table 4.: Truth table for a negation (\neg)

A	B	$A \wedge B$
\perp	\perp	\perp
\perp	\top	\perp
\top	\perp	\perp
\top	\top	\top

Table 5.: Truth table for a conjunction (\wedge). The number of inputs may differ.

A	B	$A \vee B$
\perp	\perp	\perp
\perp	\top	\top
\top	\perp	\top
\top	\top	\top

Table 6.: Truth table for a disjunction (\vee). The number of inputs may differ.

A	B	$A \rightarrow B$
\perp	\perp	\top
\perp	\top	\top
\top	\perp	\perp
\top	\top	\top

Table 7.: Truth table for an implication (\rightarrow)

In cyber-physical systems, propositional logic is used to describe a system's dependencies and capabilities. For example, propositional logic can describe which components influence

A	B	$A \leftrightarrow B$
\perp	\perp	\top
\perp	\top	\perp
\top	\perp	\perp
\top	\top	\top

Table 8.: Truth table for an equivalence (\leftrightarrow)

which sensor signal. Alternatively, propositional logic can be used to model the discrete relations within a system, such as with binary circuits, or in discrete event systems. Often, normal forms such as conjunctive normal form (CNF) or disjunctive normal form (DNF) are used in propositional logic. CNF is created in translating a logical formula into the form $\bigwedge_i \varphi_i$. So that it consists of several terms φ each with a number of disjunct literals. The different terms φ are then connected by conjunctions. Satisfiability algorithms use these and similar forms to check for satisfiability.

To illustrate the usage of propositional logic it can be stated that

$$\neg tank_{high} \wedge \neg tank_{low} \wedge valve_{open} \rightarrow sensor_0$$

which expresses simple facts about the world in the running example (see figure 2). Here, the signature is $\{tank_{high}, tank_{low}, valve_{open}, sensor_0\}$. This expression shows that when the tank is neither completely full, nor completely empty, the system is working normal (we expect $sensor_0$ to evaluate to true). Assigning truth values as part of a model M leads to:

$$\neg[[tank_{high}]]_M \wedge \neg[[tank_{low}]]_M \wedge [[valve_{open}]]_M \rightarrow [[sensor_0]]_M$$

where $[[\cdot]]_M$ denotes the semantic interpretation through model $M : \{\{tank_{high} \rightarrow \perp\}, \{tank_{low} \rightarrow \perp\}, \{valve_{open} \rightarrow \top\}, \{sensor_0 \rightarrow \top\}\}$, which will then be interpreted as

$$\neg\perp \wedge \neg\perp \wedge \top \rightarrow \top$$

It is evident that this formula, when interpreted with the defined connective priority, will consequently evaluate to \top and thus prove that the system is working normally and the formula is satisfiable. This, however, is the trivial case. Most challenges within this thesis focus on how to obtain information from unsatisfiable knowledge bases. So far, however, the problem with using plain propositional logic is its limited expressiveness.

1.2. Predicate Logic

Similar to the last section, this section is primarily adapted from the definitions of Russel and Norvig [85]. Predicate logic extends propositional logic through the use of functions and quantifiers. Compared to propositional logic we will use predicate logic only for few demonstrative examples. As such within this chapter, we will omit using quantifiers, as these are not necessary for the rest of the thesis. Predicate logic usually facilitates more complex use-cases by making general statements about the world and through improved reasoning. In propositional logic it is defined that literals are joined by connectives to form propositional formulae which can then be evaluated through a model. Predicate logic uses

constants, relations, and functions as building blocks. The relations and functions are also referred to as predicates (hence the name). Constants are those symbols that were denoted as variables in propositional logic, namely the names of objects such as *tank*, *valve*, or x_0 . Predicates are 1-ary in the case of functions and n -ary in the case of relations. Relations relate one object to one or more other predicates or constants. An example of this is $connected(tank, valve) \wedge healthy(tank)$, where $connected()$ is a 2-nary relation predicate that states the *tank* is connected to a *valve* and $healthy()$ is a 1-nary function predicate that states the constant *tank* has no faults. A function is also an action on objects, for example, when specifying that a tank is full, denoted as $full(tank)$. Here $full$ is a 1-ary function predicate that maps a tank to the state full. Please note that this is an example interpretation due to human attributed semantics. The smallest building block of a predicate logic expression is a term.

Definition 5 (Term). *Terms are predicate logic expressions or their negation $\varphi : f(t_0 \dots t_n)$, where f is a predicate, t are constants or predicates, and n is some natural number.*

Terms are usually specified within formulae. The smallest building block for predicate formulae are, similar to propositional logic, the atomic formulae.

Definition 6 (Atomic formula). *An atomic formula φ is an expression consisting of a single term or its negation.*

Also similar to propositional logic, predicate logic combines several atomic formulae into more complex formulae.

Definition 7 (Predicate logic formula). *If φ_0, φ_1 are predicate logic formulae (or atomic formulae), then these formulae together with their connectives $\varphi_0 \wedge \varphi_1, \varphi_0 \vee \varphi_1, \varphi_0 \rightarrow \varphi_1, \varphi_0 \leftrightarrow \varphi_1$ are also predicate logic formulae.*

Formulae are evaluated recursively. A semantic interpretation (i.e. model) is defined through

Definition 8 (Predicate logic model). *A model is a truth assignment $M : \Sigma_\varphi \rightarrow \{\top, \perp\}$ for all constants within a predicate logic formula φ , where Σ_φ is the set of constants of formula φ .*

The semantics are defined such that a predicate evaluates to true, when the meaning within the universe corresponds to the intended meaning, given the truth assignments to the constants. More specific, the model assigns truth values to the constants. The semantic interpretation then interprets the predicates according to the intended meaning within the universe to evaluate the predicate logic formula. The connectives are similarly defined as for propositional logic (see Tables 4, 5, 6, 7, 8). For example, the predicate logic term $full(Tank)$, can be interpreted by substituting the constant *Tank* by some instance t_1 and thus concluding that t_1 is full: $full(t_1) \rightarrow \top$. An important part of the predicate logic model is the intended interpretation. Predicates as words are inherently ambiguous. It is therefore necessary to assume an intended interpretation. The term $full(tank)$, therefore, should be interpreted that the water level is full, given some tank. It should not be interpreted that every tank within every possible universe is full.

As was already mentioned at the beginning, predicate logic also allows the use of universal \forall and existential \exists quantifiers. Although they are not used within this thesis, they shall be briefly mentioned for completeness. The quantifiers facilitate generalizations within a set of predicate logic expressions. For example, using the universal quantifier $\forall x \text{ full}(x) \wedge \text{tank}(x)$ is interpreted that all constants x are tanks and that those tanks are all full. Using the existential quantifier $\exists x \text{ full}(x) \wedge \text{tank}(x)$ means that within the universe there exists at least one tank that is full.

The usage of predicate logic, as far as it is used within this thesis, will be illustrated using a small example derived from the running example (Figure 2):

$$\begin{aligned} \text{constants} &= \{\text{pipe}_0, \text{pipe}_1, \text{tank}\} \\ \text{relations} &= \{\text{connects}/2\} \\ \text{functions} &= \{\text{full}/1, \text{empty}/1, \text{faulty}/1, \text{healthy}/0, \text{Tank}/1, \text{Pipe}/1\} \end{aligned}$$

The example consists of two pipes connected to a tank. These are the three constants that are interpreted. The relation signature is given by the 2-nary predicate *connects/2*. The functional signature contains functions to express the state of a tank, the presence of tanks and valves, and information about healthy and faulty components. With these it is possible to describe the running example as:

$$\begin{aligned} &\text{connects}(\text{pipe}_0, \text{tank}) \wedge \text{connects}(\text{tank}, \text{pipe}_1) \wedge \neg \text{empty}(\text{tank}) \\ &\wedge \text{healthy}(\text{pipe}_0) \wedge \text{healthy}(\text{pipe}_1) \wedge \text{Tank}(\text{tank}) \wedge \text{Pipe}(\text{pipe}_0) \wedge \text{Pipe}(\text{pipe}_1) \end{aligned}$$

As in propositional logic the model is interpreted by assigning truth values. One possible interpretation model, given the above example is:

$$\begin{aligned} \text{tank} &= t \\ \text{pipe}_0 &= p_0 \\ \text{pipe}_1 &= p_1 \end{aligned}$$

where t , p_0 , and p_1 are instances of the tank and pipes, respectively. Those instances are then entered into the knowledge base, such that

$$\begin{aligned} &\text{connects}(p_0, t) \wedge \text{connects}(t, p_1) \wedge \neg \text{empty}(t) \\ &\wedge \text{healthy}(p_0) \wedge \text{healthy}(p_1) \wedge \text{Tank}(t) \wedge \text{Pipe}(p_0) \wedge \text{Pipe}(p_1) \end{aligned}$$

These are then semantically interpreted. It is now possible to use satisfiability theory to prove whether this knowledge base is consistent. For example, should some interpretation now assign $\text{healthy}(p_0) = \perp$, while all other predicates are assigned \top the knowledge base would be unsatisfiable.

1.3. Satisfiability Modulo Theory

Many types of systems can be modelled using propositional logic or predicate logic. In Boolean circuit diagnosis, for example, the different circuits are modelled in propositional logic, the binary input variables for the circuits are assigned some value, and the set of propositional logic expressions is evaluated for consistency [72]. But cyber-physical systems

are harder to model. In cyber-physical systems a heterogeneous set of inputs must be evaluated and be represented in a logical model. Satisfiability Modulo Theory (SMT) allows to extend satisfiability by checking not only for assignments of binary values, but for real values and arrays as well. For this, satisfiability is augmented with a background theory such as linear arithmetic (SMT- \mathcal{LRA}), bit fields, difference arithmetic, or arrays. This thesis focuses on the linear arithmetic part. In the following, it will be explained how propositional logic is extended to check the satisfiability of these more complex theories. In this section, the definitions are adapted from Moura et al. [84].

In general, SMT is a predicate logic using quantifiers, free functions, or other extensions. But to integrate SMT with processed data from cyber-physical systems, only a subset of SMT is required, where multiplication, division, addition, subtraction, equality, and inequality checking are the only permitted operations [87]. All other operations are those adapted from propositional logic. This means SMT logic using linear arithmetic also avoids quantifiers and predicates. As such, the syntax of an SMT expression is defined in the following definitions. The signature is similar to the propositional logic signature.

Definition 9 (SMT signature). *The SMT signature Σ is a set of variables.*

The set of variables is defined in the same way as in propositional logic. The only difference is that in SMT-logic those variables can be real-valued instead of Boolean. A literal in SMT-logic is therefore only a Boolean variable or its negation. But those Boolean variables are a subset of all SMT-logic variables as those can also include real values.

Therefore, apart from the Boolean variables it is possible to define arithmetic formulae.

Definition 10 (Arithmetic atom). *An atomic formula (arithmetic) φ in SMT is an expression consisting of at least one variable and one variable or real number connected through $\{\cdot, +, -, \div\}$.*

Intuitively, an arithmetic atom corresponds to an algebraic term that can be evaluated to a scalar when all variables are instantiated.

Definition 11 (Evaluation atom). *An atomic formula (evaluation) φ in SMT is an expression consisting of at least one variable or real number connected through $\{=, \leq, \geq\}$.*

Evaluation atoms compare real numbers and are used semantically to lead to a truth value upon evaluation. In addition to the evaluation atom, a logical atom is defined on Boolean variables.

Definition 12 (Logical atom). *An atomic formula (logic) φ in SMT is an expression consisting of at least one variable or Boolean truth value $\{\top, \perp\}$ connected through $\{\rightarrow, \leftrightarrow\}$.*

Each of these atomic formulae can be integrated into a more complex SMT formula, encompassing an arbitrary amount of different atomic formulae.

Definition 13 (SMT formula (SMT- \mathcal{LRA})). *If φ_0, φ_1 are atomic formulae or SMT formulae, then $\varphi = \varphi_0 \oplus \varphi_1$ with $\oplus \in \{\wedge, \vee, \rightarrow, \leftrightarrow, =, \leq, \geq\}$ is also an SMT formula.*

Using the SMT formula and its decomposition into different atomic formulae, it is possible to define the semantic interpretation

Definition 14 (SMT Model). *A model is a truth assignment $M : \Sigma_\varphi \rightarrow \{\mathbb{R} \cup \{\top, \perp\}\}$ for all variables within a formula φ , where Σ_φ is the SMT signature for formula φ .*

Compared to propositional logic, the model can also assign real values to the variables. The semantics are defined such that an SMT expression evaluates to true, when the meaning within the universe corresponds to the intended meaning, given the truth assignments to the variables. The semantic interpretation then interprets the arithmetic, evaluation, and logical atoms according to the intended meaning within the universe to evaluate the SMT logic formula. The connectives are similarly defined as for propositional logic (see Tables 4, 5, 6, 7, 8). The model uses multiplication, division, addition, subtraction, and comparisons through interpretation of the operator \oplus in the usual manner with $\oplus \in \{\cdot, +, -, \div, \wedge, \vee, \rightarrow, \leftrightarrow, =, \leq, \geq\}$. If a symbol is Boolean, \oplus is restricted to Boolean operators.

Satisfiability theory seeks an assignment of all variables of a term to achieve satisfiability. Besides the assignment of Boolean values as in propositional and predicate logic, the model also assigns real values. More formal, given a formula φ , a model M , a variable assignment $\alpha \in \{\mathbb{R}, \top, \perp\}$ is computed such that $[[\varphi]]_M = \top$. In many cases several complex terms φ exist. To compute whether the set of SMT expressions is satisfiable it is defined

Definition 15 (Satisfiability of (SMT – \mathcal{LRA})). *Provided some model M exists, a set φ of SMT – \mathcal{LRA} is satisfiable, when $[[\bigwedge_{\varphi_i \in \varphi} \varphi_i]]_M = \top$.*

In short this is written as $M \models_T \varphi$, where T is a certain theory, in this case linear arithmetic. To illustrate, when the pressure in a tank is measured there must be some way to associate this measurement with some symbol. Those symbols can then be evaluated through the propositional logic expressions $tank_full \wedge valve_closed \rightarrow \perp$. SMT, therefore, is a suitable extension to model this problem, for example, as $tank_level \geq 0.9 \wedge valve_state = \perp \rightarrow \top$. In this case, SMT is interpreted such that $tank_level \geq 0.9$ models the propositional logic symbol $tank_full$ and $valve_state = \perp$ models the state of the valve being closed. This power of SMT- \mathcal{LRA} logic to integrate (in-)equations and other arithmetic operations is the useful property being leveraged to diagnose cyber-physical systems.

To perform diagnosis using SMT – \mathcal{LRA} it is not only necessary to check the satisfiability of a set of logical expressions φ . In many cases it is known that the set is unsatisfiable and an algorithm must find a variable assignment such that the maximum number of clauses within the expressions is satisfied, where a clause is a set conjunctions.

Definition 16 (Maximum Satisfiability of (SMT – \mathcal{LRA})). *A maximum satisfiable set (mss) of a set of SMT – \mathcal{LRA} expressions φ are those expressions that can be satisfied even though φ itself is unsatisfiable, with $mss = \operatorname{argmax}_{\alpha, M} \varphi$, where α is a complete assignment to all variables within φ , using a single model M .*

Later in the thesis we will use this variable assignment to integrate values read from a cyber-physical system into the formulae. The complement of the maximum satisfiable set

is the minimum unsatisfiable set (also called minimum core). Computing these sets is NP-complete, as the known NP-complete problems vertex cover and SAT can be reduced to the problem of computing the maximum unsatisfiable set [88]. In later chapters *SMT – \mathcal{LRA}* will be used to perform diagnosis by representing the health of components of a cyber-physical system through symbols and evaluating residual values through in-equations.

1.4. Reasoning

The previous sections have discussed propositional logic, predicate logic, and *SMT- \mathcal{LRA}* with their syntax and semantics. This section shows how new knowledge can be inferred from using different reasoning techniques. We will limit the discussion on propositional logic, but all reasoning approaches are generalisable to predicate logic and *SMT- \mathcal{LRA}* .

In the past, logical reasoning was understood to either reason from the general case to the special case (deduction), or from special cases to the general case (induction). In both types of reasoning, valid knowledge could only accumulate. So, starting from a set of known and valid facts, a reasoning system could only infer new valid facts. Obviously, using such a system in the real world can lead to severe limitations, when new knowledge emerges which may be contrary to the already known facts. For example, imagine the thought experiment of Plato's allegory of a cave. Within the experiment the subjects are supposed to be in a cave facing the cave's wall. Behind them is a fire illuminating the cave. Having lived their entire lives in this configuration, while visitors only stand behind them and casting a shadow, the subjects need to assume that other humans are only black shapes on the cave's wall that can speak their language. So their knowledge base consists of facts which state that humans are only two dimensional black shadows on walls. They could deduce that if one of those human-shapes has a voice, the others should have voices as well. But if one of the subjects would turn around and they would see real, three-dimensional humans, their knowledge-base would become invalid and classical inductive reasoning would not be possible anymore.

More technical, classical reasoning is done through Modus Ponens and Modus Tollens [85]. Assume a knowledge base contains the propositional logic formula $A \rightarrow B$. Further, it is known that A is a valid fact. Using Modus Ponens, it is possible to conclude

$$\frac{A \quad A \rightarrow B}{B}$$

which leads from the knowledge of A and the expression $A \rightarrow B$ to B . Contrarily, Modus Tollens allows reasoning denying the consequence. Assuming the existence of a propositional logic formula $A \rightarrow B$ and the known fact $\neg B$, then it is possible to conclude

$$\frac{A \rightarrow B \quad \neg B}{\neg A}$$

But reasoning by relying on facts only and never considering that facts could be invalid or invalidated over time, poses some significant limitations. To mitigate these constraints, non-monotonic reasoning was introduced.

One approach for non-monotonic reasoning was developed by McDermott and Doyle [30, 31] with their modal logic. This logic introduces special operators to denote if a logical statement is *just* possible or whether it is necessary. As such, a knowledge base may include several statements about some fact in parallel, all of those may be possible. When a new fact is entered through interaction with the environment, some of those possible statements are invalidated, thus making the knowledge base more precise.

Reiter [89] developed a theory called default reasoning. Default reasoning contains logical statements that are assumed to be true unless proof to the contrary exists. This enables practitioners to develop systems which assume how the world around them might look and behave, while facilitating an update of this belief over time. Formally, Reiter denoted this reasoning through

$$\frac{A : B}{C}$$

with statements A, B, C . It is interpreted that when A is true and it is consistent with the theory that B is true, then it is possible to conclude that C is true. For example, if on some oven an activity lamp is activated and it is consistent to assume that the oven works correctly, then it can be concluded that a plate is heated. Later, with new knowledge it might be concluded that although the lamp is on, the plate is still cold. It can then be deduced that the theory that the oven works correctly is not consistent anymore with the observations.

A non-monotonic reasoning approach related to diagnosis is abductive logic programming [90, 91]. While deduction reasons from the general to the specific and induction reasons from the specific to the general, abduction is the process to find justifications for observations within a given theory. Formally, Minker [90] defines three conditions that need to hold for abductive reasoning, given a set of logical expressions Φ , an observation α , and explanations δ :

1. $\Phi \cup \delta \models \alpha$
2. $\Phi \cup \delta$ is consistent
3. δ is minimal with respect to set inclusion

Together, these conditions mean that a consistent theory and a set of logical expressions are needed, which can be used to generate justifications and explanations for observations. The difference between abductive reasoning on consistent knowledge bases and consistency-based diagnosis is the way the logical statements are formulated [92]. While abductive reasoning searches for justifications for some observations, a propositional logic formula could be formulated as $\alpha \rightarrow a \wedge b$, where α is some observation and a, b are facts. In technical terms, such a formulation could be viewed as *backwards*. In consistency-based diagnosis, however, these expressions are formulated *forwards*, such that a propositional logic formula as $a \wedge b \rightarrow \alpha$ is formulated. The formal definition of consistency-based diagnosis is presented in Section 1.6.

1.5. Causality

To investigate dependencies within any kind of system, it needs to be known how components within the system interact. Analysing the causality within a system shows which components influence which other components and shows, how faults propagate throughout the system. Pearl [54] has provided a comprehensive formalisation of causality. In the following, the concepts which are most important for this thesis are reviewed and adapted.

Usually causality is defined between signals. Namely, if one signal changes its value and then, after a small but reasonable temporal interval, another signal changes its value, and this change does not occur without the first signal, it is said that the first signal causes the second signal [93]. Components usually subsume several signals. Thus, causality between two components means that one component emits some effect that causes a change in another component that cannot be explained by any other effect. For example, an increase in water throughput by a pump causes a tank to fill. However, no methodology exists to proof whether some effect in some component is caused by exactly the previous component's effect. If one claims that an increase in water throughput in all possible worlds causes the water in the tank to rise, one forgets it is also possible that some agent inputs water from some other source into the tank that is unknown to the observer. Generally, causality researchers assume that some ground-truth exists, where all causes of some effect can be known and enumerated. This is often not the case in cyber-physical systems. Causes are mostly limited to common occurrences that human experts think of. Therefore, causality in cyber-physical systems is always an approximation of some true, unknown causality. In this thesis such approximated causality is only referred to as dependencies, since true causality in the sense of Pearl cannot be determined. In causality research, dependencies within systems are usually described using a directed-acyclic graph $D = (V, E)$ with V being the set of vertices and E being the set of edges. A possible causal graph for the running example is depicted in Figure 4. Vertices A to $F \in [0, 1]$ are causal, random variables. This means their value expresses a belief, given some external observations, which is often done through assigning probabilities. Vertices A, B, C denote the health of components (defined through some probability). As such, they express the belief in the nominal working behaviour of the two pipes (that the pipes are neither leaky nor clogged) and the tank (not to be leaky). For example, if some measurement would indicate changes in the water flow through the pipe, the belief in the correct working of the pipe would decrease. Vertex E indicates the probability, whether the water level within the tank is as expected. Vertex F indicates the probability whether the water pressure at the end of the output pipe is expected to be correct. Vertex D is an assumed, unobserved confounding variable and surmises information about the probability, whether the correct water pressure is applied to the input pipe.

A causal graph can be captured within a causal model. Ibrahim et al. [94] who refer to Halpern [95] define a binary causal model that we adapt here to the domain of real numbers such that it fits with later definitions

Definition 17 (Causal Model). *A causal model is a tuple $M = ((U, V, R), F)$, where $U \in \mathbb{R}^U$ is a set of exogenous variables, $V \in \mathbb{R}^V$ is a set of endogenous variables, R associates every variable with a value in \mathbb{R} , and F associates to each variable $v \in V$ a function that determines the value of v*

Exogenous variables are those variables that are external to the model. In cyber-physical systems these are usually inputs or parameters. The endogenous variables are those that are part of the causal model. Given values for the exogenous variables, the endogenous variables are assigned values through functions F . Pearl refers to these functions as structural equations [54]. The assignment R can be compared to the value assignments α in earlier sections. Usually, Pearl interprets the exogenous and endogenous as random variables. In this thesis, we will use the idea of the causal model and, in particular, of the structural equations to create models using arbitrary real-valued variables in domains outside of probability theory. Further, we will relax the graph structure to be a directed graph, rather than a directed, acyclic graph.

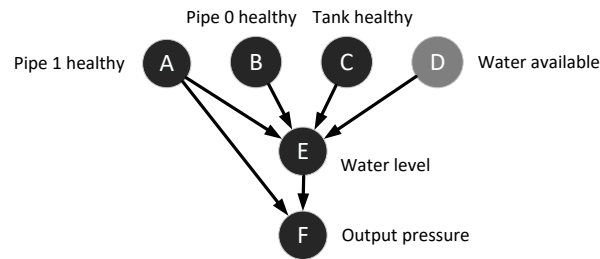


Figure 4.: The causal graph for the running example (Figure 2)

First, we will briefly look how probabilities are calculated, when using random variables within a causal graph as in Figure 4. This is the standard way in traditional causality research. The probabilities are calculated through traversing the graph from causes to effects and through using the chain rule. For the example in Figure 4 this would translate into: $P(a, b, c, d, e, f) = P(F = f | A = a \wedge E = e)P(A = a)P(E = e | A = a \wedge B = b \wedge C = c \wedge D = d)P(B = b)P(C = c)P(D = d)$, where the lower-case characters are actual values. So, each concrete value $v = V$ for some random variable V is calculated through the structured equations F , which calculate the value of v by taking the parent node's value and adding some random noise disturbance. To illustrate, in the example structural equations would exist to calculate a water level (some differential equations), the component's health (some piece-wise equations) and so on. It is the task of the designer of the dependency graph to find suitable structural equations. In later chapters it is shown how those structural equations correspond to different types of models in cyber-physical systems. In the context of cyber-physical systems, creating causal models (such as those in Definition 17) manually is too costly and complex. Therefore, models are often learned from data. Pearl assumes that a ground truth causal model exists and that data is only used to *fill out* the causal model with empirical values [54]. For this, he distinguishes two types of causality: inferred causation and genuine causation. While inferred causation describes causal behaviour according to some data, genuine causation relies on some external, known ground-truth.

Definition 18 (Inferred Causation). *A variable $X \in [0, 1]$ has a causal influence on variable $Y \in [0, 1]$, if a directed path exists from X to Y in every minimal graph structure consistent with some external data.*

To infer causation, Pearl requires the existence of a set of data, often in the form of observations from a system under investigation, that can be used to discover causal

behaviour. In particular, the data is used to discover some causal graph, which is then approximated to some known ground truth. The term inferred causation is used to denote a causal graph that has been approximated, but may still be different, from the ground truth. Finding such an inferred causation is the task of algorithm DDGD in Chapter 4 Part III.

Definition 19 (Genuine Causation). *A variable X has a causal influence on Y if there is a third variable Z and a context S , both occurring before X , such that Z and Y are dependent given S , and Z and Y are independent given S and X , where the context S is some possible universe with an assignment to the random variables X, Y, Z .*

So for genuine causation between two variables to exist at least three variables are necessary: X only genuinely causes Y , when there exists some other variable, for example Z , which is independent in the context (such as a variable assignment according to Def. 17) S , but is dependent given $X \cup S$. This means that genuine causation can only be determined with respect to some outside context and only in relation to other variables.

However, obtaining the genuine causation is in practical terms often impossible. To analyse the effects of causal variables within a causal model, operations can be performed to simulate behaviour. Using the so called *do-operator*, variables can be set to actual values $x = X$. The do-operator was introduced by Pearl et al. [54] to allow the analysis of causality through the use of counterfactuals [96]. Nowadays, the do-operator has become a useful tool to analyse causal processes asking *what if*, by assuming some random causal parameter to be of some value (often denoted as setting $x = X$) and propagating the values throughout a causal diagram. Practitioners can use the do-operator to set some of the variables within a causal graph to some fixed, assumed value. By calculating the other values within the causal graph under the assumption of this fixed value, it is possible to determine which values are causally dependent on this variable. Formally, this is computed through the causal effect.

Definition 20 (Causal Effect). *Given two disjoint sets of variables \mathcal{X}, \mathcal{Y} , the causal effect of \mathcal{X} on \mathcal{Y} , denoted as $P(y \mid do(x))$, is a function from \mathcal{X} to the space of probability distributions on \mathcal{Y} . For each concrete instance x of \mathcal{X} , $P(y \mid do(x))$ calculates the probability of $\mathcal{Y} = y$ induced by deleting from the model all those structured equations corresponding to variables \mathcal{X} and substituting $\mathcal{X} = x$ in the remaining equations. The resulting change in the remaining values is called the causal effect.*

Calculating the causal effect corresponds to a substitution of all occurrences of a random variable within a causal model with a fixed, concrete instance of some other value. Technically, the structured equations calculating the probability for some variable are removed and replaced through the value within the do-operator. In the running example this could, for example, be the operation $pipe0healthy = \top$ in propositional logic. This enables an analyst to *simulate* how probabilities change within a causal model, when some scenario is assumed. This simulation is also known as calculating counterfactuals. Using the do-operator to compute counterfactuals, an analyst is enabled to assume what would happen, if variables had other values. Reevaluating the structured equations with such an assumed value then leads to new probabilities that reflect the assumed value. Halpern [97] and Bochman et al. [56] have extended this theory to account for actual causality. The difference to Pearl's definition is that some causal dependencies (modelled in logic) are only relevant in a suitable context.

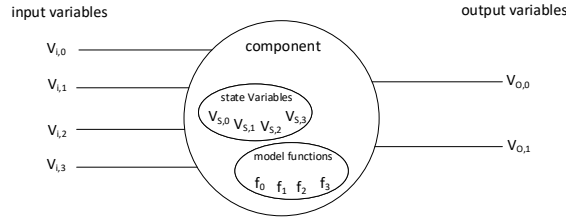


Figure 5.: Structure of a single component (Def.: 21)

Next, we will formally adapt the ideas of a causal model into the domain of cyber-physical systems. For this, remember the intuition that the true causality according to Pearl cannot be achieved. Instead, the formalisations below help practitioners to model value propagations and dependencies throughout a system. Below, a formal definition of components is introduced, where components are defined through their inputs, outputs, states, and internal functions (structural equations). The choice of these functions is entirely dependent on the use-case and can, for example, entail single electronic capacitors or huge inverters for motor control. The defined functions are often specified by experts. But some approaches, such as the one by Yuan et al. [98] have shown to learn behaviour data-driven. The basic building block of a cyber-physical system is defined here as a general component, based on previous work [70].

Definition 21 (Component). *Each component is a tuple $(\mathcal{V}, \mathcal{F}, \text{name})$, with $\mathcal{V} = \mathcal{V}_i \cup \mathcal{V}_o \cup \mathcal{V}_s$, where $\mathcal{V}_i \in \mathbb{R}^{\mathcal{V}_i}$ is the set of input variables, $\mathcal{V}_o \in \mathbb{R}^{\mathcal{V}_o}$ is the set of output variables, and $\mathcal{V}_s \in \mathbb{R}^{\mathcal{V}_s}$ is the set of state variables. \mathcal{F} is a set of component model functions $f_p \in \mathcal{F}$ over the variables \mathcal{V} , with $f_p : \mathbb{R}^{\mathcal{V}} \rightarrow \mathbb{R}$, where $p \in \mathbb{N}$ denotes an additional function index. **name** is the symbolic name for the component.*

Definition 22 (Set of Components - CM). *The set of components $\{(\mathcal{V}_0, \mathcal{F}_0, \text{name}_0) \dots (\mathcal{V}_n, \mathcal{F}_n, \text{name}_n)\}$, with variables \mathcal{V}_n , component model functions \mathcal{F}_i , **name**_{*i*} \in COMPS, COMPS being the set of symbolic name for the components, and $i, n \in \mathbb{N}$.*

This definition of components (Figure 5) allows us to formulate the system's parts through more elaborate formalisms such as qualitative physics, temporal causal graphs (TCG), and their related bond-graphs [25]. Please note that the functions \mathcal{F} can be compared to the structural equations in Pearl's causal models. The connections between components are formulated within the connection model.

Definition 23 (Connection Model - CON). *The connection model is a directed graph $D = (V, E)$, where the vertices denote the components $v \in V$ and edges $e(v_j, v_k, p_{-i}) \in E$ are annotated by a connective p_{-i} , where $-$ is a placeholder for effort e or flow f . Edges connect input $v_j \in \mathcal{V}_i$ and output variables $v_o \in \mathcal{V}_o$ of components. $p_{e,i}$ and $p_{f,i}$ are sets describing the type of connection either as flow $p_{f,i} : v_j + v_o = 0$, or effort $p_{e,i} : v_j = v_o$. $f \in \mathbb{N}$ is the flow function index, $e \in \mathbb{N}$ is the effort function index, \mathcal{P} is the set of connection functions, and $j, k \in \mathbb{N}$ are component indices.*

In the following, the connection predicate p_{-i} is left out for clarity. The connection model (Figure 6) complements the component model in that it relates the components to each other.

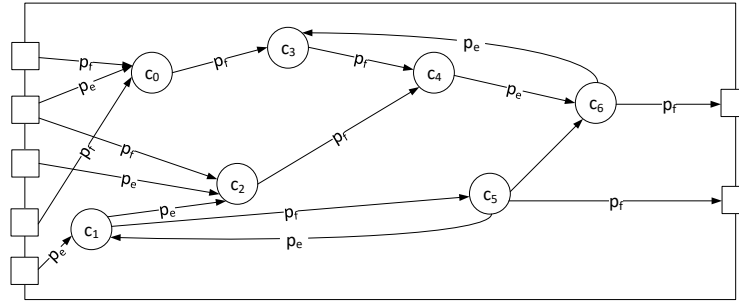


Figure 6.: Structure of an exemplary connection model (Definition 23). Edges denote the distribution of flow and effort. Input and output variables are subsumed within each vertex.

It must be noted that, according to Definitions 21 and 23, the input and output variables are connected to each other. But for readability Figure 6 subsumes all the variables relevant for each component within one vertex. Inputs are either connected to previous components or describe external inputs, for example, the ambient temperature. The component and connection models themselves are used to propagate output values from one component to the inputs of subsequent components.

Theoretically, two types of component models exist, although both are described through Definition 21. These are: i) Components that do not monitor the system. These are most components that are usually found in cyber-physical systems, such as conveyors, tanks, pipes, valves, pumps, inverters etc. For these, the component model propagates their values from one component to the next using the connection model. ii) Components that monitor the system. We will usually refer to those components as sensors. These read parts of the current values within the system. Component models that describe sensors predict the sensor's expected value. Consistency-based diagnosis will compare this prediction to the real values read from the actual cyber-physical system. Such a prediction will be denoted as \hat{a} and will be used to evaluate, whether a cyber-physical system behaves as expected. In particular, these are used as inputs for the residual generation functions which are introduced below.

Component models for diagnosis can often be simplified into two distinct expressions: the component's health state and its output values. A diagnosis algorithm only needs to know, whether a certain component is healthy, without regard of its internal function. Also, to propagate values from one component to the next it is only necessary to know the component's output while treating the component itself as a black-box. Therefore, residuals and output models are formally introduced.

The observable parameters of cyber-physical systems and their components consist of inputs, outputs, and state variables, while assuming full observability. To improve analysis of these systems several approaches use observers [99] or structural equations [42] to calculate residual values. Many approaches exist that calculate residual values for fault diagnosis [68]. A residual is defined as

$$h(\mathbf{x}, \mathbf{u}, \mathbf{y}, \lambda) = \begin{cases} 0; & \text{no fault} \\ \neq 0; & \text{else} \end{cases} \quad (1.1)$$

where $\mathbf{x} \in \mathbb{R}^x$ is the set of the component's state variables, $\mathbf{u} \in \mathbb{R}^u$ is the set of the components input variables, $\mathbf{y} \in \mathbb{R}^y$ is the set of the components output variables, and $\lambda \in \mathbb{R}^\lambda$ is a set of parameters. Variables \mathbf{x} , \mathbf{u} , and \mathbf{y} can be interpreted as concrete instantiations of the variables \mathcal{V} in Definition 21. Usually, the set of parameters λ contains thresholds τ which provide guard conditions around the input, output, and state variables. These determine when one of the input parameters exceeds its expected value. The set of all residuals is denoted as \mathcal{R} . Typically, $h(\cdot) : \mathbb{R}^{x \times y \times u \times \lambda} \rightarrow \mathbb{R}$ is implemented through thresholds, analytical equations or through statistical approaches [68, 88]. Without loss of generality, the underlying assumption is that the system and its residuals were designed in such a way that during the system's normal operation the residual values are zero. In the simplest case the residual is a difference between a variable's set point and its actual sensor reading $\alpha_i = \hat{\alpha}_i - m_i$, where $\hat{\alpha} \in \mathbb{R}$ is the model prediction and $m_i \in \mathbb{R}$ is the sensor value. $\hat{\alpha}$ is determined through *CM* and *CON* by propagating values from the system inputs to system outputs and sensors. $\hat{\alpha}$ is then read from those components representing sensors within *CON*. More details will be shown using the theoretical framework of fault propagation in Chapter 5 Part III. To model faulty behaviour and to show the propagation of faults it is necessary to introduce an output model and its extended fault-augmented model.

Definition 24 (Output model). *An output model is a function $\mathbf{y} = n(\mathbf{x}, \mathbf{u})$ with $n : \mathbb{R}^{x \times u} \rightarrow \mathbb{R}^y$ and $\mathbf{x} \in \mathbb{R}^x, \mathbf{u} \in \mathbb{R}^u$.*

Definition 25 (Fault-augmented model). *A fault-augmented model is an output model $n(\mathbf{x}, \mathbf{u})$ augmented with discrete analytical functions $f_m : \mathbb{R}^{\Xi_m} \rightarrow \mathbb{R}$ and $f_a : \mathbb{R}^{\Xi_a} \rightarrow \mathbb{R}$. The functions are modelled either multiplicative or additive $\mathbf{y} = n(\mathbf{x}, \mathbf{u})f_m(\Xi_m) + f_a(\Xi_a)$ with a suitable set of parameters $\Xi_m \in \mathbb{R}^{\Xi_m}, \Xi_a \in \mathbb{R}^{\Xi_a}$.*

Using the fault-augmented model, we only look at the output model instead of input or state models, since within cyber-physical systems faults propagate through (at least theoretically) measurable edges connecting components within the system's graph. Noise and other hidden values within a component are ignored, since these result in unknown deviations in the output model. The faults affecting the input, state, or hidden variables of a component are therefore assumed to be relevant only to the component's output \mathbf{y} . To illustrate, assume some pump with one input, one output and some internal variables such as current, voltage, flow, noise, and some fault model. For the pump's functioning, we only care about the change in magnitude in the output flow, regardless of the actual source of the change within the component.

The next section introduces the foundations of consistency-based diagnosis. It will be shown how system models are specified in logic and how diagnosis is understood formally.

1.6. Consistency-based diagnosis

The main part of the thesis shows how to generate diagnosis models and apply diagnosis algorithms to cyber-physical systems. The research field of fault diagnosis intersects with the research fields of artificial intelligence [17], control systems theory [12], and physics [33]. In order to develop a common understanding over the intersection of these fields for this thesis, this section lists the common definitions.

Isermann and Balle [24] have created a compilation of common definitions, some of these have been adapted in the following analysis.

Definition 26. (*System*) *A system is a combination of interacting and interdependent components that form a unified assembly and have a single, describable purpose.*

Systems can appear in many forms in the real world and in academic model use-cases. A system could, for example, be the temperature exchange of different media on earth, the body of a single animal, a steam engine, an airplane, or a piece of software. The extent of a system depends on the granularity with which one wishes to observe the inputs, outputs, and inner functioning of a system. This includes the decision about the limits of the system. A model captures the system behaviour and its most important influences. Two kinds of models can be distinguished: analytical and data-driven. Analytical models are constructed through human reasoning, for example in the case of Kepler's description of planetary orbits, or in simpler ways through the creation of expert-defined expressions. Data-driven models are based on observations often through statistical and machine-learning algorithms.

Definition 27. (*Fault*) *A fault is an unpermitted deviation of at least one parameter from the expected behaviour.*

Some faults can be glitches in the system's intended behaviour and do not lead to any kind of observable unintended behaviour. If a fault persists, or if the fault is severe enough to negatively influence the system's execution, a failure will occur.

Definition 28. (*Failure*) *A failure is the permanent interruption of the system's correct functioning behaviour.*

Failures in production machinery are often costly and may put human lives in danger. Additionally, a failure leads to a repair operation which increases the production system's downtime and may negatively affect other connected systems. As a result, the goal of any diagnosis system should be to identify those failures as fast as possible.

Definition 29. (*Symptom*) *A deviation of an observable quantity from normal behaviour.*

Sensor values that deviate from the predicted behaviour show the symptom of a fault. Since faults usually cannot be observed directly, enough symptoms have to be recorded in order to determine the location of a fault. Residuals show the mathematical distance from which the model's prediction deviates from the sensor's measurements. The act of determining whether or not a fault occurred in the system is called fault detection.

Definition 30. (*Fault detection*) *Determination of whether or not a fault is present in the system at a given time.*

Definition 31. (*Fault isolation*) *Determination of the kind, location, and time of occurrence of a fault.*

Once a fault is detected it is important to determine its exact kind, time of occurrence, and location. This is done by the process of fault isolation and comprises reasoning about the given information, the construction of inferences to determine unmeasured parameters, or the determination of missing values.

Definition 32. (*Fault diagnosis*) *Comprises the processes of fault detection and isolation.*

The research field of consistency-based diagnosis relies on terminology and common definitions that have been kept and been improved upon through several decades [13]. The definitions in this chapter are mainly adapted from Reiter [17], Feldman et al. [22], and Metodi et al. [100]. Some definitions were changed slightly to correspond to later chapters in this thesis. The original formulation can be found in a journal article [70]. The goal of consistency-based algorithms is to ensure the consistency between the collection of all logical expressions describing the system and a set of observations, such that

$$SD \wedge OBS \wedge COMPS \text{ is satisfiable} \quad (1.2)$$

where the system description SD is a qualitative model in the form of a set of logical expressions (often in propositional or SMT logic) describing the normal behaviour, $COMPS$ is the set of components, and OBS is the set of observations from sensors (process data). Both will be described in later chapters in more detail.

First, sensor readings and observations are introduced formally.

Definition 33 (Sensor Reading). *A sensor reading $m \in \mathbb{R}$ is a measured physical value.*

A sensor reading is usually a real number describing values such as temperature, pressure, current, flow, derivatives, or other features generated from control theoretic methods etc. Sensor readings may be subject to denoising or other quantitative pre-processing methods which are not discussed in this thesis. A sensor reading can be modelled $m = f(\mathbf{u}, \mathbf{y}, \mathbf{x})$, $\mathbf{u} \in \mathbb{R}^u$, $\mathbf{y} \in \mathbb{R}^y$, $\mathbf{x} \in \mathbb{R}^x$, $\mathbf{m} \in \mathbb{R}$, $f : \mathbb{R}^{u \times x \times y} \rightarrow \mathbb{R}$, interpreted as being some function depending on a subset of inputs, outputs, and states. The drawback is that in the real world m may contain noise and, even more important, may behave unexpectedly in the presence of faults. Usually, sensor readings are in some way discretised through, for example, residuals. The model predictions for the residual computation is obtained using the component model.

For evaluating sensor values, symbolic logic is a suitable and intuitive solution. Using logic, it is possible to categorise system behaviour into *ok* and *not ok* and perform reasoning.

Definition 34 (Observation). *A single observation $\alpha \in \{\top, \perp\}$ is a discretised assignment $\alpha = d(m)$ to some or all inputs and outputs of a cyber-physical system, where $d(\cdot)$, with $d : \mathbb{R}^m \rightarrow \{\top, \perp\}$, is a suitable discretisation function. OBS is the set of observations.*

If necessary, this binary interpretation can be relaxed to ternary logic $\{-1, 0, 1\}$ to be, for example, used with qualitative physics models [27, 57]. Here 1 and -1 denote deviations above and below the normal value, respectively, while a value of 0 denotes normal working behaviour. In the binary view of Definition 34, a component can only be in one of two states: healthy or unhealthy. For healthy components the model assigns the truth value \top to the corresponding component's symbolic name. A logical solver finds these truth values for all symbols within the knowledge-base through a variable assignment.

Definition 35 (Health Assignment). *The health assignment specifies $(d(c_i) = \top) \vee (d(c_i) = \perp), \forall c_i \in \text{COMPS}$. $i \in \mathbb{N}$ is the index.*

Translating the health assignment into logical expressions is done through the observation evaluation.

Definition 36 (Observation Evaluation - φ_d). *φ_d is the set of expressions evaluating generated observations. Each term φ is of the form $\varphi_{d,i} : \tau_l \leq m_i \leq \tau_h \rightarrow \alpha_i$, where $\tau_l, \tau_h \in \mathbb{R}$ are some threshold values and $r_i \in \mathbb{R}$ is the residual.*

Instead of using the measurement m_i it is also possible to evaluate residuals r_i . The component health function $h(\cdot)$ (Eq.: 1.1) is implemented through the observation evaluation. All component models for a system are contained within the set CM , which is the set of component model functions \mathcal{F} (see Definition 21). In order to conform to standard diagnosis terminology, we aggregate CM and CON within the system description SD . The system description SD is a tuple (CM, CON) with CM being a set of component models $CM = \{f_0, f_1, \dots, f_n\}$ and CON being the connection model. The idea with using SMT logic is to obtain a set of expressions to represent SD and α for cyber-physical systems. Therefore, a more practical formulation of SD can be found. An example expression would be $c_0 \wedge c_1 \rightarrow \tau_l \leq r_i \leq \tau_r$, where $c_0, c_1 \in \text{COMPS}$, $r_i \in \mathbb{R}$ is the residual, and $\tau_l, \tau_r \in \mathbb{R}$ are some thresholds. Thresholds τ can be defined through experts or can be determined using the component model CM .

A logical solver finds the truth values for the health assignment for all symbols within the knowledge-base. In case of faults the knowledge-base is no longer satisfiable, since some expressions no longer evaluate to \top . The task is to find the minimum number of expressions, whose removal would lead to a satisfiable knowledge-base. Each set of expressions leading to an unsatisfiable knowledge-base is labelled as a conflict. From the set of all conflicts, hitting sets can be computed. A hitting set is a set-cover problem, where a union of subsets is selected, such that some larger set can be explained. In other words, a hitting set identifies elements in subsets, such that the collection of all subsets explains the original set [17]. These are determined by selecting the minimum number of elements from each subset such that at least one element from each subset is chosen.

The logical expressions describing a cyber-physical system for diagnosis can be formulated as follows.

Definition 37 (Logical System Description). *A system description formulated in a suitable logic is a set of formulae $\Phi = \varphi_c \wedge \varphi_d \wedge \varphi_p \wedge \varphi_h$, where each $\varphi_{c,i} \in \varphi_c$ is a logical description of the component model function f_p , φ_d is a set of threshold descriptions, φ_h is a set of health states, and φ_p is a set of connections.*

The idea with using a logical system description Φ is to obtain a set of expressions to represent SD and OBS for cyber-physical systems. The logical system description comprises subsets of individual expressions describing the connections, health states, and observations from the components within the cyber-physical system. This description is used as the basis for diagnosis.

Definition 38 (Diagnostic System). *A diagnostic system is a three-tuple $(\Phi, COMPS, OBS)$, where Φ is the system description formulated in logic, $COMPS$ is the set of components, and OBS are the observations, such that $\Phi \cup COMPS \cup OBS$ is satisfiable.*

Note that here we exchanged SD with Φ , as we refer to SD as a general system description, which could be logical expressions, equations, or black-box models, and use Φ as the system description of logical expressions that can be used for diagnosis. The above definition of the diagnostic system are in accordance with the definitions from Reiter [17] and de Kleer [19].

Definition 39 (Diagnosis). *A diagnosis ω is a hitting set derived from the set of conflicts, which satisfies $\Phi \wedge OBS \wedge COMPS_\omega \wedge COMPS_{COMPS \setminus \omega}$*

where $COMPS_\omega$ are the faulty components and $COMPS_{COMPS \setminus \omega}$ are the remaining healthy components. De Kleer [101] mentioned that obtaining a diagnosis is usually done through solving a satisfiability problem and through the use of nonmonotonic reasoning. This thesis adopts a practitioners view in that it is assumed that some diagnostic algorithm is provided with the diagnostic system $(\Phi, COMPS, OBS)$ as its input, and outputs the diagnosis. How the computation of the diagnosis is done, is subject to the applied diagnosis algorithm. For this thesis the Hitting Set Tree (HS-tree) algorithm of Reiter [17] in the modern form of Rodler [34] can be assumed. A brief overview of different diagnosis algorithms is given in the next section.

In many cases the size of individual diagnoses can be quite large and contain sometimes hundreds of components. Confronting an operator with such a large set of possible components to be checked is infeasible. Instead, the size of a diagnosis needs to be limited. A minimum cardinality diagnosis is a diagnosis that contains the smallest possible number of components.

Definition 40 (Minimal Cardinality Diagnosis). *A Minimal Cardinality Diagnosis is a diagnosis ω' , so that $|\omega'| \leq |\omega|$*

Please note that below, $COMPS$ is sometimes substituted for F . In those cases F always denotes a subset of $COMPS$, such that $F \subset COMPS$. Further, in contrast to common literature, which denotes with $COMPS$ the name of the components (the propositional logic symbols) and their health states (the variable assignment to the propositional logic symbols), we will distinguish F to mean the health state and \tilde{F} to mean the names of the components.

1.7. Consistency-based diagnosis algorithms

Several good consistency-based diagnosis algorithms exist. The goal of this thesis is to provide a methodological basis such that these algorithms can be used with cyber-physical systems, in particular industrial production systems. The algorithms are described only abstractly to give the reader a basic understanding. The goal is to make the methodology of this thesis independent of any concrete diagnosis algorithm.

1.7.1. Reiter's Hitting Set algorithm

Some of the earliest formal definitions in consistency-based diagnosis come from Reiter [17] when he introduced his HS-tree algorithm in 1987. He starts out with a tuple (SD, COMPS) modelling the system and its components. Given another set OBS describing current observations within a system they analyse the tuple (SD, COMPS, OBS) and calculate conflicts, when $SD \wedge OBS \wedge \{\neg AB(c_0), \dots, \neg AB(c_k)\}$, with $k \in \mathbb{N}$ is inconsistent. The AB predicates stem from McCarthy [102] and describe whether some component behaves abnormally. From these inconsistencies a tree is generated bottom-up: The leaf nodes contain sets of single components. Thus, the cardinality for each leaf node is one. The parent nodes of the leaf nodes contain sets of cardinality two. The tree is filled until the root node contains one set of all components. Finding the minimum cardinality diagnosis is done by traversing the tree bottom up, left to right and checking which hitting sets can be calculated.

1.7.2. General Diagnosis Engine - GDE

The GDE diagnosis algorithm was introduced in 1987 by De Kleer [19] for the diagnosis of binary circuits. It uses the diagnostic system three-tuple (SD, COMPS, OBS) to check consistency of a logical knowledge base. It requires at least a weak-fault model predicting the normal working behaviour. Inconsistencies arise, when the model's prediction differs from the system's observations. GDE identifies those inconsistencies and calculates those components within COMPS that could, according to the system description SD, be a cause for the inconsistency. Concretely, GDE starts with the assumption that SD is correct. When an inconsistency arises, it retracts *assumptions* that the system model is correct. The set of those retracted assumptions is called the set of fault hypotheses. However, a fault hypothesis can become arbitrarily large in complex systems. Therefore, GDE discriminates fault hypotheses through a hitting set calculation to find the minimum cardinality diagnosis.

1.7.3. Conflict-directed A* - CDA*

A physical system model SD is often formulated as a graph. It is therefore intuitive of interpreting diagnosis as a search problem within a graph. For this task the CDA* algorithm [103] was developed. CDA* is an adaptation of the well-known heuristic A* algorithm [104]. A* searches graphs using a target function, which calculates the path costs from the start node to each visited node. The target function is designed such that it never overestimates costs. Thus, A* can always select the path with the lowest cost to reach a goal.

CDA* works by interpreting the underlying system graph as a collection of ok-assumptions. Searching is done by retracting ok-assumption to find a diagnosis. CDA* starts by assuming that all components within a system are healthy $\forall c_i \in \text{COMPS} : c_i \rightarrow \top$. If a fault occurs, some of those assumptions cannot be true anymore. In binary circuits (and many other systems) components may mask faults. Finding system parts that may mask faults is done through the calculation of cones. CDA* identifies cones within a system graph and creates fault hypotheses for each path containing a cone with a possible fault. The components within each path with a possible fault make up one hypothesis. A second graph is created

from the set of all hypotheses, where the graph root is the empty set. The first child-nodes after the empty set are the system components belonging to the first hypothesis. In the second graph level, the system components of the second hypothesis are written and so on. The A* algorithm is then used to find that path through the hypothesis graph with the lowest cost.

1.7.4. SATbD

A SAT-based algorithm was introduced by Metodi et al. [100]. The idea is again based on the calculation of cones within a system graph. The algorithm calculates which components influence which system outputs. So called "sections" are calculated which describe components that influence the same system output. For SATbD the authors determined that only the last component of a cone (the *dominating* component) can be diagnosed, as all other components within a cone are masked. SATbD uses the tuple (SD, COMPS, OBS) and a predicate logic formulation of binary circuits. Added to the predicate logic formulation is the expression $sum_leq(\neg H_{c_0}, \neg H_{c_1}, \dots, k)$, where $H_{c_i} \in \text{COMPS}$ denote the health of a component and sum_leq is the sum of faulty components up to a depth of k . Changing k leads to an iterative fault diagnosis process. k is often determined through the assumption that the minimum cardinality diagnosis must be smaller than the number of outputs per section.

1.7.5. SAFARI

SAFARI (Stochastic Fault Diagnosis Algorithm) [22] is a greedy stochastic hill climbing algorithm for approximating the set of minimal diagnoses in a diagnostic system. Its input consists of a diagnostic system description given as a fault-augmented propositional logic formula and an observation. SAFARI starts with a random diagnosis. Taking the random diagnosis as a baseline, SAFARI iteratively improves the diagnosis by randomly "flipping" fault literals in the propositional logic formula. This is a one-way process so that literals that were flipped from faulty to healthy will not be flipped back. Each outcome is consistency checked, which results in diagnoses with strongly monotonically decreasing cardinality. This process is restarted N times to start from N different random initial diagnoses. The number of "flips" and the number of restarts N can be adjusted to optimally cover the search space to find minimal cardinality diagnoses.

1.8. Interfacing consistency-based diagnosis algorithms

Regardless of the actual implementation of the diagnosis algorithm, a formal interface is required which describes the type of information exchanged from and to such an algorithm. Figure 7 depicts an abstract representation of a diagnosis algorithm. The goal for the methodology in this thesis is to conform to this interface, such that many diagnosis algorithms can be used. All algorithms take the three-tuple (SD, COMPS, OBS) as their input (see Definition 38). But the implementation of the different symbols may differ. While CDA*, for example, has a system description through graphs, GDE and Reiter's HS-tree

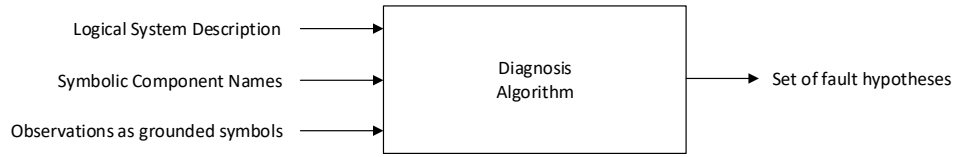


Figure 7.: Interfaces to diagnosis algorithms

use logical formulations. However, it is possible to convert graphs into logic expressions and vice versa [105].

1.9. Creating System Descriptions from Data

Not all system descriptions for diagnosing cyber-physical systems can be created by experts. Some of the reasons are missing literacy in propositional, predicate or SMT logic, high labour costs, unavailable experts, or possible human errors due to model complexity. It is therefore highly desirable to learn dependencies, or even causalities, from data such that the workload of experts is reduced. Two main approaches for such automated model creation shall be introduced here: i) correlation-based, ii) Granger-causality based. The advantage of correlation-based methods is that they can be used with little expert knowledge and their results are easily interpretable. But the types of association found may not be as accurate. Dependencies found using Granger-causality are more robust, but usually interpreting the results involves more expert knowledge. Chapters 3 and 4 in Part III will introduce algorithms to learn propositional logic system descriptions from data.

1.9.1. Correlation-based Approaches

A mostly uninformed approach to learn dependencies from data is to compute correlations between signals of a time-series.

Given input signals $S \in \mathbb{R}^n$ and output signals $M \in \mathbb{R}^m$ from a black-box process, for each output variable $m_j \in M$, a correlation analysis using the numerical values of the input data is performed. For each signal $s \in S$, the input signals having the largest absolute correlation with m_j are identified. We use Spearman correlation to describe the non-linear and distribution independent relationships between variables. Spearman correlation is defined as

$$\rho = \frac{\text{cov}(rX, rY)}{\sigma(rX) \cdot \sigma(rY)} \quad (1.3)$$

where rX and rY are the ranks of two signals, σ is the standard deviation, and cov is the covariance. Hallin and Peri [106] have shown that using rank-based methods work well for time-series data.

To integrate a system description learned through correlations, it is in some cases possible to relax the strong definitions of the connection model *CON* (Def.: 23). *CON* itself depends on prior knowledge of the connectives, which is hard to achieve without any external,

domain specific knowledge. One goal in this thesis is, therefore to learn a propositional logic model from data that does not rely on a manually specified component model.

1.9.2. Granger Causality

A stronger association between components in cyber-physical systems can be achieved using Granger Causality [107, 108]. With Granger Causality it is possible to compute whether one signal has a significant effect on some other signal over time. Granger causality is defined as follows: Given two variables $X \in \mathbb{R}$ and $Y \in \mathbb{R}$, predicting future values of X is compared to predicting future values of X and Y . If the prediction is better, when predicted through X and Y , it is said that Y Granger causes X . Formally, this is for two variables X and Y done through the two vector-auto-regressive models [109]

$$X_t = \sum_{k=1}^p \mathbf{A}_k \mathbf{X}_{t-k} + \epsilon \quad (1.4)$$

and

$$X_t = \sum_{k=1}^p \mathbf{A}'_k \mathbf{X}_{t-k} + \sum_{k=1}^p \mathbf{A}_k \mathbf{Y}_{t-k} + \epsilon' \quad (1.5)$$

with time $t \in \mathbb{N}$, lag parameter $k \in \mathbb{N}$, index $p \in \mathbb{N}$, $\mathbf{A}_k, \mathbf{A}'_k$ being a matrix of coefficients of the vector-autoregressive-model, \mathbf{X}_{t-k} being past values of $X \in \mathbb{R}$, \mathbf{Y}_{t-k} being past values of $Y \in \mathbb{R}$ and noise $\epsilon, \epsilon' \in \mathbb{R}$. If the prediction of X and Y (Eq.: 1.5) is better than the prediction using only values of X (Eq.: 1.4) X Granger-causes Y .

In practical terms this means that when some time-series of a temperature can be extrapolated better using the temperature and pressure values, compared to the temperature values alone, then the pressure granger causes the temperature. It must be noted that Granger causality cannot deal with spurious correlations and confounding variables, which makes it inferior to Pearl's definitions (Chapter 5, Part II). However, it can be learned from data and has a lower computational cost, which makes it suitable for the use with cyber-physical systems. In this thesis, Granger Causality is used to create propositional logic system descriptions from dependencies between an expert-defined label (component health state) and process data. The result are propositional logic expressions that associate a process signal to components influencing the process signal.

2. State of the Art

This chapter will first highlight the history and current research in fault diagnosis and its applications. It presents fault diagnosis from the artificial intelligence and from the control theoretic side. This is followed by a survey on causality research in applied contexts. Causality underlies many of the model-generation methods within this thesis and a body of research is available. But many sources are outside the field of cyber-physical systems. It is therefore important to provide an overview how causality research can be utilized for diagnosing cyber-physical systems. Afterwards, specializations are introduced, in particular with focus on SMT and industrial applications. While some research gaps are identified when listing different sources throughout the chapter, a summary is provided at the end which provides an overview of how this thesis fills in identified research gaps.

2.1. Consistency-based Diagnosis

Consistency-based diagnosis is based on the theory of non-monotonic reasoning. According to Minker [90], the research field of non-monotonic reasoning goes back to the 1959 paper by McCarthy [110] in which he states that a program needs to select the premises on which it bases its conclusion, depending on the situation. A comprehensive theory of non-monotonic reasoning was presented by Reiter [111]. Following this idea, truth maintenance systems were developed, which later were used as the foundations for consistency-based diagnosis algorithms. Main proponents of these systems were Doyle [112] and De Kleer [113]. Recently, Bochman has published a book on non-monotonic reasoning within a logical theory of causality [114]. Nowadays, non-monotonic reasoning is often realised in the form of answer-set programming. Souchio et al., for example, have presented a method to evaluate expert's degree of belief into answer-set programming to find faults [115].

The research field of consistency-based diagnosis can be traced back to the 1974 paper by Brown [43]. In his proposal, Brown was the first to describe the theoretical modelling, analysis, and diagnosis of electronic components. In his article he presents his idea of diagnosis either on single, atomic components of a single-tube AM radio receiver, or on composite components such as power supply, amplifier, and oscillator in a communications receiver converter. Brown credits Sussman [18] for the idea of implementing an electronics repairman. This piece of software was supposed to be able to take a schematic of a diagnosis device and a "suitable description of the device's public extrinsic and intrinsic properties". These should be detailed enough to let the repairman autonomously produce a plan of the device. Brown claims that the electronic repairman can use this information to create a parse-tree of the device. The electronic repairman was supposed to exhaustively search a tree of components and measure, if a component's described behaviour matched the behaviour measured by the repairman. Discrepancies between the prediction and the measurements are symptoms of a fault. The schematic of the failing device can be

recognized as the set of components (COMPS) proposed by De Kleer and Reiter. Further, the device's "public extrinsic and intrinsic properties" are similar to Reiter's system description (SD), while Brown's measurements are Reiter's observations (OBS). In 1976, de Kleer [116] presented the first ideas of modern-day consistency-based diagnosis. In his paper he showed how an electrical circuit can be described by logic descriptions, though he limits his study to direct-current (DC) circuits. He investigates how values can be propagated through a qualitative logical system description and how these propagated values can be used to diagnose faults in a circuit. He created his system description by modelling DC circuits according to Kirchoff's Law [117] and the usual physical properties of the atomic components such as resistors, capacitors, diodes, and inductances. With Kirchoff's Law he modeled the connections between components. These laws are an important step in propagating values throughout the circuit as these enable a diagnosis program to make inferences about unknown values.

The foundation for diagnosing binary circuits with consistency-based diagnosis was laid by Reiter [17], and De Kleer and Williams [19] in 1987. Reiter introduced the formalisation mechanisms that are still used by most authors in the field of consistency-based diagnosis today. Reiter leveraged the notion of AB-literals from McCarty [102] to model the possibility that single components in a circuit may fail. Though McCarthy's paper primarily dealt with modelling common-sense in artificial intelligence, he introduced AB literals to be able to model an expression such as "Not all birds can fly". He wanted to avoid the pitfall having to describe a bird that cannot fly as an abnormal bird. Instead, his goal was to describe a normal bird having the abnormal property that it cannot fly. Based on the description of failing components through the AB-literals, it became possible to attempt diagnosing a circuit. Reiter showed the formal computation of conflict and hitting sets [116] as well as the spanning up of the hitting set tree (HS-tree) to generate diagnosis candidates. He was also one of the first to show an example of incorporating satisfiability modulo theory with linear arithmetic into predicate logic statements, though he did not call it SMT. But similar approaches can also be found in an article by Finin et al. [92]. Compared to the paper of de Kleer and Williams, however, Reiter did not consider the inference step to generate new possible measurements by lookahead in a circuit.

After de Kleer, Williams, and Reiter had laid the foundations of consistency-based diagnosis several different research directions followed. De Kleer continued to develop and improve his approach in diagnosing Boolean circuits. For this, he extended his original theory with behavioural modes [118]. These enable the diagnostician to specify in which way a component fails (strong-fault models). He mentions that in the previous approach a resistor was as likely to change its resistance as it was to suddenly become a current source when it fails. Obviously, the first failure mode is much more likely than the latter. With behavioural modes it is exactly specified which failures a component can assume, including the unknown failure. This makes logical reasoning easier, as the result of a reasoning step specifies exactly what failure mode occurred. First, de Kleer described his approaches on single faults. In reality, however, multiple components may fail at the same time in a technical system. Therefore, he extended his theory on how to diagnose multiple faults in Boolean circuits [19]. This is done by combining the information gained by one, or optimally, multiple measurements and then finding the smallest common subset of components that explain the observed behaviour. Later, de Kleer extended this theory to first diagnosing intermittent faults [119] and then to multiple persistent and intermittent faults [120]. Intermittent faults occur, for example, when two wires are shortened or through stochastic physical processes. Poole [121] has provided a discussion, where he contrasts consistency-

based diagnosis with abductive diagnosis. His finding is that abductive diagnosis reasons (in our terms) with the flow of causality, such that the diagnoses entail the observations. Consistency-based diagnosis is formulated such that the observations entail the diagnoses. Research into Boolean circuit diagnosis is still ongoing [122, 123], although many articles nowadays contain hybrid or highly optimised approaches for single use-cases.

By now, many consistency-based diagnosis algorithms exist [124, 103, 22, 34, 23]. In the following, some major research directions shall be highlighted. This thesis formulates a theory which explains how to use these diagnosis algorithms in the context of cyber-physical systems and through the use of residual values. In this regard, Stern et al. [124] have studied shortcuts to calculating diagnoses. Ignatiev et al. [125, 126] and Grastien [64] have presented diagnosis algorithms based on satisfiability modulo theory that are also used in this approach. Matei et al. have published articles [127, 128] about diagnosing cyber-physical systems using differential equations. Cazes and Kalech [129] and Kalech et al. [13] address the problem of uncertain observations, which is an ongoing problem in cyber-physical systems. Similar to Ignatiev and Grastien, they also base their algorithm on satisfiability theory. Cyber-physical system diagnosis has recently been addressed by Muskardin et al. [130] and in the surveys by Dowdeswell et al. [131] and Yucesan et al. [132]. In many approaches, residuals are interpreted binary, however, Koscielny et al. [57] have shown that interpreting ternary residuals can bring some significant advantages. Wang et al. [133] have published an article where they semi-automatically created diagnosis models based on ontologies for car production line diagnosis. The work is in parts similar to this thesis in that they also present a method to create models. The method in this thesis, however, is more causality and residual oriented and gives a deeper analysis of fault diagnosis theory. This thesis' approach has also similarities to the BRIDGE approach introduced by Trave-Massuyes et al. [61], as it also presents a method to fuse the research fields of the control theoretic Fault Detection and Isolation approach (FDI) and the artificial intelligence research-based fault diagnosis approach (DX).

2.2. Fault Detection and Isolation

Within fault detection and isolation research, cyber-physical systems are usually referred to as hybrid systems. These systems are characterised through operating in different modes, where each mode is a unique configuration of the discrete input values to the system. Within each mode equations govern the behaviour of continuous variables. The system progresses from one mode to the next through the use of mode transitions, which act as guard conditions. Khorasgani et al. have defined hybrid systems more formally [42]. Hybrid systems are different to the Boolean circuits that are analysed in traditional consistency-based diagnosis. Instead, they comprise several complications such as continuous values, time dependency, and require more complex models. Most systems that one finds in the real world and especially in technical systems are hybrid.

We will now first introduce some standard fault detection and isolation schemes and then show how these are applied in current research directions.

Provan [134] has developed an approach that uses a composite automaton to model the continuous aspects of a hybrid system. To create an automaton, he abstracts the continuous signals into qualitative states by translating them into a finite set of polynomials. In a second step it becomes possible to compute transitions between these qualitative states.

Subsequently the automaton is transformed into an equation set which can then be translated into propositional logic. The propositional logic description can then be used with a standard consistency-based diagnosis algorithm to find faults. Klenk et al. [135] have shown how to augment models of cyber-physical systems with manual fault annotations. Butt et al. [136] explain how faults in fault models can be distinguished into multiplicative (component) faults and additive (sensor) faults.

Struss [137] published a paper on the fundamentals of consistency-based diagnosis of dynamic systems. In this, he described how hybrid systems can be modelled without resorting to a complete simulation of the system under investigation. He proposed to capture the temporal and dynamic behaviour of a hybrid system in a set of modes to model the system. Each mode has distinct state, temporal and Continuity, Integration, and Derivatives (CID) constraints that affect all modes. For one mode, all variables have a domain capturing the permissible values. Diagnosis is performed by checking whether the set of constraints together with the observations from sensors is consistent. He demonstrates his approach on a car's anti-braking system and claims to find all the faults which usually occur. When dealing with hybrid systems there always exists the problem of discretisation. Provan used a composite automaton for this. Struss divides his system into modes by discretizing the underlying sensor values (this can be translated into an automaton [37]). Lin [138] already showed in 1994 that online and offline diagnosis for discrete event systems (DES) can be realised by using simple Moore and Mealy automata.

Daigle et al. [139, 140] have adapted a discrete event approach to diagnose continuous systems. They claim that each fault that occurs in a continuous system has a unique fault signature. They also claim that there exists a measurement ordering that describes which sequence measurements deviate until a fault occurs. To capture fault ordering, they manually construct a temporal causal graph. Under the assumption that all fault signatures and measurement orderings are known, they employ a diagnoser that traces the states through the temporal causal graph based on measurements. The output of the diagnoser is a fault trace. A second diagnosis algorithm takes this fault trace and determines which components must be faulty to explain this trace. This second diagnosis step is similar to the diagnosis lattice introduced by Reiter.

Grastien et al. [141] have developed an approach to extend Reiter's diagnosis algorithms which was described for binary circuits to include DES and hybrid systems. Their approach is similar to Daigle et al., Struss, and Provan in so far as they transform the continuous parts of a model into qualitative states. Following this, their preferred-first algorithm goes through Reiter's diagnosis lattice and computes valid hypotheses with the goal of finding a minimum cardinality diagnosis. An improvement compared to previous work is that they implement their hypotheses tests with a SAT solver. By using the solver, they were able to solve many more problem instances than a standard solver for DES systems. Rintanen and Grastien [142] have shown how to translate diagnosability into a satisfiability problem to check whether a system has exhibited failure behaviour. They do this on transition systems called automata. In such a system, it can be proved if an arbitrary number of events ever leads to a state of failure. Their approach is to translate this test into an SAT problem and thus show significant improvements over the state-of-the art.

An active research field between causality research and fault diagnosis are bond-graph approaches mainly pursued by a research group around Gautam Biswas [143, 25]. Gao et al. [68] provide a good overview over bond-graphs and other consistency-based diagnosis methods. Narasimhan and Biswas [41] have proposed an FDI system for diagnosing the

fuel-transfer system for fighter aircraft. In their approach, they model the fuel-transfer system with hybrid bond-graphs. In another work, Khorasgani and Biswas [42] describe a hybrid system model through hybrid minimal structurally overdetermined sets. In their work, hybrid systems are defined as systems containing discrete and continuous values, while minimal structurally overdetermined sets are sets of equations that contain exactly one more equation than free variables. Ding has also published some seminal work about residual-based fault identification techniques [144]. Borutzky [145] has introduced a data-driven method for failure prognostics based on bond-graphs. Niu et al. [146] and Daigle [139] have both presented approaches for diagnosis using hybrid bond-graphs. For minimally structurally overdetermined sets, Khorasgani [12] has presented a method using the Dulmage-Mendelson Decomposition. For analytical redundancy relations, Lunze [147] has presented a recent fault diagnosis approach. Another residual-based approach, particularly for cyber-physical systems, was presented by Khan et al. [148]. Zhong et al. [149] have published a residual-based approach in an event triggered setting. Here, residuals are only evaluated once certain events occur and use an unknown-input observer approach. The approach relies heavily on expert knowledge and omits symbolic processing. The idea of event triggering, however, could be interpreted as a sampling interval for the approach in this thesis. This was further explored by Wu et al. [150] in their approach of an external diagnostic observer.

Roychoudhury et al. [151] have shown how to use hybrid bond-graphs (HBG) to diagnose hybrid systems. HBGs abstractly model the system by describing causal, continuous relationships between components. In Daigle et al. [139] they have shown how to employ the developed HBGs to diagnose a spacecraft power distribution system. Prakash et al. [152] have used an extended framework with HBGs to make improvements in diagnosing two-tank systems. Jung and Sundström have used analytical redundancy relations (ARRs) and combined these with machine learning for diagnosing hybrid systems [88]. The theory of redundancy relations was summarized by Gertler [153], but can be traced back to Mach et al. [154] and Potter et al. [155]. A decentralized approach for the formulation of ARRs was introduced by Perez-Zuniga et al. [156]. Verdieri et al. [157] have defined ARRs for fault diagnosis from a mathematical perspective.

Some recent approaches have come up describing hybrid fault diagnosis techniques, which fuse the traditional structural equation-based approach with data-driven techniques. The idea behind these approaches is similar to this thesis, where shortcomings in analytical (expert knowledge-driven) approaches are overcome by integrating knowledge generated automatically from data. Oromi et al. [158] have presented an approach which learns residual values from data and fuses them with results from a structural analysis. Mohammadi et al. [159] have presented a data-driven fault diagnosis method, where residuals are learned as a grey-box neural network model. Both of these articles point into the direction where Yuan et al. [98] have shown how to learn differential equations from data. Bayouhd et al. [160] have presented a method to execute hybrid system diagnosis using a coupling of discrete and continuous event techniques, based on the calculation of ARRs.

2.3. Causality And Dependencies

Causality is an integral part of accurately describing a cyber-physical system. While theoretical causal research originates from the social and medical sciences [54], some

authors have integrated causal concepts into their research works in technical fields many decades ago [27]. Some of the drawbacks of causality research will be discussed in Chapter 5 Part III.

In the 1980s the seminal works about Qualitative Process Theory from Forbus [27] and De Kleer [32, 101] led to significant advances in describing quantitative physical processes through qualitative expressions. De Kleer takes an approach through his idea of *envisioning* [32] that focuses more on diagnosis use-cases than the more abstractly described approach of Forbus. The most important link to consistency-based diagnosis is through De Kleer's introduction of the no-function-in-structure principle and his class-wide assumptions. Using these, he interprets each component within a system as an information processor and defines a methodology for the propagation of effects through a system by using flow-like and effort-like forces. Later works by Williams [161] and Raiman [162] extended Qualitative Process Theory by highlighting the treatment of continuity in the case of Williams, and discretisation, in the case of Raiman. Common misconceptions about qualitative physics were answered by Williams and De Kleer in 1991 [163], which concludes that qualitative physics is a suitable methodology for analyzing many kinds of real physical processes. The works by Mosterman, Biswas et al., and others [25, 42, 41] carry on the idea of flow-like and effort-like propagation through the introduction and qualification of hybrid bond graphs. Matei et al. [127] have attempted to bridge the gap between qualitative and quantitative approaches by combining parameter tracking with analytical redundancy relations.

Recently, Nielsen et al. [164] have described a causality detection approach for Multilevel Flow Modelling, although, compared to our approach, they do not deal with fault diagnosis. Jaber et al. [165] have presented an approach to improve causal reasoning and investigated the use of partial ancestral graphs. In the same area, Schölkopf et al. [166] have described how to use causal reasoning with semi-supervised learning. They propose a method to automatically obtain causal variables, which many approaches in the literature assume to be given. This deep-learning based approach, however, is infeasible for diagnosis methods like the present one when no faulty data is available. The same can be said for the approach presented by Martinez-Garcia [167], who also presented a diagnosis approach based on non-faulty data. Chao et al. [168] have tried to mitigate the problem of little available data by presenting a framework which uses either physics-based models grounded on first principles, or a convolutional deep neural network approach. The present work neither relies on first principle models only, nor requires black-box approaches such as neural networks. A practitioner may choose any residual generating function that suits the use case. The approach is able to integrate generated residuals and then perform symbolic fault diagnosis. Deep learning diagnosis approaches have also been proposed by Fink et al. [169], especially with use cases in graph interpretation, natural language processing, and transfer learning. However, they do not present actual diagnosis methods. Wu et al. [170] have presented an automated method to learn causal dependencies in clinical data using neural networks. Li et al. [171] have presented an approach to perform Granger Causality on partial time intervals to improve accuracy. Using this approach is useful for heavily varying time-series, where piece-wise regularities can be identified. This piece-wise regular behaviour is not relevant for the use-cases in this thesis. But can be seen as a useful extension. Laurent et al. [172] have presented a counterfactual resimulation method for rule-based models. Their method allows the analysis of causal effects in rule-based models, since the calculation of counterfactuals according to Pearl [173] is not applicable to these kinds of models. Kühnert et al. [174] have presented a somewhat similar method to the one in this thesis, where they learn causal structures from data using Granger Causality. But in

contrast to this thesis, their goal was to learn the causal structure and not the propositional logic model of the system under investigation. Duan et al. [175] have present an approach for direct transfer entropy which can also detect nonlinear dependencies. Chikihara et al. [176] have presented a multivariate approach for Granger Causality analysis, where they use a supervised classifier to distinguish the direction of causality between two variables and determine whether causality exists at all. Recently, Moraffah et al. [177] have provided a good overview over methods for treatment effect estimation and causal discovery in time-series. Grünbaum et al. [178] have presented a causal discovery method from observations. Kolb et al. [87] and Kumar et al. [179] have presented methods to learn SMT expressions and MaxSAT problems from data. But their goal was not the application to diagnosis. Tank et al. [180, 181] have described approaches to apply Granger Causality to time-series data. Similarly, Lusch et al. [109] have applied Granger Causality to infer network structure. Yuan et al. [182] have shown how to use Granger Causality with principal component analysis to establish root causes based on features. Nadim et al. [183] have provided a comprehensive overview over causality learning techniques and present their own method based on process mining and explainable machine learning. But their method, while quite general, is limited to discrete event systems. In applied causality research Singh et al. [184] have presented an approach to detect causes of groundwater depletion in the US using Granger Causality. Sugihara et al. [185] have presented an approach for causality detection in complex ecosystems. They claim that their approach can detect causality, where Granger Causality tests fail. For this, they developed their method convergent cross mapping using manifold calculation.

The approaches within this thesis use the idea of structural equations and understand causality as introduced by Pearl [54, 186]. However, causal relationships are not identified through creating a causal graph and then applying the do-operator like Pearl suggested. Instead, minimal structural patterns are identified, which allow a practitioner to divide a system into small fragments and use them for fault diagnosis. For example, the quantization procedures are used to interpret the flow of time through small intervals [187]. In the present approach, however, a method is provided which is easier to use for practitioners (i.e. less reliant on elaborate formalisms) and gives a formal justification why using residual calculation is superior to quantifying pure sensor data. Using the approach, practitioners can divide the system under investigation into minimal structure patterns, which are easier to analyse. Further, the description of a system (the model) is created through the formal definitions of a component and a connection. The information necessary, such as input and outputs to a component, is already a part of the construction documentation in most real-world production systems. Further, the system modelling and diagnosis can be directly linked to digital twins [60].

2.4. Satisfiability Modulo Theory

Several authors have published research of using SMT in the context of fault diagnosis. Since research on SMT entails the applied-research side, as well as the theoretical research side, this section gives a brief overview of how SMT is applied to solve diagnosis problems.

SMT is used to include several theories such as linear arithmetic, difference logic, and bit vectors into a propositional or predicate logic framework. In this case, linear arithmetic can be with Booleans, integers, or real numbers. Ernits and Dearden [188] were one of

the first authors to propose using SMT logic for diagnosis with their approach called diagnosis modulo theories (DMT). They evaluated their system on the ADAPT-LITE system [189] simulating spacecraft power distribution. They claim that while their approach is computationally more expensive in each time step, it has significant advantages when the system is faulty and an inconsistency occurs. Previous approaches, for example for spacecraft, often had a computationally infeasible complexity when faults occurred.

Grastien [39, 64] used SMT for the diagnosis of hybrid systems. He is also one of the first people to use SMT in the sense proposed in this thesis. In his work, he discretises values in a hybrid system into a set of distinct states. Each observation $\langle \tau, A \rangle$ is understood as a behaviour A at time τ , where A is a partial assignment of the variables in a state. Measurement errors are included by including constraints such as $v_{min} \leq v_{current} \leq v_{max}$, which states that the observed voltage must be between two tolerance thresholds v_{min} and v_{max} . Each variable is augmented with an indicator stating at which time-step the variable expression is valid. The conjunction of these statements about the system create the system description. Observations are included in a similar manner. The statements are solved using an appropriate SMT solver. In this way, the usual procedure of consistency-based diagnosis can be employed. Inconsistencies between the system description and the observations lead to inconsistencies. From these, minimal hitting sets are computed. Subsequently, diagnoses according to Reiter can be created.

Eggers et al. [190] have presented a method to model ordinary differential equations in SMT for hybrid systems. Based on the example of a room with an indoor stove, they describe the SMT syntax for modelling the initial conditions, state transitions, and the target condition. The advantage of their approach is that it replaces the need of explicitly encoding ordinary differential equations through (in-)equations when modelling hybrid systems. Provan [191] has shown how over- and underapproximations of ordinary differential equations and differential algebraic equations (DAEs) can lead to sufficient threshold-based diagnoses. Fraenzle et al. [192] have augmented SMT with stochastic methods in order to analyse stochastic hybrid systems. By using bounded-model checking together with probabilistic hybrid automata, piece-wise deterministic Markov processes, and stochastic differential equations they are able to create an analysis system without the need to formulate intermediate finite-state abstractions as the methods mentioned above do.

An approach in some sense similar to SMT is the differential dynamic logic of André Platzer [193, 194]. He uses predicate logic integrated with differential equations to prove the correctness and safety of cyber-physical systems. As with SMT logic, this allows him to reason with real numbers. The difference to this thesis is that his approach would need to be adapted to the propositional logic expressions for consistency-based diagnosis as was shown in Chapter 1, which requires knowledge of the differential equations of a cyber-physical system. These equations, however, are often unknown.

2.5. Industrial Approaches

In industrial applications many practical fault diagnosis methods were developed. They are sometimes based on a formal theory such as diagnosis from first principles according to Reiter [17] and his hitting set algorithm. Often, however, diagnosis approaches are adapted to use-cases or new approaches to fault diagnosis are developed. Two of these methods are spectrum-based fault localization and case-based reasoning. These approaches and

current research directions are introduced first. Then a choice of other practical and applied diagnosis approaches are presented. The sources in this section are intended to provide an overview and give the reader some context and perspective, not to be exhaustive.

Spectrum-based fault localization [195, 196] is usually employed in diagnosing and verifying software components. When executing a sufficiently complex program code there exist many possibilities in which the code can run. External conditions trigger which of these possibilities is executed. The different possibilities of executing program code can be represented through a graph. Verifying the program code means that a certain percentage of all possible paths through the graph needs to be tested [197]. To localise where a fault occurred during a test, the program code is divided into single components, such as a for-statement, a while loop etc. By executing many of the program's possible paths and having multiple paths with intersecting components, fault isolation becomes possible. Abreu et al. [196] extended this approach to multiple faults and by taking component failure probabilities into account. Pill et al. [198] have adapted spectrum-based fault localization to logic programs, thus demonstrating that fault localization is compatible to verification and diagnosis of logic circuits.

Case-based reasoning (CBR) is a more intuitive approach than model-based diagnosis or spectrum-based fault localization [199, 200]. The main idea is that the CBR diagnosis system gains knowledge as it runs. If a new fault occurs that is not within the diagnostic system's knowledge-base, a new case is created. Such a case is an abstract representation of the problem instance. The representation can be plain text, some (semi-) formalized knowledge, or the output of a question-answer system. Once the case has been filled out, a solution to the problem is proposed. The set of possible solutions often comes from expert knowledge as well. For example, for every new case an expert would enter the description of the case and then enter a possible solution. The case and its solution are saved into a memory within the diagnostic system. If the system encounters a new instance of this fault in the future, it can retrieve the solution from memory. Therefore, each problem only needs to be encountered once and curated by an expert. Afterwards, the system can create diagnoses on its own. CBR has been successfully applied in industrial use cases [201, 16]. Many other industrial approaches to industrial fault diagnosis have been presented in the last years. Struss et al. have published a diagnosis method for automobiles [202] and for process industrial use-cases [14]. Biswas et al. have presented a method to diagnose spacecraft [203]. Garramiola et al. [204, 205] have provided an overview over many fault diagnosis methods for railway traction drives. Nguyen et al. [206] have adapted a Bayesian fault diagnosis approach to nuclear engineering systems. Jung [207] has published a methodology for large-scale systems using structural analysis. Frisk et al. [208] have presented an approach selecting optimal residuals for diagnosing a car engine. Youssef et al. [209, 210] have presented a diagnosis method for robotic platforms in the case of unmanned ground vehicles. Wang et al. [211] have applied a consistency-based diagnosis algorithm to a power distribution network. Lyu et al. [212] have presented a neural-network based method where a residual building, soft threshold, global context -based method (RSG-method) is introduced for diagnosing motors in high-noise industrial environments. Mousavi et al. [213] have presented an ARR-based approach using the formulation of fuzzy rules. But as with most other residual-based approaches, they rely heavily on pre-existing expert knowledge.

2.6. Major Related Work

This section provides an overview of the closest research related to this thesis. With regard to performing diagnosis using SMT logic expressions, the work of Grastien et al. [64] is in some regards similar. The major difference is that they did not include the generation of residuals in their work and focused on the diagnosis of discrete event systems. Thus, they also did not formulate their expressions in such a way that they would be used with many consistency-based diagnosis algorithms. Another close work that mentions diagnosing systems using residuals together with machine learning techniques was presented by Matei et al. [127]. In contrast to this thesis, they focused on learning physical models (such as differential equations) using a hybrid expert-driven and machine learning-based approach. They optimised their model such that it can be used for real-time applications in rail-switch diagnosis. The work in this thesis provides a theoretical foundation that justifies why the use of residuals is advantageous. As such it bolsters the foundations of approaches such as Matei's [127].

Additionally, this thesis' main point is a formulation of SMT logic, which can incorporate the outputs of residual generation algorithms and machine learning algorithms. Because of the general formulation of the diagnosis model in this thesis, the content is also close to the BRIDGE approach introduced by Trave-Massuyes [214, 61]. The main difference here is that this thesis is not supposed to be a theoretical analysis and matching between different research fields, but instead is a practical approach to diagnose cyber-physical systems. The BRIDGE approach can be interpreted as a *horizontal* comparison between definitions in artificial intelligence and control theory. This thesis makes a *vertical* matching: It provides a theoretical basis of causality and a formalisation for diagnosis models using a unified logical framework, together with an integration of residuals.

Another quite close line of research was done by the research group around Daniel Jung. They published an approach to combine consistency-based diagnosis with data-driven anomaly classifiers [215, 216]. Compared to this thesis, their aim was towards a novel technique for residual generation using one-class support vector machines. The output of these machines was then diagnosed. In another work, Lundgren et al. [217] presents another data-driven approach for finding faults compatible with consistency-based diagnosis. In still other work, Jung [218] has used a grey-box approach with neural networks and structural analysis to perform fault diagnosis. Still, none of these approaches provide a common logical basis to combine the multitude of residual generation methods with the set of generally applicable diagnosis algorithms that was introduced in Section 1.7 Part II.

2.7. Summary and Research Gaps

In this chapter we have identified several relevant research directions. These are in short: a) General research toward causality [54], b) Causality in cyber-physical systems (De Kleer [32], Fink [169]), c) Consistency-based diagnosis [20], d) Bond-graphs [140] and Structural equations [148], e) Satisfiability Modulo Theory [39].

The proposed methodology for diagnosing cyber-physical systems touches each of these research fields: This thesis introduces an extension to causality research in cyber-physical

systems. The approach uses ideas of causality by Pearl and some of the notions described by Forbus and De Kleer. However, Pearl's formulation of causality is formal and expert-driven. He assumes that a ground-truth causality exists which can be augmented by data. Though this is true in many real-world scenarios, in industrial systems such ground-truth is unavailable. Williams, De Kleer, Raiman, and others have worked to provide a technical and more data-oriented perspective on causality and fault propagation. This thesis, therefore, can be viewed as an extension of the research of Williams et al., while at the same time using some of the formal definitions of Pearl (RQ2). By introducing minimal topologies (i.e. structure patterns) and analysing fault propagation, this thesis provides a formalism to analyse causality. This also extends the research done by Daigle, Biswas, and Fink [140, 203, 169], but uses definitions which are closer to the available documentation resources of production systems. For example, the definition of a component and connection model (Section 1, Part I) are made close to their bond-graph definitions. However, some of the formalisms could be avoided to make it more relevant in practice.

In terms of consistency-based diagnosis, this thesis is mainly an extension of approaches pioneered by Grastien et al. But while Grastien et al. and others have shown how diagnosis algorithms can be formulated using satisfiability and SMT, the thesis introduces a novel diagnosis methodology based on SMT logic (RQ1), which uses SMT logic as a vehicle to bridge the gap between cyber-physical system signals and a diagnosis algorithm. Following the tradition of consistency-based diagnosis, this thesis shows a novel approach to apply existing diagnosis algorithms to the new area of production systems. Some of the methods introduced in this thesis are extensions to the work done by Trave-Massuyes et al. in that they show how some mechanisms from control theory can be applied in the artificial intelligence field of fault diagnosis. Fault diagnosis in the field of control theory is also known as fault detection and isolation. Several authors have published control theoretic approaches using techniques such as analytical redundancy relations, minimum satisfiable subsets, and associated hybrid bond-graphs and observer patterns. One very important idea in these approaches is the generation of residual values. This thesis uses the idea of residual values to discretise information from cyber-physical systems to use logic (RQ3). As such, a technique is introduced which uses residuals to generate assignments to logical expressions. Each residual is assigned to an expression within the set OBS.

Overall, the presented approach can be mapped onto a generic digital twin model presented in Niggemann et al. [60]. Taking a formulation of minimal topologies of system graphs as a baseline, the thesis shows how diagnosis is carried out with residual calculation and without residual calculation. Thus, the thesis provides a justification for the approaches presented by Lunze [147], Khan [148], and several others. Besides introducing minimal system topologies and studying the benefit and creation of suitable residual values, this thesis extends common definitions from the field of consistency-based diagnosis and shows how to perform consistency-based diagnosis with cyber-physical systems. For this, a novel approach using satisfiability modulo linear arithmetic [69, 39] is presented to combine the residual-based approach with consistency-based diagnosis algorithms.

Most of the established methods used in this thesis have been proposed in different contexts and often in non-diagnostic use cases. Therefore, the novelty certainly lies in applying these methods to cyber-physical system diagnosis. The usage of SMT for diagnosis was first proposed by Grastien [39]. Ideas towards causality and formulating causality using a formalism go back to Pearl [219], and Mostermann and Biswas [25, 143]. Analysing residuals and observations is in line with thoughts by Khorasgani et al. [12] and Maier et al.

[37]. Adapting cyber-physical systems to facilitate diagnosis was identified by Niggemann et al [60], Yuan et al. [98], Tao et al. [59], and many others. Integrating observations through SMT- \mathcal{LRA} is also one of the first approaches to create a generally usable diagnosis method for cyber-physical systems. To the best of the author's knowledge, thus far no approach exists which combines these methods into a coherent method to perform consistency-based fault diagnosis for cyber-physical systems. For this, according to the contributions in this thesis, estimating physical dependencies and generating causal system descriptions is a novel idea and has thus far not been attempted in the field of consistency-based diagnosis.

Part III.
Solution

1. Solution Approach

This part of the thesis describes the realisations of the four contributions introduced in Section 1.3 of Part I. The different chapters present a novel and practical diagnosis method for cyber-physical systems using SMT- \mathcal{LRA} , two novel methods to learn system descriptions from data, and a new theoretical foundation of fault propagation.

For consistency-based diagnosis for cyber-physical systems, an algorithm (HySD) is introduced which uses propositional logic and SMT- \mathcal{LRA} expressions for its system description (Contribution 1). The innovation lies in using SMT- \mathcal{LRA} to integrate and evaluate heterogeneous process data from cyber-physical systems in the form of residual values, while employing traditional consistency-based diagnosis algorithms (Contribution 2). Thus, the new algorithm HySD is one of only few diagnosis algorithms which have been developed specifically for their use with cyber-physical systems. Derived from HySD, an algorithm HySD_simple is presented which avoids SMT- \mathcal{LRA} formulations and instead uses only propositional logic. This is beneficial for use-cases where the evaluation done by SMT- \mathcal{LRA} is part of pre-processing and thus can be removed from the algorithm itself.

However, the main algorithm HySD relies on expert-created models. These can be created with minimal structure patterns that are introduced as a novel theory to track fault propagation. The theory contains component and connection models and can be used to model a cyber-physical system with actual values and predictions. To do this, however, a significant amount of expert knowledge is required. To mitigate the reliance on expert-created models, two algorithms DDRC and DDGD are introduced, which learn propositional logic weak-fault models with a minimum of expert knowledge (Contributions 3 and 4). Such learning of system descriptions is a novel idea and, although it has been shown to technically work in model use-cases, has not been applied to diagnosing cyber-physical systems. Further, the algorithm DDA-IM is presented which translates a learned system description such that it can be used for diagnosis using HySD_simple. It must be noted, that HySD_simple and DDA-IM together make up the functionality of HySD, but are introduced as two separate algorithms. They are used, when instead of expert-defined system descriptions, only learned propositional logic system descriptions are available.

Please note that in addition to the major algorithms, in the evaluation (Part IV) a criterion is introduced to check the validity of the generated system descriptions using Algorithm CheckDiag. The algorithm empirically checks how suitable a generated system description is to diagnose faults. Therefore, CheckDiag can be used to tune the parameters of algorithms DDRC and DDGD in a generate-and-test approach.

The architecture of the different algorithms is presented in Figure 8. Input to diagnosis algorithm HySD are the expert-generated component and connection models, as well as a set of thresholds. From these, HySD generates the SMT- \mathcal{LRA} system description and performs diagnosis using the observations OBS. For algorithm DDRC, the input are two sets of time-series data S and M , as well as thresholds for the correlation quotient. Algorithm DDGD requires a single time-series with annotated fault data, a lag parameter,

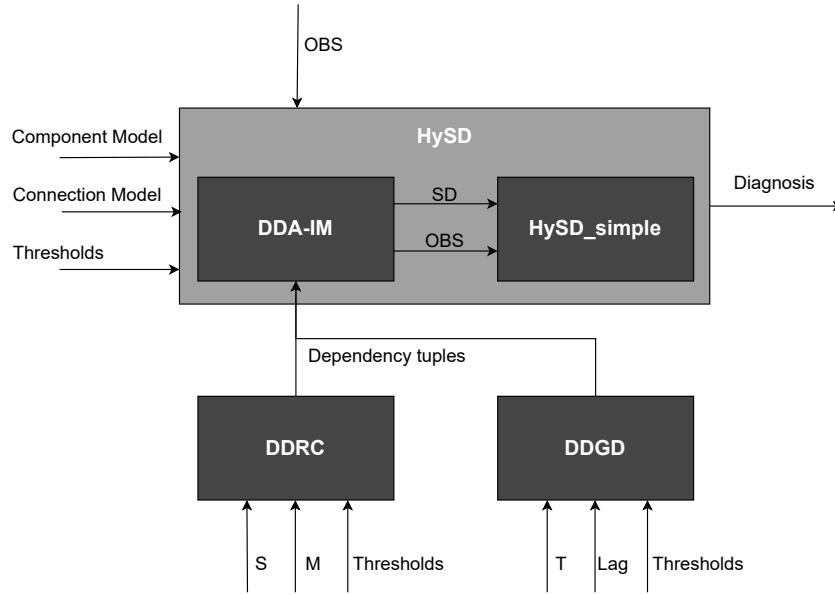


Figure 8.: Architecture of the major algorithms in this thesis. HySD integrates the functionality of algorithms DDA-IM and HySD_simple. Input to the algorithms are the expert generated models on the left side for HySD, the time-series data on the bottom for algorithms DDRC and DDGD, and the observations on the top. Output is a minimum cardinality diagnosis.

and a set of thresholds. Output of both algorithms DDRC and DDGD is a set of dependency tuples. The tuples indicate either the correlation between signals in the case of algorithm DDRC, or the Granger causality between components and process signals, in the case of algorithm DDGD. The respective tuples are converted into a propositional logic system description within DDA-IM and combined with residuals, given observations OBS. As such, DDA-IM's functionality is integrated within HySD. Finally, if only a propositional logic system description exists, compared to a SMT- \mathcal{LRA} system description, algorithm HySD_simple is used for diagnosis. Otherwise, the diagnosis functionality of HySD is used. For a complete description, refer to the next sections describing the usage and pseudocode of the algorithms in detail.

Figure 9 summarises the methods presented in this part. The black boxes within Figure 9 indicate novel approaches which are introduced within this thesis. Some of this was presented through the publications at the International Conference on Prognostics and Health Management in 2021 [75] and on the Workshop on the Principles of Diagnosis in 2022 [76]. Creating SMT- \mathcal{LRA} expressions as diagnosis models and using them for cyber-physical system diagnosis was presented by the author at the AAI (Association for the Advancement of Artificial Intelligence) conference in 2019 [69]. Further, the causality in cyber-physical systems and using it for fault diagnosis was published in March 2022 in the Engineering Applications of Artificial Intelligence [70]. The grey-colored boxes indicate topics, which are necessary for the methods in this thesis and therefore needed to be developed (in the case of the specificity check), or adapted (in the case of generating residuals), but that are no explicit contributions. For the cyber-physical system, we assume a common representation of process data available as time-series. In the case of this thesis, the representation can be assumed to be time-series of signal values at certain time points.

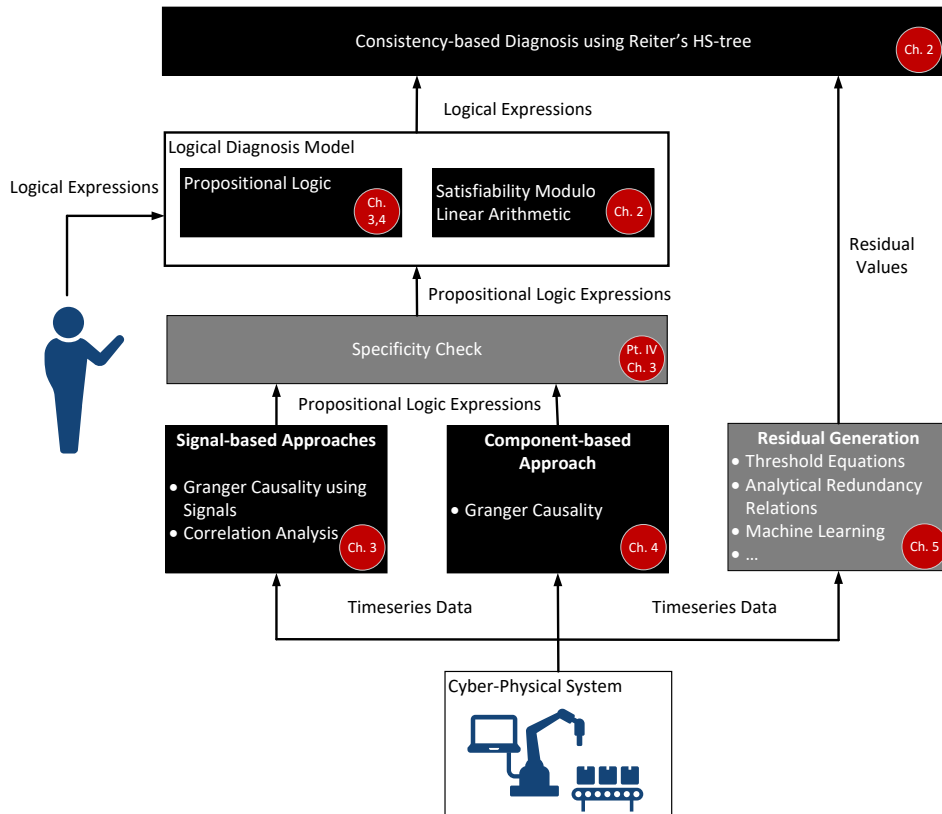


Figure 9.: The solution approach presented in this thesis part. Black boxes indicate novel developments for diagnosing cyber-physical systems. Grey boxes show corollaries, where some adaptations have been made, but that are not the focus of this thesis. Circles associate boxes to chapters within this part.

However, this assumption is generaliseable to common digital twin APIs [60]. Chapter 2 demonstrates how process data from cyber-physical systems is discretised to integrate it as observations into consistency-based diagnosis. Given process data, chapters 3 and 4 show how process data is used in order to learn logical diagnosis models and how to process observations. In particular, Chapter 3 shows how propositional logic rules can be learned from process data alone, given a single expert defined threshold. The theory for this was explained within the foundations (Chapter 1 Part I). Chapter 4 shows how a suitably parameterized Granger Causality calculation can help finding dependencies (causal influences) between expert-annotated component health states and process data. Finally, Chapter 5 presents the minimal structure patterns to create expert-generated system descriptions and track fault propagation in cyber-physical systems.

2. A New Algorithm for Consistency-based Diagnosis for Cyber-Physical Systems

This chapter introduces the Hybrid System Diagnosis algorithm (HySD). HySD is a novel algorithm to integrate a cyber-physical system model with a traditional consistency-based diagnosis algorithm to achieve a generally applicable and practicable method for fault diagnosis. Its advantage is the integration of heterogeneous process data and lightweight formalisation. After introducing the algorithm in the first section, the second section shows its usage leveraging some expert created SMT system descriptions. The third section discusses benefits and drawbacks of such expert created models.

The intuition behind the algorithm is that diagnosing faults in cyber-physical systems is still a hard task and often done by human experts. A procedure that adapts well-known diagnosis algorithms from other domains to cyber-physical systems might mitigate the work-load of human experts and speed up fault localization and recovery. The central idea behind HySD is to use SMT logic to evaluate signals from cyber-physical systems and integrate them into a logical knowledge base. This knowledge base can then be used for diagnosis using established and well-researched diagnosis algorithms. While using SMT logic for diagnosis is not new, using it to implement diagnosis for cyber-physical systems is a novel idea. In short, the goal of HySD is to check some set of system descriptions of the form $\varphi_i : c_0 \wedge c_1 \rightarrow \tau_l \leq r_0 \leq \tau_h \dots$, with $c_0, c_1 \in COMPS$, thresholds $\tau_l, \tau_h \in \mathbb{R}$ and residual $r_0 \in \mathbb{R}$, and $\bigwedge \varphi_i$ with $\varphi \in \Phi$ for consistency and identify those components that lead to faults (deviations in the residual). Classical consistency-based diagnosis has two major drawbacks: i) it is not able to process residuals, since it cannot deal with continuous values. Instead, a significant amount of pre-processing and adaptation would be necessary to transform continuous values into the Boolean observations required by classical consistency-based approaches. ii) The system description in classical approaches is usually derived from the circuit itself and its associated, known truth tables. But in cyber-physical systems this information cannot be easily derived. Therefore, to obtain a suitable system description, the formalisms introduced in Section 1.5 in Part II and the theory of fault propagation in Chapter 5 are necessary, especially their definitions of component and connection models. From these models, an SMT system description can be derived by associating sensor components to all components monitored by a single sensor (in the cone of the sensor [100]). Additionally, the sensor itself can be evaluated using residuals (Equation 1.1) and thus integrating the residual using a threshold-based evaluation within the SMT system description.

Overall, this chapter answers RQ₁ and presents Contribution 1 by showing a general approach to diagnose cyber-physical systems. It presents a novel solution to integrate many well-known consistency-based diagnosis algorithms within an SMT representation that was uniquely adapted to cyber-physical systems. Further, it is shown how diagnosis is

performed through using residual values. These are integrated into the SMT expressions and thus can be seamlessly evaluated for each diagnosis run. The main benefits for cyber-physical systems lie in the plain and explainable formulation using SMT expressions and the evaluation of residual values through different implementations of the residual generation functions $h()$ (Eq.: 1.1). These functions can be used to aggregate signals into one residual to monitor specific components or sub-modules. This can be done through statistical methods, differential equations, or even machine learning techniques. But they can also be implemented quite trivial through thresholds on single signals. The complete diagnosis procedure of algorithm HySD can be mapped onto the DigitalTwin AI reference model introduced by Niggemann et al. [60], especially to the failure mode API, while the data acquisition is part of their data and extrapolation APIs.

The input data to the algorithm consists of the connection and component models, process data, and thresholds. The SMT logic allows the formulation of residuals and the system description. Both of these can then be solved in a second step using a single call to an SMT solver to extract the minimum core and obtain the minimum cardinality diagnosis. Using this method, practitioners can leverage well-optimised traditional consistency-based diagnosis algorithms, while at the same time integrating them with cyber-physical systems which only few sources in the literature have attempted. At the end of the chapter, an example is given how this method is applied to a well-known four-tank system. Several benefits of this approach can be identified: Using consistency-based diagnosis in conjunction with calculated residuals brings the advantage to model a system how it should behave in normal operating conditions (as a weak-fault model), which is far easier than enumerating all fault modes for a strong-fault model. The logical expressions have the advantage to be interpretable and editable by human experts. Additionally, in the last 30 years a significant amount of research and experiments have been carried out to develop efficient algorithms to solve large numbers of symbolic rules to perform diagnosis [19, 22, 20]. The goal of these algorithms is to ensure the consistency between the collection of all logical expressions and a set of observations, such that

$$SD \wedge OBS \wedge COMPS \text{ is satisfiable}$$

, where SD is the set of logical expressions describing the normal behaviour, OBS are the observations from the sensors (introduced previously in Chapter 1 Part II), and $COMPS$ are the components. In later chapters it will be demonstrated how the definition of SD can be relaxed when learning logical expressions data-driven (see chapters 3, 4).

2.1. Algorithm HySD

Algorithm HySD is the solution for the function stub $diag_cont(\mathbf{X}, \gamma)$ in Section 1.1 in Part I. HySD implements the model γ through the connection model CON , the component models CM , and thresholds τ . In many cases, CON and CM need to be defined by experts (Chapter 5 presents a method to construct CM from minimal structure patterns). But once these models are created, sensor components can be identified and associated to those components that they are supposed to monitor. From this, the left-hand-side (the conjunctions) of the SMT logic expressions is created. The output from the sensor components is then evaluated through an expert-defined residual generating function (Equation 1.1). Although Equation 1.1 could also be substituted through, for example,

machine learning algorithms [220]. The thresholds are used to evaluate the residual for each signal and create the right-hand-side of the SMT logic expressions as observations. Thus, from the three input elements HySD creates a set of SMT expressions $c_0 \wedge c_1 \wedge \dots \rightarrow \tau_l \leq r_0 \leq \tau_r$, for some exemplary components c , thresholds τ , and residuals r . Differences such as concept drift during operation [221] or changes in operation mode can be mitigated through two methods: i) Updating thresholds τ dynamically, according to current system parameters. This can be extracted from programmable logic controllers or from manufacturing execution systems, for example. ii) Updating the component models CM , by using piece-wise component model functions (see Definition 17).

The formal algorithm is depicted in Algorithm 1. First a brief overview over the algorithm is given and then the individual functions are explained in detail. Functions `makeSMT_xxx()` convert the quantitative models into well-formed SMT expressions. In the way these functions are used here they aggregate information, such as the connection model, component models, residuals, and thresholds into a set of SMT expressions. Together, lines 1-3 form the qualitative model as the knowledge base. During live diagnosis, in each timestep the sensor values m from the cyber-physical system are obtained using function `getReadings`. These are discretised into observations (residual values) using `generateSymptoms` and inserted into the SMT expressions within the qualitative model. The resulting qualitative model Φ is checked for consistency. The minimum number of unsatisfiable expressions are determined using first `generateHypotheses` to obtain diagnosis candidates and then discriminated using `diagnose`. A logical solver may be used to implement `generateHypotheses` and `diagnose`. In general, however, `generateHypotheses` generates sets of conflict sets containing many possible fault candidates, while `diagnose` discriminates those faults into the minimum cardinality diagnosis. Next, the individual functions within Algorithm 1 will be explained in detail.

Algorithm 1: HySD: finding minimum cardinality diagnoses in cyber-physical systems

Data: CM, CON, τ

Result: $\tilde{\omega}$

```

1  $\varphi_p \leftarrow \text{makeSMT\_CON}(CON);$  // Def.37
2  $\varphi_c \leftarrow \text{makeSMT\_CM}(CM, \varphi_p);$  // Def.37
3  $\varphi_d \leftarrow \text{makeSMT\_Tau}(\tau, \varphi_c);$  // Def.37
4 while Running do
5    $m \leftarrow \text{getReadings}();$  // Def.33
6    $\varphi_h \leftarrow \text{generateSymptoms}(m, \varphi_d);$  // Def.35
7    $\Phi \leftarrow \text{makeSMT\_OBS}(\varphi_h);$  // Def.37
8   if  $\neg \text{check-sat}(\Phi)$  then
9      $hyp \leftarrow \text{generateHypotheses}(\Phi);$  // Def.39
10     $\tilde{\omega} \leftarrow \text{diagnose}(hyp);$  // Def.39
11    return  $\tilde{\omega};$ 
12  else
13    return  $\emptyset;$ 

```

`makeSMT_CON()`:

The function `makeSMT_CON()`: $G \rightarrow \varphi_p$, where G is a graph according to Definition 23, and φ_p is an SMT logic expression, is used to convert numerical and graph-based input into

SMT expressions. In line 1 the function generates an SMT expression from the connection model. The connection model was formally defined in Definition 23 and describes a directed graph connecting some component's output model to some other component's inputs, or to some external connections. For this, `makeSMT_CON()` traverses through the graph and associates components describing sensors (observable values) to components monitored through these sensors. The result is a tuple of the form $(c_i, c_{i+j}, \dots, c_s)$, where c_i, c_{i+j} are some components with $i, j \in \mathbb{N}$ and c_s is a component describing a sensor. This associates each sensor component to a set of other, non-sensor components. To illustrate, assume that in the running example (Section 1.4 Part I) there is a flow sensor in front of valve 1. `makeSMT_CON()` would now create a tuple of the form $(valve_0, pipe_0, tank, flowsensor)$, which associates the flowsensor to the components in front of it. Taking the tuples as a basis, `makeSMT_CON()` then converts it into an SMT template. For the connection model, the template would in a first step reduce to a propositional logic formula $\bigwedge_c c_i \rightarrow c_s$, where c_s is some sensor component and $\bigwedge_c c_i$ is a subset of associated components. For the running example this would translate into $valve_0 \wedge pipe_0 \wedge tank \rightarrow flowsensor$.

`makeSMT_CM()`:

In line 2, `makeSMT_CM(): (CM, φ_p) \rightarrow φ_c` , with CM being the set of component models and φ_p, φ_c being some set of SMT expressions, is used to integrate the component model. For this, the propositional logic expressions generated from the connection model are augmented to evaluate components. In this case, `makeSMT_CM()` inserts expressions to describe the faulty behaviour for some component, given the component model (Definition 21). For the running example this might be an SMT expression of the form $m_{flow} \leq \tau_0 \rightarrow flowsensor$, which states that the value of $flowsensor$ depends on the evaluation whether the measured value (Def. 33) is below some threshold τ_0 . `makeSMT_CM()` would thus replace each occurrence of $flowsensor$ with the output model. This would then result in the SMT expression $valve_0 \wedge pipe_0 \wedge tank \rightarrow m_{flow} \leq \tau_0$.

`makeSMT_tau()`:

Line 3 uses `makeSMT_tau(): (τ, φ_c) \rightarrow φ_d` , with τ being a set of thresholds and φ_p, φ_c being some set of SMT expressions, to insert actual thresholds into the SMT expressions. Supposing thresholds are stored within some structure, such as a look-up table, `makeSMT_tau()` traverses through each SMT expression and replaces thresholds τ with actual values. In the running example this would result in the SMT expression $valve_0 \wedge pipe_0 \wedge tank \rightarrow m_{flow} \leq 42$, where 42 (for example, meters per second) is some arbitrary threshold value.

`getReadings()`:

The function `getReadings(): $\mathbb{R}^n \rightarrow \mathbb{R}^n$` , with $n \in \mathbb{N}$ obtains time series data from some cyber-physical system hardware. Through abuse of notation we say that the input to the function is \mathbb{R}^n , although the input is read from physical devices instead of being a parameter. Therefore, it is not indicated as the function input, but only as the output. Time series data is a set of tuples $(t, s_0, s_1, \dots, s_{n-1})$, where $s \in \mathbb{R}$ is some real value and $t \in \mathbb{N}$ is some timestamp. If needed, the function may implement noise processing, scaling, centering, NA-value removal, or other common pre-processing techniques.

`generateSymptoms()`:

Function `generateSymptoms(): $\mathbb{R}^n \rightarrow \varphi_h$` with $n \in \mathbb{N}$ and output φ_h uses the obtained sensor readings from a cyber-physical system, generates residuals using $h()$ (Eq.: 1.1), and applies the discretisation function $d(\cdot)$ (Def.: 34) to each residual to obtain Boolean

observations. In many cases, these Boolean observations are also known as symptoms. The health function $h()$ is a piece-wise function which outputs o in the fault free case, and a number different from o otherwise. Formally, the function takes a snapshot of the current component's state, input, and output. Sensor readings are categorised into state variables (such as water levels in tanks or a capacitor's charge), input variables (inflow, input voltage), and output variables (outflow, output current). Then, individual residuals are generated by assigning one or more of those categorised sensor values to multiple health functions $h()$. In practice, $h()$ can also be evaluated for single signals. To illustrate, assume that within the running example it is possible to measure the tank's water level (l_0) and the flows through valve 0 and valve 1 ($flow_0, flow_1$). Then some residual r_0 could be defined through $r_0 = h(\mathbf{u} = \{l_0, flow_0, flow_1\})$, which would describe the healthiness of the tank. Contrarily, it would also be possible to define some other residual r_1 just for a single signal, such as $r_1 = h(flow_1)$, which would only describe the healthiness of the components closest to it. Please note that all residuals are real numbers $r_0, r_1 \in \mathbb{R}$. For a Boolean evaluation within some logical framework, the residuals need to be discretised through function $d()$. This is usually done through thresholds in order to obtain $\alpha_0 = d(r_0)$ and $\alpha_1 = d(r_1)$ with $\alpha_0, \alpha_1 \in \{\top, \perp\}$. Using threshold equations, it is possible to formulate $h()$ and $d()$ within one expression $\alpha = d(h(m))$ for some sensor reading m . Using a trivial implementation of thresholds, this would result in $\alpha_1 = m \leq \tau_0$, for some threshold $\tau_0 \in \mathbb{R}$.

`makeSMT_obs()`:

In line 7, `makeSMT_obs(): $\varphi_h \rightarrow \Phi$` , with φ_h and Φ being sets of SMT expressions, assigns actual values from the residuals of the cyber-physical systems to the symbols within the SMT expressions, such that $OBS \rightarrow \{\top, \perp\}$. In the running example, this would result in the expression $valve_0 \wedge pipe_0 \wedge tank \rightarrow \top$ in the fault free case.

`check-sat()`:

The function `check-sat(): $\Phi \rightarrow \{\top, \perp\}$` checks whether the SMT logic expressions Φ are satisfiable. If Φ is satisfiable, the cyber-physical system is working as intended and no faults are present within the system. If Φ is unsatisfiable, some number of faults within the system are present and Φ is said to be inconsistent according to the observations. According to Definition 15, `check-sat()` assigns truth values to symbols of Φ to satisfy the expression. If no possible assignment can be found, `check-sat()` outputs \perp . To illustrate, in the running example the expression $valve_0 \wedge pipe_0 \wedge tank \rightarrow \top$ models the normal working behaviour. If the observation is changed to \perp , such that $valve_0 \wedge pipe_0 \wedge tank \rightarrow \perp$, at least one of the symbols $valve_0, valve_1$, or $tank$ would need to be assigned \perp in order to satisfy the expression. Therefore, Φ is unsatisfiable.

`generateHypotheses()`:

In case Φ is unsatisfiable, function `generateHypotheses(): $\Phi \rightarrow hyp$` with SMT expression Φ and sets of hypothetically faulty components $hyp \in COMPS$, generates possible fault candidates by computing the maximum satisfiability of Φ (Definition 16). For this, it computes a complete variable assignment for Φ where it attempts to assign \top to as many symbols as possible. The resulting set hyp is a set of possible candidate sets, each containing possible faulty components. To illustrate, using two SMT expressions: i) $valve_0 \wedge pipe_0 \wedge tank \rightarrow \perp$ and ii) $pipe_2 \wedge valve_3 \rightarrow \perp$ some possible candidate sets are $\{pipe_0, pipe_2\}$, $\{tank, pipe_2\}$, $\{valve_0, valve_3\}$, or other permutations. In many cases, the set of hypotheses is still too large and complex to be useful in practice.

diagnose():

Function `diagnose()`: $hyp \rightarrow \tilde{\omega}$, with hyp being a set of hypotheses computes the minimum hitting set $\tilde{\omega}$ from the set of hypotheses (Def.: 40). This is done by finding the set with the smallest cardinality which explains all of the faulty behaviour within the cyber-physical system. In the context of this thesis this is done by computing Reiter's HS-tree [17], but other solutions based on extracting the minimum core from a satisfiability solver such as Z3 [222] are possible as well.

So far it was shown how HySD is employed using CM , CON , and τ as input. Namely, the component models, connection models, and thresholds. But in some use-cases, experts may be able to create an SMT-logic system description without first going through the formalisations CM and CON . Instead, they may be able to create the system description directly. For this case, algorithm `HySD_simple` was developed, as a special case of algorithm `HySD`. Therefore, it is possible to write `HySD` with a slight change in notation to create algorithm `HySD_simple`.

Algorithm 2: `HySD_simple`: finding minimum cardinality diagnoses in cyber-physical systems with existing system descriptions

Data: SD, OBS

Result: $\tilde{\omega}$

```

1 while Running do
2    $\Phi \leftarrow \text{generateSymptoms\_simple}(OBS, SD);$  // Def. 35
3   if  $\neg \text{check-sat}(\Phi)$  then
4      $hyp \leftarrow \text{generateHypotheses}(\Phi);$  // Def. 39
5      $\tilde{\omega} \leftarrow \text{diagnose}(hyp);$  // Def. 39
6     return  $\tilde{\omega}$ ;
7   else
8     return  $\emptyset$ ;
```

Please note the absence of the `makeSMT_xxx()` functions, due to SD already being represented in SMT logic, or propositional logic expressions. Further, the parameters of `generateSymptoms` were changed in order to integrate the model SD and the observations OBS , which are now assumed to come from some external source. This is different to Algorithm 1, which encoded this information within the iteratively created SMT expressions φ and the function `getReadings`, respectively. Therefore, we use the redefined function `generateSymptoms_simple()`.

`generateSymptoms_simple()`:

Function `generateSymptoms_simple()`: $OBS, SD \rightarrow \Phi$ with observations OBS , propositional logic system description SD , and output Φ , uses the obtained sensor readings from the cyber-physical system, generates residuals using $h()$ (Eq.: 1.1), and applies the discretisation function $d(\cdot)$ (Def.: 34) to each residual to obtain Boolean observations. This is similar to function `generateSymptoms()` in algorithm `HySD`. The difference is that, instead of entering threshold equations directly into SMT- \mathcal{LRA} expressions, function `generateSymptoms_simple()` performs pre-processing and thus calculates $\alpha = d(h(m))$ without SMT. Instead, the calculation needs to be performed in program code, such that

only the result α in the form of Boolean values is assigned as observations into the right-hand-side of the variables in SD. Some possible output would then be the propositional logic expression $valve_0 \wedge pipe_0 \wedge tank \rightarrow \top$.

2.2. Using HySD

This section shows how to use HySD to diagnose cyber-physical systems. In particular, it highlights how a real cyber-physical system may be modelled and diagnosed using the methods developed so far. For clarity, the cyber-physical system is approximated as a linear system characterised through its inputs, outputs, and state. For non-linear systems the behaviour needs to be modelled more complex. To use HySD, we will describe the construction of the health function $h()$ and the residual values. For the residual generation an observer-pattern based on the state-space representation of a linear time invariant cyber-physical system is chosen. The residuals from the observer are processed to obtain Boolean residuals. Then these Boolean residuals are captured as observations in SMT logic for diagnosis. Each of these steps is now discussed in detail. For all methods, a suitably chosen sampling interval is assumed. The interval depends entirely on the system under investigation and can usually be in the range from milliseconds to several seconds or even minutes. A wrongly chosen interval may lead to the algorithm missing important system transitions.

At first, a linear state space representation is introduced, where the system state is propagated through the explicit Euler method. This represents some cyber-physical system with its states, inputs, and outputs. Suitable outputs can be assumed to be calculated through the output model (Def.: 24), while the adjacency matrices describing the interactions of the state-space representations can be created from the connection model (Def.: 23). The state-space itself is defined through

$$\begin{aligned} \mathbf{x}(t+1) &= f(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) &= n(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\tau}) \end{aligned} \quad (2.1)$$

where $\mathbf{x}(t+1) \in \mathbb{R}^x$ is a vector of the state in the next time step, $\mathbf{x}(t) \in \mathbb{R}^x$ is the current state vector, $\mathbf{u}(t) \in \mathbb{R}^u$ is the observable input vector, $\mathbf{y}(t) \in \mathbb{R}^y$ is the observable output vector, and $\boldsymbol{\tau} \in \mathbb{R}^\tau$ is a vector of expert-defined threshold values. Function $n(\cdot)$ is the output model defined in Definition 24. The function $f(\mathbf{x}, \mathbf{u}), f : \mathbb{R}^{x \times u} \rightarrow \mathbb{R}^x$ models the current state and its current input and from this computes the next state. Therefore, it is possible to write

$$f(\mathbf{x}(t), \mathbf{u}(t)) = A\boldsymbol{\Delta}(\mathbf{x}, \mathbf{u}, t) + B\mathbf{u}(t) \quad (2.2)$$

with $A \in \mathbb{R}^{x \times u}$ being a matrix of real numbers stating which current state influences the next state of the same component and $B \in \mathbb{R}^u$ being the incidence vector of connected components showing the connections between the system's components, and $\boldsymbol{\Delta}$ being a vector of functions. $B_{ij} = 1$ denotes an input, $B_{ij} = -1$ an output, and $B_{ij} = 0$ indicates no connection, with $i, j \in \mathbb{N}$.

Within this section we assume the existence of some system description Φ that was generated from some connection model using functions `makeSMT_xxx()`. `generateSymptoms()` creates symptoms from observations. Within `generateSymptoms()` are piece-wise residual

generating functions as specific implementations of the general health function $h(\cdot)$ (Eq.: 1.1).

$$\text{generateSymptoms}(m_i, \tau_i) = \begin{cases} 0 & \text{if } m_i \leq \tau_i, \\ 1 & \text{else} \end{cases} \quad (2.3)$$

that relate the thresholds to the current values \mathbf{m} , where in the case of linear time invariant systems $m \in \mathbf{x}$ is a measurement of the tank levels. For the linear case $\text{generateSymptoms}(\cdot)$ can be computed with SMT solvers. For non-linearities, $\text{generateSymptoms}(\cdot)$ may be calculated using an external method, where only the calculated result is inserted as a variable into an SMT expression. OBS implements an observer pattern on the system.

$$\text{OBS} = \begin{bmatrix} \text{generateSymptoms}(m_0, \tau_0) \\ \vdots \\ \text{generateSymptoms}(m_n, \tau_n) \end{bmatrix} \quad (2.4)$$

For this, it uses function $\text{getReadings}()$ with $m \in \mathbf{x}, \mathbf{u}, \mathbf{y}$, and thresholds τ . After having discussed the basic value propagation it is now possible to transform the information into logical statements using separate symptoms. In the following, the observer pattern is used to construct SMT expressions of the residuals through hypotheses for fault diagnosis, which can then be used by standard consistency-based diagnosis algorithms. Weak-fault models are used to model the normal behaviour of the system. Diagnosis requires the tuple (SD, COMPS, OBS). SD is given by the set of SMT expressions $\text{SD} = \bigwedge_{\varphi} \varphi$. Following from Equation 2.4 it is possible to implement $\text{generateSymptoms}(m_0, \tau_0)$ using thresholds

$$\text{OBS} = \begin{bmatrix} |m_i(t) - \hat{\alpha}_i(t)| \leq e \\ \vdots \\ |m_j(t) - \hat{\alpha}_j(t)| \leq e \end{bmatrix} \quad (2.5)$$

where vector $\text{OBS} \in \{\top, \perp\}^*$ is the comparison between the observation $m_i(t)$ and the model prediction $\hat{\alpha}_i(t)$ and $e \in \mathbb{R}$ is some error bound. The model prediction can be obtained from experts, or can be learned through statistical and machine learning methods. If this comparison is smaller than some error bound e , the corresponding component is healthy. In practice, $|m_i(t) - \hat{\alpha}_i(t)| \leq e$ may also be implemented through two thresholds $\tau_l, \tau_h \in \mathbb{R}$, such that $\tau_l \leq m_i \leq \tau_h$, where the thresholds are expert-defined. If the elements of OBS are semantically interpreted through an SMT solver, it is possible to obtain the diagnosis vector

$$\text{OBS} = [r_0 \quad \dots \quad r_n]^T$$

with residual $r_i \in \{\top, \perp\}$. This vector is the system's health state and shows for each component whether or not it is faulty, given the current observations. What is missing is a discrimination of these faults into the minimum cardinality diagnosis. For this, the system description from functions makeSMT_xxx is used, such that OBS is associated to SMT expressions. By interpreting the statements, it is possible to translate the numerical data within the state-space model of the cyber-physical system into symbolic information used for diagnosis through the vector OBS. For each new time step the SMT expressions are reevaluated, thus generating new hypotheses. Reevaluation is necessary since we do not specify a sampling rate for the underlying data. Therefore, the diagnosis starts as soon as sufficient data is available. This increases computation time, but will lead to faster diagnosis in the real world.

After having described an implementation of `getReadings()` and `generateSymptoms()` explicitly, it is now possible to employ algorithm `HySD_simple`, as we assume that Φ is given. Using the observations `OBS`, `HySD_simple` is called as `HySD_simple((SD, OBS))` and computes the minimum cardinality diagnosis.

2.3. Example: HySD

We will illustrate the formalisations within the more complex four-tank system (see Fig.: 3). Sensor readings are measurements such as flow rates [m/s] and the tank level [m]. These sensor readings are processed by some residual generation algorithm. Threshold equations (Def. 34) for valves are expressed as an SMT expression of the form $d_i : \alpha_i = \tau_l \leq m_i \leq \tau_h$ with $i \in \mathbb{N}$ and thresholds τ_l, τ_h . It is assumed that the inflow and outflow can be measured at the associated valves in each inflow and outflow pipe. For the state, input, and output vectors we thus have

$$\mathbf{x} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} \text{sp_flow}_0 \\ \text{sp_flow}_1 \\ \vdots \\ \text{sp_flow}_6 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} \text{flow}_0 \\ \text{flow}_1 \\ \vdots \\ \text{flow}_6 \end{bmatrix}$$

where `sp_flow` are setpoints for the valve-throughput, `flow` are the measured flows, and h are the water levels within the tanks. Matrix

$$B = \begin{bmatrix} 1 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & -1 \end{bmatrix}$$

is the incidence matrix showing the connections. All of these parameters can be obtained from the connection model `CON` and component models `CM`. Within the component models, equations govern the flow of information. In the process industry difference equations are a common tool to propagate values and are conveniently supported in SMT.

To model the water level in each tank it is possible to use difference equations

$$x(t+1) = \frac{1}{D}(Q_i(t) - C_d a \sqrt{2gx(t)}) \quad (2.6)$$

with the tank diameter $D \in \mathbb{R}$, the discharge coefficient $C_d \in \mathbb{R}$, the size of the orifice $a \in \mathbb{R}$, and the gravitational constant $g \in \mathbb{R}$. $Q_i(t) \in \mathbb{R}$ is the sum of all inputs to the current tank $Q_i = \sum_{q \in u} q$ for each relevant input q in \mathbf{u} . A vector describing the current output of a state-based component (such as a tank) $\Delta(\mathbf{x}, \mathbf{u}, t)$ can be created, given its new state.

$$\Delta(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} u_i(t+1) = \frac{1}{D_i}(C_{d,i} a_i \sqrt{2gx_i(t)}) \\ u_j(t+1) = \frac{1}{D_j}(C_{d,j} a_j \sqrt{2gx_j(t)}) \\ \vdots \end{bmatrix}.$$

With this model it is possible to propagate the state of the system as it evolves through time by calculating each component's output value. However, given this information, a control system cannot yet determine the full behaviour of the system. To calculate `generateSymptoms(·)` two functions $o(h, \tau)$ and $l(h, \tau)$ are introduced. These are concrete implementations for the upper and lower thresholds. The function $o(h, \tau)$ indicates upper thresholds for values and is calculated by

$$o(h_i, \tau_i^o) = \begin{cases} 0 & \text{if } h_i \leq \tau_i^o, \\ 1 & \text{else} \end{cases}. \quad (2.7)$$

Likewise, the lower thresholds are calculated by the function $l(h_i, \tau_i^l)$. Applying matrix OBS to the four-tank model results in

$$\text{OBS} = \begin{bmatrix} o(h_0, \tau_0^o) \\ \vdots \\ o(h_3, \tau_3^o) \\ l(h_0, \tau_0^l) \\ \vdots \\ l(h_3, \tau_3^l) \end{bmatrix} \quad (2.8)$$

For ease of notation we use τ_i^o to denote the threshold for the upper limit of tank i and τ_i^l to denote the lower limit of tank i .

Formulating weak-fault models with SMT- \mathcal{LRA} for the valves in the four tank model results in

$$\varphi_{V,i} : (flow_i^l \leq flow_i) \wedge (flow_i^u \geq flow_i) \rightarrow r_i. \quad (2.9)$$

When the valve is healthy its actual flow will be between two thresholds ($flow_i^l$ and $flow_i^u$). Using *CON* and *CM*, the system description Φ in Line 7 of Algorithm 1 would contain the propositional logic system description

$$\begin{aligned} v_0 \wedge t_0 &\rightarrow r_0^{fl} \\ t_0 \wedge t_3 \wedge v_3 &\rightarrow r_3^{fl} \\ t_0 \wedge t_1 \wedge v_1 &\rightarrow r_1^{fl} \\ t_0 \wedge t_2 \wedge v_2 &\rightarrow r_2^{fl} \\ t_0 \wedge t_0 \wedge v_3 \wedge t_3 &\rightarrow r_0^l \\ t_0 \wedge v_1 \wedge t_1 \wedge v_4 \wedge t_3 &\rightarrow r_1^l \\ t_0 \wedge v_2 \wedge t_2 \wedge v_5 \wedge t_3 &\rightarrow r_2^l \\ t_0 \wedge t_1 \wedge t_2 \wedge v_3 \wedge v_4 \wedge v_5 \wedge v_6 &\rightarrow r_3^l \end{aligned} \quad (2.10)$$

The right-hand-side of each implication is the observation that was processed from the sensor readings through comparison with a model prediction. The left-hand-side consists of the symbolic names (*name*) of the components, while their position is derived from the connection model. Thus, an expert would traverse through the connection model and

associate all those components to a residual that might have an influence, according to the expert's opinion. Using HySD, the set of expressions Φ is valid for the system's normal operation. Diagnosis is done when the set of expressions becomes invalid, due to changes in the residuals on the right-hand-side. HySD uses non-monotonic reasoning to find those elements in Φ , whose removal would lead to the system description becoming satisfiable. For example, with the health assignment $\{r_0^{fl} = \perp, r_0^l = \perp\}$ and all other observations being \top , the removal of lines $v_0 \wedge t_0 \rightarrow r_0^{fl}$ and $t_0 \wedge v_0 \wedge v_3 \wedge v_1 \wedge v_3 \rightarrow r_0^l$ would make Φ consistent again. The objective, however, is to identify the minimum set that is responsible for the inconsistency. The calculation of this set is achieved by calling one of the algorithms introduced in Section 1.7 in Part II. As a result, we obtain the hitting set with the minimum cardinality diagnosis: $\{\{v_0\}, \{t_0\}\}$.

Manually modelling cyber-physical systems brings the advantage that unobserved behaviour can be inferred using suitable expert-generated methods. If, for example, the level of a tank cannot be observed, a differential equation can be used to calculate the tank's level given its input and output. Additionally, it is possible to specify exactly how each component is connected with other components and how those interactions take place. This can be done very well by process experts and can in many cases be somewhat automated from design documentation. Creating these expert models is similar effort to the formulation used by Mostermann et al. [25] and their bond-graphs. However, it is also evident that the presented method requires some significant amount of manual modelling effort that may be quite costly in practice. How to create system models SD automatically from data is the subject of the next chapters.

3. A New Algorithm to Learn System Descriptions from Process Signals

The last chapter has shown that creating the system description SD of the diagnosis system tuple $(SD, COMPS, OBS)$ takes a lot of manual effort and is thus costly in industry. Also, cyber-physical systems change over time, as parts of the system are added, removed, or reconfigured. Thus, experts would need to create new models intermittently. This chapter answers RQ 3.1 and presents Contribution 2 by introducing two algorithms: i) The algorithm *Data-driven Diagnosis Rule Creation* DDRC to create a system description automatically from process data using correlations. This is the main contribution of this chapter. ii) Through the introduction of the *Data-driven Diagnosis Algorithm* DDA-IM to diagnose cyber-physical systems. DDA-IM uses an implementation of $HySD_simple()$ (Chapter 2) to translate the output of DDRC into propositional logic. Using DDRC, an uninformed approach to fault diagnosis was adopted which focuses on minimizing external input (expert knowledge). In particular, with DDRC, a correlation-based method is introduced which creates propositional logic system descriptions from process data.

The intuition in this chapter is two-fold: i) The goal is to obtain a propositional logic system description purely from process data (with minimal input from experts). This is necessary for algorithms such as $HySD$ and $HySD_simple$ to perform diagnosis almost autonomously. ii) We evaluate how diagnosis results are influenced by using only signal amplitudes for diagnosis rather than using residuals. Using only signal amplitudes is easier to implement in practice, and avoids the careful formulation of residuals. Signal amplitudes are also used by many phenomenological approaches in machine learning [69]. We will use the insights gained to compare this approach to residual-based approaches in later chapters.

Overall, DDRC attempts to find high correlations defined through some threshold, and then creates dependencies between signals. The only requirement is that the set of input signals to the algorithm can be divided into a set of signals that influence the production process (denoted as S) and a set of signals from quality control measurements (denoted as M).

The presented algorithm DDRC is based on automatically calculating the Spearman correlation (see Chapter 1, Part II) between signals in historical process data. We define process data as two time series S and M . S is of the form $((t_0, s_0), \dots, (t_{n-1}, s_{n-1}))$, with t being a timestamp and some value $s \in \mathbb{R}$. M contains signals from quality control measurements $m_i \in \mathbb{R}$, and is of the form $((t_0, m_0), \dots, (t_{n-1}, m_{n-1}))$ with $S \cap M = \emptyset$. The existence of the two sets M and S is important and can in many cases be obtained from cyber-physical systems. For example, injection molding machines and compounding processes generate process data S during production, while associated laboratories and automated quality control devices generate data-set M . Output of algorithm DDRC is a set of tuples $(s_0, s_1, \dots, s_i, m_0)$, $i \in \mathbb{N}$, in which each tuple associates one signal $m \in M$ to one or more signals $s \in S$. Algorithm DDA-IM then creates a propositional logic system description from these tuples by connecting signals s through conjunctions and creating an implication between the

conjunctions and signal m . It must be noted that DDRC is only defined on signals. Therefore, experts are required to map signals S to component symbols $COMPS$.

3.1. Algorithms DDRC and DDA-IM

First we will provide a brief overview over the functionality of algorithm DDRC. Then, the algorithm is presented and analysed function by function. We assume that for cyber-physical systems a sufficient correlation between two signals implies a certain amount of temporal dependency (which Balzereit et al. [81] called causal hypotheses). Please note that this is a strong assumption that only holds if sets S and M are well defined. Input to algorithm DDRC is time-series $T = S \cup M$. The set of the signal names of a vector shall be denoted with the tilde symbol (\tilde{T}). Therefore, the signal names of time-series T , S , and M are denoted as $\tilde{T} = \tilde{S} \cup \tilde{M}$. Table 9 shows some exemplary data for time-series T as some possible input to DDRC. At time point $i \in \mathbb{N}$ a fault is injected, where sensor s_1 deviates from its normal value.

t	s_0	s_1	\dots	s_{n-1}	m_0	m_1	\dots	m_{q-1}
0	5	100		6.6	1	12		80
i	5.5	30		6.3	3	15		81
$i + 1$	5.3	34		6.9	3	15		80
k	5	102		6.5	1	12		81

Table 9.: Signal division into sets S and M . $n, i, k, q \in \mathbb{N}$

DDRC (Alg.: 3) calculates the Spearman correlation coefficient over a time period of normal and nearly identical production. Therefore, changes in recipes and operating modes need to be avoided to mitigate concept drift. The calculated Spearman correlation is then converted into a lower triangular matrix. The matrix is then traversed and evaluated according to threshold τ . Calculating the correlation ρ (Eq. 1.3) from the signals $S \cup M$ describes dependencies between signals in S to signals in M . ρ is a matrix with rows being signals from set S and columns being signals from set M . The correlation is calculated during normal system operation, without stops or parameter changes. ρ is post-processed such that one signal $m \in M$ is associated to one or many signals $s_i \in S$. These dependencies m to several s_i are captured in a set of tuples P , such that for each tuple one signal $m_j \in M$ is associated to $\forall s_i \in S$, with $\rho_{s_i} > \tau$ for all signals $m_i \in M$. This ensures that only those signals are represented within the tuple whose Spearman correlation is greater than threshold τ and that thus have a meaningful contribution to the sensor value m_j (i.e. the assumed causation). The result is a set of tuples with variable length of the form $(s_0, s_1, \dots, s_i, m_j)$. These tuples are then used as the input for algorithm DDA-IM. In the end, the set of tuples P is of the form

$$\begin{aligned}
 &(s_0, s_1, \dots, s_x, m_0) \\
 &(s_0, s_1, \dots, s_y, m_1) \\
 &(s_0, s_1, \dots, s_z, m_2) \\
 &\vdots
 \end{aligned} \tag{3.1}$$

with $x, y, z \in \mathbb{N}$, for example. This means, a subset of sensor values s is associated with one measurement variable $m \in M$.

Algorithm 3: DDRC: data-driven diagnosis rule creation

Data: S, M, τ

Result: P

```

1  $\rho \leftarrow \text{Spearman}(S \cup M);$  // Eq.1.3
2  $edges \leftarrow \emptyset;$ 
3 foreach  $row, column \in \rho$  do
4   | if  $|\rho(row, column)| > \tau$  then
5   | |  $edges \leftarrow edges \cup (row, column);$ 
6  $P \leftarrow \emptyset;$ 
7 foreach  $m \in M$  do
8   |  $S_m \leftarrow \text{select}(edges, m);$ 
9   |  $t \leftarrow (S_m, m);$ 
10  |  $P \leftarrow P \cup t;$ 

```

Spearman():

Function **Spearman()**: $\mathbb{R}^{(|S|+|M|) \times n} \rightarrow \mathbb{R}^{M \times S}$, where $n \in \mathbb{N}$ is the length of the time series data, calculates the Spearman correlation (Eq.: 1.3) between sets M and S .

select():

Function **select()**: $(edges, \mathbb{R}) \rightarrow S_m$ associates edge tuples $\rho_{i,j}$ from correlation matrix ρ and $i, j \in \mathbb{N}$ to the tuple S_m . In the end S_m contains exactly those signal names from S that have a high correlation to M .

The algorithm DDA-IM (Listing 4) takes the output from algorithm DDRC and creates symbolic, propositional logic expressions for diagnosis out of the set of tuples P , reads process data and then calls **HySD_simple**. DDA-IM is used as a wrapper to pass the learned propositional logic system description and observations to **HySD_simple** for diagnosis.

Algorithm 4: DDA-IM: data-driven diagnosis algorithm for physical systems

Data: P

Result: ω_{min}

```

1  $\Phi \leftarrow \text{toSymbolic}(P);$  // Def.37
2  $\alpha \leftarrow \text{getReadings}();$  // Def.34
3  $\omega_{min} \leftarrow \text{HySD\_simple}(\Phi, \alpha);$  // Def.40

```

toSymbolic():

The function **toSymbolic()**: $P \rightarrow \Phi$, with a set of tuples P , converts the tuples into a set of propositional logic expressions Φ . One expression is generated for each tuple by inserting a conjunction (\wedge) between the sensors of set S and creating an implication to each associated variable m . Therefore, the result is of the form: $\varphi_i : \bigwedge_{s_i} s \rightarrow m$ for all m and associated signals s_i .

`getReadings()`:

The function `getReadings(): $\mathbb{R}^n \rightarrow \mathbb{R}^n$` , with $n \in \mathbb{N}$ obtains time-series data from some cyber-physical system hardware. The function is similar to the one within algorithm HySD. The difference is that to use `HySD_simple()` the readings need to be obtained from outside of the diagnosis algorithm. This is due to the design decision to keep the interface of `HySD_simple()` as close as possible to classical consistency-based diagnosis algorithms. Therefore, its required parameter `OBS` needs to be obtained from outside of `HySD_simple()`.

`HySD_simple()`:

Function `HySD_simple()` (Alg.: 2) performs the consistency-based diagnosis. The result is a set of diagnosis candidates ω_{min} . The accuracy of the diagnosis and the performance of algorithm `HySD_simple()`, however, largely depends on a suitably generated system description SD. If the process data as the input to algorithm DDRC contains too many spurious correlations or can only be described through non-linearities, SD may contain too little or even wrong information. This is different from classical consistency-based diagnosis, where the performance of diagnosis approaches is always attributed to the diagnosis algorithm itself, while SD is assumed to be consistent and known a-priori.

3.2. Example: DDRC and DDA-IM

This section describes an example usage of algorithms DDRC and DDA-IM. Suppose the existence of a time-series of the form of Table 9. Calculating the Spearman correlation coefficient from this time series might result in a matrix, such as the one depicted in Table 10.

ρ	s_0	s_1	\dots	s_{n-1}	m_0	m_1	\dots	m_{q-1}
s_0	1.0	0.1		0.1	1	0.1		0.1
s_1	0.1	1.0		0.1	0.1	0.7		0.1
\vdots								
s_{n-1}	0.1	0.1		1.0	0.1	0.1		0.1
m_0	0.1	0.1		0.1	1.0	0.1		0.1
m_1	0.1	0.7		0.1	0.1	1.0		0.1
\vdots								
m_{q-1}	0.1	0.1		0.1	1	0.1		1.0

Table 10.: Correlation matrix ρ of algorithm DDRC. $n, i, k, q \in \mathbb{N}$

From the table DDRC would then extract the dependency tuples. In the case of Table 10 this would be the tuple (s_1, m_1) , since only between these two signals the correlation is above some threshold. In reality, the correlations exhibit more variance. Therefore, we will now analyse how the tuples are created on simulated data for the one-tank running example. Please note that the variable names are due to the simulation, which was performed using OpenModelica 1.13.

Given a suitable time-series, it is possible to employ algorithm DDRC to calculate ρ and to obtain the tuples P .

$$\begin{aligned}
& (t1level, t1medT, valve4flow) \\
& (t1level, t1medT, t2level, t2medT, valve5flow) \\
& (t1medT, valve6flow) \\
& (t1level, t1medT, t2t4flow) \\
& (t1level, t1medT, t2level, t2medT, t3t4flow) \\
& (t1medT, t4sinkflow)
\end{aligned} \tag{3.2}$$

The tuples have been identified on a threshold $\tau = 0.3$. The rightmost signals are the quality signals $m \in M$. Here *quality* was defined to mean those signals, which signify normal production, such as flow-signals. All other available signals were taken as the signals in set S . The prevalence of the medium temperature $tXmedT$ is due to the simulation being executed in a very stable process with low variance. With the tuples P , it is now possible to use algorithm DDA-IM to translate the tuples into a propositional logic system description using `toSymbolic ()` to obtain

$$\begin{aligned}
& t1level \wedge t1medT \rightarrow valve4flow \\
& t1level \wedge t1medT \wedge t2level \wedge t2medT \rightarrow valve5flow \\
& t1medT \rightarrow valve6flow \\
& t1level \wedge t1medT \rightarrow t2t4flow \\
& t1level \wedge t1medT \wedge t2level \wedge t2medT \rightarrow t3t4flow \\
& t1medT \rightarrow t4sinkflow
\end{aligned} \tag{3.3}$$

by inserting implications and conjunctions, accordingly. Now it is possible to assign observations to the quality signals. This is done through function `getReadings ()` within DDA-IM. In propositional logic this is realised through the following assignments.

$$\begin{aligned}
& valve4flow \Leftrightarrow \top \\
& valve5flow \Leftrightarrow \top \\
& valve6flow \Leftrightarrow \top \\
& t2t4flow \Leftrightarrow \top \\
& t3t4flow \Leftrightarrow \top \\
& t4sinkflow \Leftrightarrow \top
\end{aligned} \tag{3.4}$$

if no fault exists within the system. \top denotes a true value and is interpreted such that no fault exists. To show how to perform diagnosis on the running example, we assume Φ to be grounded with a random health assignment, using the propositional logic system

description from `toSymbolic()`.

$$\begin{aligned}
 & t1level \wedge t1medT \rightarrow valve4flow \\
 & t1level \wedge t1medT \wedge t2level \wedge t2medT \rightarrow valve5flow \\
 & \quad t1medT \rightarrow valve6flow \\
 & \quad t1level \wedge t1medT \rightarrow t2t4flow \\
 & t1level \wedge t1medT \wedge t2level \wedge t2medT \rightarrow t3t4flow \\
 & \quad t1medT \rightarrow t4sinkflow \\
 & valve4flow \Leftrightarrow \top \\
 & valve5flow \Leftrightarrow \top \\
 & valve6flow \Leftrightarrow \perp \\
 & t2t4flow \Leftrightarrow \perp \\
 & t3t4flow \Leftrightarrow \top \\
 & t4sinkflow \Leftrightarrow \top
 \end{aligned} \tag{3-5}$$

In this case, the algorithm `HySD_simple()` would compute the set of diagnoses as the two sets $\{\{t1level, t1medT\}, \{t1medT\}\}$. Identifying the minimum cardinality diagnosis would mean to take the smallest set of diagnoses, thus returning $\{t1medT\}$ as the correct diagnosis.

Using the presented uninformed and correlation-based approach with algorithm `DDRC` has some important limitations. Since process data in the form of time-series is the only source of information one strong assumption is that this data-set needs to encompass all behaviour that the cyber-physical system exhibits. This requires a sufficient instrumentation and sampling rate [223]. Also, spurious correlations cannot be ruled out. Since no ground truth exists, the process data is *believed* unequivocally. Further, correlation analysis is a linear method. Non-linearities, therefore, may not be recognised correctly. Thus, spurious correlations may lead to too many dependencies, while the limitations on linearity may lead to too few complex dependencies. The evaluation in Section 3.2.3 of Part IV shows that for simple use-cases, where S and M are clearly defined, algorithm `DDRC` generates useful system descriptions despite its limitations. In the next chapter, algorithm `DDGD` is presented to mitigate many of the shortcomings of algorithm `DDRC`.

4. A New Algorithm to Learn System Descriptions from Residuals and Component Health States

The last chapter has shown the maximally uninformed approach to create propositional logic system descriptions. This chapter presents a novel, more-informed, but still data-driven method.

In particular, this chapter presents an alternative solution for RQ 3.1, by introducing algorithm DDGD which uses Granger Causality (see Chapter 1 Part II) [107] to find dependencies within cyber-physical systems. Granger causality is a common method to learn approximations of causality from data in the natural sciences [184, 181, 109]. Further, it has been used in many applications [98, 224] and to detect temporal dependencies in sequences of data [93]. Given two signals, Granger Causality determines whether past values of one signal affect the second signal. If those past values have an effect on the current values of the second signal it is said that signal 1 Granger-causes signal 2. Calculating the Granger Causality is beneficial in use-cases where the expert knowledge needed for causal analysis cannot be determined. For example, Chapter 1 in Part II has shown that for many types of causal analysis a ground truth of causal dependencies needs to be known. But in many use-cases that need to learn dependencies from data, exactly this ground truth is unavailable.

In this thesis, Granger Causality is used to find dependencies in cyber-physical systems. Input to the algorithm are two matrices: The first is a matrix with component health states F . This specifies for each component at which point in time it was healthy or unhealthy. The second is a matrix with residuals created from process data. The task of Granger Causality is to find the dependency between a change in the component health state and the state of the residuals. In general, the method is generalisable to real valued data, but in practice we limit it to Boolean component health states and Boolean or real valued residuals. Therefore, the presented method is a partial solution to the function $diag_cps(\mathbf{X}, \mathcal{L})$ that was defined in Section 1.1 in Part I, where \mathcal{L} is a set of health states. The output is then a dependency that specifies, which residual is associated to which components. The dependencies then need to be translated into propositional logic. This is done similar to algorithm DDRC by calling algorithm DDA-IM in Chapter 3. It must be noted that for Granger Causality the strong requirement is here that a collection of faulty states exists, whose influence explains the behaviour of the residual values. While DDGD learns Granger causalities in a supervised manner, its goal is to obtain a weak-fault model system description. This is achieved by only including the information specifying which residuals were affected at the moment a fault occurs. The assumption is that the data used for learning the algorithm contains most of the variances that are attributable to fault modes. Using this assumption, it is possible through the weak-fault model formulation to also recognise faults that were not part of the training data. As was the case with algorithm DDRC, at the end a set of tuples is created, where each tuple is of the form (c_0, c_1, \dots, r_0) with components c and residual r .

While using Granger Causality in applied use-cases is not new, the novelty and practical use of the presented algorithm lie in its application to generate a propositional logic system description for consistency-based diagnosis.

4.1. Algorithm DDGD

This section describes how Granger Causality can be applied to cyber-physical systems algorithmically. The general method is similar to the one used in Chapter 3 with algorithm DDRC. There DDRC used as input a matrix of process data that was divided into sets S and M and created an output of dependency tuples. In this chapter we use a matrix divided into sets F and \mathcal{R} , where $F \subseteq \text{COMPS}$ is a set of component health states (Def.:35) and \mathcal{R} is a set of residuals calculated through functions such as Equation 1.1. We use the potentially smaller set F instead of COMPS, since Granger Causality detects only those components whose health-state exhibits some variance.

Algorithm DDGD uses two sets of signals F and \mathcal{R} . The first is a time-series with a fault injection created in the form of $F = (c_0, c_1, c_2, \dots, c_{n-1})$, where $c \in F \subseteq \text{COMPS}$. Each c_i is a named Boolean variable indicating if the component is faulty at a given time with $\tilde{T} = \tilde{F} \cup \tilde{\mathcal{R}}$, where $\tilde{\mathcal{R}}$ is the set of names of the residuals (note the difference to $\tilde{T} = \tilde{S} \cup \tilde{M}$ in the previous chapter). Time-series T is defined here as the union of F and \mathcal{R} . Table 11 shows exemplary data. At time point $i \in \mathbb{N}$ a fault is injected, where component c_0 is set to faulty. Columns r_0 to r_{q-1} denote residuals generated through equation 1.1 [69]. Applying Granger Causality to the elements of Table 11 yields a set of tuples with variable length of the form $\{(c_0, c_1, \dots, c_i, r_j), \dots\}$, where $r_j \in \mathcal{R}$ and $c_i \in F$.

Algorithm 5 shows how DDGD creates propositional logic diagnosis rules from process data. We will first describe the algorithm briefly and then discuss each of the functions in detail. Input to the algorithm is the time series T , the names \tilde{T} , the lag, and parameters (thresholds) τ . The lag defines how far into the past Granger Causality will attempt to find dependencies. Lines 3 and 4 extract the relevant components according to one row ϵ of the time-series T , and create Boolean residuals, respectively. Then F is merged with the residuals \mathcal{R} . The multivariate Granger Causality test itself returns a matrix with p-values ρ , which are sorted such that only those dependencies are selected, which are likely to cause changes in some component c_i according to threshold τ_p . The p-values denote, whether the obtained Granger-causal dependency is actually statistically significant. At the end, a set of tuples P (the Granger causal dependencies) is returned.

t	c_0	c_1	\dots	c_{n-1}	r_0	r_1	\dots	r_{q-1}
o	T	T		T	T	T		T
i	\perp	T		T	T	\perp		T
$i+1$	\perp	T		T	T	\perp		T
k	T	T		T	T	T		T

Table 11.: Expert defined components $c \in \tilde{F}$ related to Boolean residuals $r \in \tilde{\mathcal{R}}$. $n, i, k, q \in \mathbb{N}$. In practice, residuals are real-valued and need to be discretised first.

`extractComps()`:

Algorithm 5: DDGD: Data-Driven Granger diagnosis

Data: \tilde{T}, T, τ, lag
Result: P

```

1 foreach  $\epsilon \in T$  ; // For each row in T
2 do
3    $F \leftarrow extractComps(\epsilon, \tilde{T})$ ; // Def.25
4    $\mathcal{R} \leftarrow makeResiduals(\epsilon, \tau)$  ; // Eq.1.1
5    $\zeta \leftarrow F \cup \mathcal{R}$ ;
6    $\rho \leftarrow granger(\zeta, lag)$ ;
7    $P \leftarrow \emptyset$ ;
8   foreach  $c_i, elem \in \rho$  do
9     // Over all rows and columns
10    if  $elem > \tau_p \in \tau$  then
11    |  $P \leftarrow P \cup elem$ ;
12 return( $P$ );

```

Function `extractComps()`: $(\epsilon, \tilde{T}) \rightarrow F$ with ϵ being a tuple of expert-defined component health states from time series T at a single point in time and \tilde{T} being the set of component names, analyses which components are relevant at the time of the signal in tuple ϵ . In general, the function puts exactly those data points into F that belong to components c_0, c_1 , etc, where the health state has some variance. Please note that input to the algorithm is for practical reasons a single, multivariate time-series T . As a result, the component health states within the time series need to be extracted.

`makeResiduals()`:

Function `makeResiduals()`: $(\epsilon, \tau) \rightarrow \mathcal{R}$ with ϵ being a tuple of component health states from time series T at a single point in time and τ being a set of parameter (threshold) values. Similar to function `extractComps()`, the function puts exactly those data points into F that belong to residuals r_0, r_1 , etc. Since T usually contains real valued residuals, the function internally needs to discretise these into Boolean ones. The discretisation into Boolean values is then done through some function of the form of `generateSymptoms()`: $\mathbb{R}^n \rightarrow \{\top, \perp\}^*$ with $n \in \mathbb{N}$, that was introduced as part of algorithm HySD in Chapter 2.

`granger()`:

The function `granger()`: $(\zeta, lag) \rightarrow \rho$, with ζ being a tuple of component health states and residuals, $lag \in \mathbb{N}$ determines how far into the past Granger's algorithm looks, and ρ being a set of tuples. The parameter lag is either defined through by experts or can in some cases be determined through statistical methods such as the Akaike Information Criterion [225]. The output of the function is a set of tuples, whose rightmost element denotes the residual and the elements on the left of it denote the dependend (Granger-causal) components (c_i, c_j, \dots, r_k) , with $i, j, k \in \mathbb{N}$.

4.2. Example: DDGD

Table 12 presents some example data in form of a time series. To illustrate, we assume this is from the four-tank system in the running example. The table contains Boolean component health states and real valued residuals.

t	c_0	c_1	\dots	c_{n-1}	r_0	r_1	\dots	r_{q-1}
o	\perp	\top		\perp	0	1		20
k	\perp	\perp		\perp	0	1		20

Table 12.: Expert defined components $c \in \tilde{F}$ related to residuals $r \in \tilde{R}$. $n, k, q \in \mathbb{N}$. Residuals are real-valued.

Applying function `makeResiduals()` to Table 12 with some threshold, leads to the discretised residuals in Table Table 13.

t	c_0	c_1	\dots	c_{n-1}	r_0	r_1	\dots	r_{q-1}
o	\perp	\top		\perp	\perp	\perp		\top
k	\perp	\perp		\perp	\perp	\perp		\top

Table 13.: Expert defined components $c \in \tilde{F}$ related to Boolean residuals $r \in \tilde{R}$. $n, k, q \in \mathbb{N}$. Residuals are discretised.

Running algorithm DDGD then outputs the following tuples that associate residual values to components.

$$\begin{aligned}
 & (valve3, r_0) \\
 & (valve6, r_1) \\
 & (valve4, pipe4, r_2) \\
 & (tank2, r_3) \\
 & (valve6, r_4) \\
 & (valve4, pipe4, r_5) \\
 & (valve4, pipe4, r_6) \\
 & \dots
 \end{aligned} \tag{4.1}$$

Please note that the presented tuples are only a subset of the tuples that were detected for the four-tank system. Given the tuples, it is then possible to use algorithm DDA-IM from Chapter 3. In particular, the residuals would be converted to some propositional logic expressions of the form

$$\begin{aligned}
r_0 &\Leftrightarrow \top \\
r_1 &\Leftrightarrow \top \\
r_2 &\Leftrightarrow \top \\
r_3 &\Leftrightarrow \top \\
r_4 &\Leftrightarrow \top \\
r_5 &\Leftrightarrow \top \\
r_6 &\Leftrightarrow \top
\end{aligned} \tag{4.2}$$

Then, DDA-IM can be used for diagnosis by assuming Φ to be grounded with an assumed random health assignment

$$\begin{aligned}
&valve3 \rightarrow r_0 \\
&valve6 \rightarrow r_1 \\
&valve4 \wedge pipe4 \rightarrow r_2 \\
&tank2 \rightarrow r_3 \\
&valve6 \rightarrow r_4 \\
&valve4 \wedge pipe4 \rightarrow r_5 \\
&valve4 \wedge pipe4 \rightarrow r_6 \\
&r_0 \Leftrightarrow \top \\
&r_1 \Leftrightarrow \perp \\
&r_2 \Leftrightarrow \top \\
&r_3 \Leftrightarrow \top \\
&r_4 \Leftrightarrow \perp \\
&r_5 \Leftrightarrow \top \\
&r_6 \Leftrightarrow \top
\end{aligned} \tag{4.3}$$

Given the above system description Φ it is then possible to use algorithm DDA-IM with a call to `HySD_simple()` (Chapter 2) to compute the set of diagnoses $\{\{valve6\}\}$. The resulting set of diagnosis is in this case also the minimum cardinality diagnosis.

5. A Novel Theory of Fault Propagation

This chapter describes a novel theory that explains how to create expert-defined system descriptions analytically, by analysing fault propagation in cyber-physical systems. It also provides the theory to explain the fundamental difference between approaches such as the one in algorithms DDGD and DDRC. In particular, it is proved how approaches using residuals are superior to approaches using sensor values alone. Using this theory, especially the newly introduced structure patterns and their governing equations, can be used to implement the residual function $h()$ (Eq.: 1.1) in the previous chapters 2, 3, and 4 in Part III, as well as the connection and component models. In relation to the problem description in Chapter 1.1 in Part I, this section explains how some function $diag_cont()$ needs to adapt the real valued sensor data for Boolean diagnosis algorithms and how connection and component models can be created analytically. Using the minimal structure patterns introduced here, simplifies creating the connection and component models, since all interactions between different components can be analysed individually.

To automatically diagnose cyber-physical systems, a diagnosis algorithm requires a model predicting the system's normal behaviour: the weak-fault model. To make these predictions, the model must assume, or be provided with, an initial state of the system. Then, from the initial state onwards, the model makes forecasts about the system's future values. But to perform the predictions and forecasts, the model must have some knowledge about the components and their connections which describe all inputs for each component and their associated outputs. Calculating how the output of one component is processed by the next component is called propagation (fault propagation, if the propagated values occur in an abnormal state). Analyzing how a system behaves in the presence of faults requires a theory about how faults propagate between the system's different parts. For this, a formalisation of temporal causal behaviour is presented and two methods are introduced how fault symptoms can be analysed (Contribution 2):

- 1) The fault magnitudes such as a pressure readings can be processed as is. This brings the advantage that different value ranges can be evaluated. For example, it can be calculated whether some pressure is sufficient to boil water at 90 degrees Celsius. In the following, we will call this *signal-based* or *solution 1*. This is the theory behind algorithm DDRC.
- 2) To perform fault diagnosis, however, instead of the actual sensor values we are only interested in whether or not some signal is within normal operating conditions. This chapter shows how and why the calculation of such residual values, indicating the presence of faults (i.e. to perform the discretisation into normal and anomalous behaviour), is more efficient than using absolute fault magnitudes. In the following we will call this *residual-based* or *solution 2*. This is the theory behind algorithm DDGD.

The temporal causal behaviour in cyber-physical systems is interpreted in the sense introduced by Pearl and Halpern [54, 97] by their assumption that causal influences are modelled through structural equations. As such, the theory presented here is also similar to the Qualitative Physics theory of De Kleer [32], which refers to a wave-like (adapted from

Feynman et al.[226]) propagation from one component to the next, with components being either *visited*, *processed*, or *queued*. The system's state is only evaluated once the propagation in its current magnitude has been processed in all components. With the understanding that the system's operation is subdivided into very small timescales [161].

5.1. Component and Connection Models in Practice

Of course, each component's behaviour can be modelled with different levels of abstraction. The definitions in this chapter can be used to calculate the dependencies in cyber-physical systems as well as to determine fault propagation and fault amplitudes. But sometimes much simpler models are sufficient. In earlier chapters these were mostly illustrated through thresholds which are evaluated according to being *ok* or *not okay*. Alternatively, component models can be substituted through the classification output of neural networks or statistical methods. Thus, it is important to note that no one true formulation for component models exists.

So what is the aim of creating the connection and component models in this chapter? The connection model (Def.: 23), at least when it is created analytically with the tools in this chapter, describes the relation between components. It is thus used by Algorithm HySD to create SMT logic stumps of conjunctions between components (`makeSMT_CON()` and `makeSMT_CM()`) and the implications. In other words, it describes which components are connected to which sensor and therefore to an observation. The component models are used to calculate the expected behaviour of some component. So for some Boolean circuit this would translate into a truth table. In the process industry, these are usually realised through differential equations. The output of the component models is used to calculate the expected state for each component. Here, two instances of a component model need to be differentiated: i) The component model for a sensor outputs the prediction, which is used by Equation 1.1 (Def.: 34) to compare against the actual value (the measurement). This is the main goal for the component model. ii) Component models of components that are not sensors. The component models are used for propagating values from system inputs to sensor components. But they do not appear within the SMT logic expressions.

5.2. Definitions

According to Definitions 21 and 23, a cyber-physical system can be combined into higher-level structures made up of multiple components. The basic structures (i.e resulting subgraphs) that can be deduced from these definitions are discussed here. We call the resulting subgraphs structure patterns (see Figure 10). The structure patterns help to hierarchically analyse fault propagation by looking at propagations between the smallest numbers of components and from these infer the fault propagation in larger systems. This corresponds to an extension of the class-wide assumptions and the principle of locality introduced by De Kleer [32] to structures with more than one component.

Overall, six different structure patterns were defined. These are a novel formulation to define the minimal dependencies within cyber-physical systems. First we introduce the structure patterns as part of a connection model (Def.: 23). Then we will show how output models can be defined from these structure patterns.

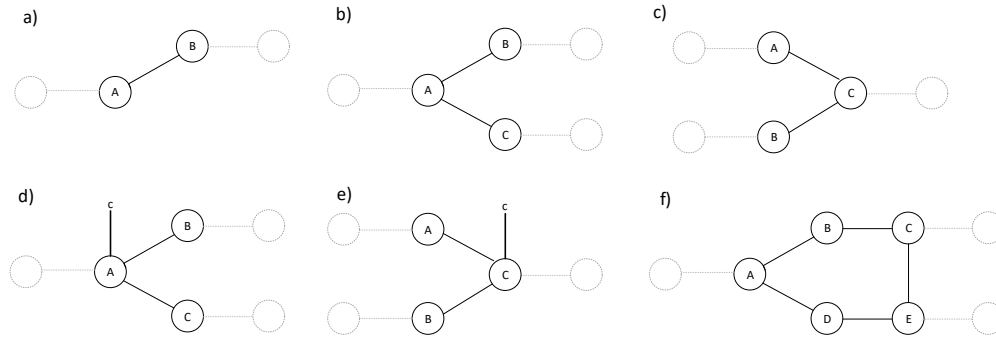


Figure 10.: Different fault propagation structure patterns

- (a) **Sequence:** Two nodes $v_i, v_j \in \{\mathcal{V}_i \cup \mathcal{V}_o\}$ connected by a single (directed) edge $e(v_i, v_j)$ with $e \in E, v \in V$. $i, j \in \mathbb{N}$ are the input and output index. V, E are the vertices and edges part of Definition 23. The structure pattern is denoted as $G_S \in CON$ (Figure 10-a)
- (b) **Junction:** A simultaneous connection from one node to two or more nodes $\{e_0(v_i, v_j), \dots, e_n(v_i, v_k)\}$ with $e \in E, v \in V, v_i \notin v_j \dots v_k$. $v_i, v_j \in \{\mathcal{V}_i \cup \mathcal{V}_o\}$. $i, j, k \in \mathbb{N}$ are the input and output index, $n \in \mathbb{N}$ is the edge index. V, E are the vertices and edges of Definition 23. The structure pattern is denoted as $G_J \in CON$ (Figure 10-b)
- (c) **Reverse Junction:** A simultaneous connection from many nodes to one node $\{e_0(v_j, v_i), \dots, e_n(v_k, v_i)\}$ with $e \in E, v \in V, v_i \notin v_j \dots v_k$. $v_i, v_j \in \{\mathcal{V}_i \cup \mathcal{V}_o\}$. $i, j, k \in \mathbb{N}$ are the input and output indices, $n \in \mathbb{N}$ is the edge index. V, E are the vertices and edges of Definition 23. The structure pattern is denoted as $G_R \in CON$ (Figure 10-c)
- (d) **Multiplexer:** A controlled connection from one node to many nodes. The edge to be taken is selected through a control signal $\{e_0(c, v_i, v_j), \dots, e_n(c, v_i, v_k)\}$ with $e \in E, v \in V, v_i \notin v_j \dots v_k$ and the edge being activated if $e_j : j = c$. $v_i, v_j \in \{\mathcal{V}_i \cup \mathcal{V}_o\}$. $i, j, k \in \mathbb{N}$ are the input and output index, $n \in \mathbb{N}$ is the edge index, and $c \in \{0, 1\}^n$ is the control input index. V, E are the vertices and edges of Definition 23. The structure pattern is denoted as $G_M \in CON$ (Figure 10-d)
- (e) **Demultiplexer:** A controlled connection from many nodes to one node. The edge to be taken is selected through a control signal $\{e_0(c, v_j, v_i), \dots, e_n(c, v_k, v_i)\}$ with $e \in E, v \in V, v_i \notin v_j \dots v_k$ and the edge being activated if $e_j : j = c$. $v_i, v_j \in \{\mathcal{V}_i \cup \mathcal{V}_o\}$. $i, j, k \in \mathbb{N}$ are the input and output index, $n \in \mathbb{N}$ is the edge index, and $c \in \{0, 1\}^n$ is the control input index. V, E are the vertices and edges of Definition 23. The structure pattern is denoted as $G_D \in CON$ (Figure 10-e)
- (f) **Cycle:** A series of connections $\{e_0(v_0, v_1), e_1(v_1, v_2), \dots, e_i(v_i, v_0)\}$ with $e \in E$ and $v \in V$. $v_i, v_j \in \{\mathcal{V}_i \cup \mathcal{V}_o\}$. $i, j \in \mathbb{N}$ are the input and output index. V, E are the vertices and edges of Definition 23. The structure pattern is denoted as $G_C \in CON$ (Figure 10-f)

These structure patterns are used as building blocks to model systems of arbitrary complexity. In the following, we will analyse the three basic blocks Sequence, Junction, Multiplexer, and in addition the structure pattern Cycle. Demultiplexers and Reverse Junctions can be inferred accordingly.

For all subsystem structure patterns and for clarity and without loss of generality, we assume a single-fault scenario, which propagates fault manifestations through the rest of

the system. Sensor readings are used to monitor in-going and out-going connections of a component. A single sensor reading can be modelled as $y = x_i f_{m_k}$ for some measurable value x_i , index $k \in \mathbb{N}$, and some multiplicative error term f_m . Additive faults f_a , such as noise, are modelled as $y = x_i + f_{a_k}$ [136].

The next sections will first analyse the realisation of the different structure patterns, according to their output model. With slight abuse of notation we will use the cup-operator $x_0 \cup \mathbf{x}$ to denote the set union of a vector with a variable. This allows us to define an operation to include a variable in an already existing vector.

5.3. Structure Patterns using Signal Values

We will first discuss how to model structure patterns for absolute signal values alone. Later, we will analyse how these structure patterns change, when using residual values.

5.3.1. Structure Pattern - Sequence

For a single sequence G_S , each node has some output model $\mathbf{y} = n(\mathbf{x}, \mathbf{u})$ (Def. 24). Therefore,

$$\mathbf{y}_i = n(\mathbf{x}_i, \mathbf{y}_{i-1} \cup \mathbf{u}_i) \quad (5.1)$$

with the edge $e(v_0, v_1)$ of G_S connecting the output of component model $y_{i-1} \in \mathbb{R}^y$ to the inputs of component model $y_i \in \mathbb{R}^y$. Note that \mathbf{u}_i are the new inputs to the component, which are not part of another component's output. In the following, we will omit repeatedly stating the output model and instead list only the fault augmented model.

The fault-augmented model (see Definition 25) for faults for a sequence of two components can be stated as

$$\mathbf{y}_i = n(\mathbf{x}_i, \mathbf{y}_{i-1} \cup \mathbf{u}_i) f_{m_i} + f_{a_i} \quad (5.2)$$

The fault manifested within output $\mathbf{y}_{i-1} \in \mathbb{R}^y$ will be processed and possibly amplified or dampened and is captured in the form of f_{m_i} and f_{a_i} (Def. 25) within the next component. The sequence G_S is therefore a chain of functions modelling consecutive fault manifestations.

5.3.2. Structure Pattern - Junction

A junction G_J is a graph structure composed of two sequence structures attached to a common node. A junction propagates faults from one node to several subsequent nodes. The fault augmented model (Def. 25) of a junction can be stated as

$$\mathbf{y}_i = n(\mathbf{x}_i, \mathbf{y}_{i-1} \cup \mathbf{u}_i) f_{m_i} + f_{a_i} \quad (5.3)$$

Here, the faults from the root component propagate to all subsequent components equally and at the same time.

5.3.3. Structure Pattern - Multiplexer

Multiplexers G_M are junctions controlled by a signal $c \in \{0, 1\}^n$. Through c , one path from the root component to another component can be selected. By interpreting c as a binary vector instead of a scalar value it is possible to select multiple paths from the root component. In this case, the edge activation is changed to $e_j : j \in \{c_1, \dots, c_n\}$ with $l, n \in \mathbb{N}$. The possibility of controlled edges results in more complex fault-augmented models. The augmented fault model is defined as

$$\mathbf{y}_i = n(\mathbf{x}_i, \{\mathbf{y}_{i-1} \cup \mathbf{u}_i \text{ if } c_i, \text{ else } \mathbf{u}_i\})f_{m_i} + f_{a_i} \quad (5.4)$$

Please note that the input is now defined as a piece-wise function, where the root component's output is only used when the component is selected. In all other cases, the component is treated to be independent of its previous components.

5.3.4. Structure Pattern - Cycle

A cycle G_C within components of a cyber-physical system creates feedback loops. These can amplify or dampen a fault magnitude [227]. In many cyber-physical systems, feedback loops are a common occurrence. For example, many control theoretic methods require feedback, or sometimes, such as in the Tennessee Eastman Process or rubber compounding processes, production material is fed back into the input.

To deal with this behaviour, we augment the fault model to

$$\mathbf{y} = a_i n_i(\mathbf{x}_i, \mathbf{u}_i) f_m(t-1) + f_a(t-1) \quad (5.5)$$

with $t \in \mathbb{N}$, component index $i \in \mathbb{N}$, $a_i \in \mathbb{R}$ being the gain term, and $f_m(t-1)$ and $f_a(t-1)$ as a recursive definition of the previous fault magnitudes. Here we need to keep track of time t , as a cycle will change its behaviour according to previous values. As time is analysed in discrete steps we denote with $t-1$, some previous time-step. For non-linear (or unknown) relations between the amplification and fault magnitudes the previous equation is rewritten to

$$\mathbf{y} = n(t, \mathbf{x}_i, \mathbf{u}_i, f_i(t-1)) \quad (5.6)$$

where the factor a_i becomes implicit and the fault terms f_m and f_a are subsumed under term $f_i : \mathbb{R}^{x \times u} \rightarrow \mathbb{R}$.

5.4. Discussion of signal-based fault propagation

It is possible to specify a component within any hierarchy as

$$\mathbf{y}_i = n(\mathbf{x}_i, \mathbf{u}_i \cup \boldsymbol{\beta}^T \mathbf{y}_{i-1, \text{in}}) f_m(\cdot) + f_a(\cdot) \quad (5.7)$$

where $n(\cdot) : \mathbb{R}^{x \times u \times y} \rightarrow \mathbb{R}^y$ is a component model function, $\mathbf{y}_{i-1, \text{in}} \subset \mathbb{R}^y$ is the set of all the outputs from other components that lead to this component, $\boldsymbol{\beta} \in \mathbb{R}$ is some amplification factor, $f_m(\cdot) : \mathbb{R}^{f_m} \rightarrow \mathbb{R}$ is some multiplicative fault model, and $f_a(\cdot) : \mathbb{R}^{f_a} \rightarrow \mathbb{R}$ is some additive fault model (Def.: 25). Inherent in these functions is a recursive calculation of

the fault manifestations (as was shown in algorithm DDRC). Since each component reacts to faults differently, the fault amplitude may be severely amplified or dampened. This becomes even more evident when looking at cycles.

With the gain factor $a_i \in \mathbb{R}$ of a cycle of components realistically being $\neq 1$, the feedback during faulty operations is going to approach either infinity $\lim_{t \rightarrow \infty} f_m = \infty$ or zero $\lim_{t \rightarrow \infty} f_m = 0$. The same is true for additive faults f_a .

Theorem 1. *For multiplicative faults a feedback cycle will always approach as limit either infinity or converge to 0*

Proof. Sketch. We use the idea of induction.

Case $|a| > 1$: Suppose a multiplicative fault model and some component output model i exist $\mathbf{y} = a_i n_i(\mathbf{x}_i, \mathbf{u}_i) f_{m_i}(t - 1)$. Then, the fault model of a connected component $j \in \mathbb{N}$, within the cycle, will inherently depend on the fault model of component $i \in \mathbb{N}$, propagated through its output variable \mathbf{y}_i . This contains the effects of the gain term a_i . Each connected component j that takes \mathbf{y}_i as its input will multiply its own gain term a_j in each fault model.

Case $|a| < 1$: Similar to case $|a| > 1$. The difference is that through propagation the fault effects will approach 0.

□

This approaching of limits makes it hard to track fault propagation and therefore to diagnose cyber-physical systems through these sensor observations alone. To evaluate whether a certain sensor reading is valid is a binary operation. The recursive amplification and dampening of fault amplitudes is consequently undesirable for fault diagnosis. The dampening of some oscillation caused by a fault was also found, without proof, by De Kleer [32].

Corollary 1. *The fault manifestations $f_{m_i}(\cdot)$ and $f_{a_i}(\cdot)$ cannot be calculated analytically*

Realistically, functions f_{m_i} and f_{a_i} calculate the fault amplitude from a subset of real-world properties. For example, f_{m_i} and f_{a_i} might take pressure, temperature, and the state of valves as their input to calculate the fault magnitude of a biological reactor's outflow. If the pressure is too high, the fault magnitude changes such that the flow towards the subsequent components is altered (flow increases). However, in the presence of faults, system behaviour is usually unknown, due to little understood influences of exogenous input variables (for example, daylight hitting the reactor casing, unexpectedly). In the best case, functions f_{m_i} and f_{a_i} can only be roughly approximated, although the degree of approximation heavily depends on the use-case. Therefore, it is difficult to find appropriate and general values for parameters, such as the thresholds τ .

5.5. Structure Patterns using Residual Values

Now, we will analyse the structure patterns and describe, which changes are made to incorporate residuals. Then, these findings are compared to the signal-based approach.

Based on the fault-augmented model in Definition 25, a residual of the form of Equation 1.1 is defined as

$$r_i = h(\mathbf{x}_i, \mathbf{u}_i \cup \mathbf{y}_{i-1}, \mathbf{y}_i f_{m_i} + f_{a_i}, \lambda_i) \quad (5.8)$$

with state \mathbf{x} , input \mathbf{u} , previous component's output \mathbf{y}_{i-1} , output \mathbf{y} with fault-augmented models for this component, and parameter vector λ . Therefore, we assume that faults effect only the output of a component $y_i \in \mathbb{R}^y$. $r_i \in \mathbb{R}$ is the calculated residual value according to Equation 1.1.

Overall, a residual is an implementation of an observer pattern [228] which *observes* a component's inputs, outputs, and states. In the case of this thesis, the residual is generated through an observer of the form depicted in Figure 11. This defines a residual r_i computed through function $h(\cdot)$ (Eq. 1.1) and discretised into Boolean values through function $d(\cdot)$ (Def. 36). Note that the functions $f_{m_i}(\cdot)$ and $f_{a_i}(\cdot)$ from the fault-augmented model (Def. 25) do not need to be stated explicitly anymore, but are assumed to be represented through the output models of previous components' y_{i-1} . Instead, residuals are defined such that they are unique to each component. Thus, only the formulation of Equation 5.8 is used to model residuals for all introduced structure patterns. Compared to the fault propagation with signal values, it is possible to use only a single formulation for all structure patterns. As a result, some residual r_i does not have any influence on other components. This causally decouples the residual amplitude between components. For structure patterns such as multiplexers, the choice of which connections are active has no implications to the calculation of residuals. However, the strongest benefits are evident in the cycle structure pattern. Here, the recursive definition of faults and their propagations is removed. Instead, each component within the cycle is analysed as an independent junction, multiplexer, or sequence, depending on its function. The main benefit of using residuals is therefore, that they decouple each component from its environment and thus remove the above introduced recursiveness, error amplification, and dampening, respectively.

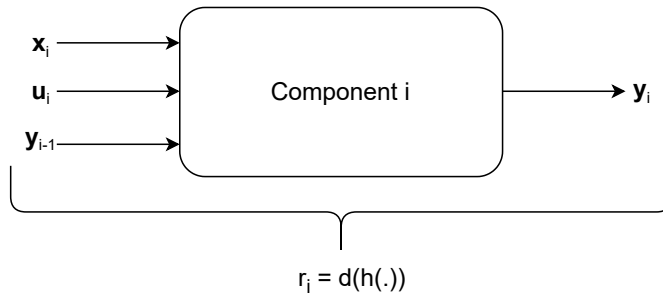


Figure 11.: Generation of a residual r_i through function $h(\cdot)$ (Eq. 1.1), discretisation function $d(\cdot)$ (Def. 36) on states \mathbf{x} , inputs \mathbf{u} , previous component's outputs \mathbf{y}_{i-1} , and outputs \mathbf{y} .

5.6. Discussion of residual-based fault propagation

So far we have shown the disadvantages of propagating fault magnitudes through a system (solution 1) and we have shown how the calculation of residuals can make the tracking of faults easier (solution 2). Table 14 provides a brief summary. In solution 1, fault propagation **signal-based, without the calculation of residuals** has a number of strong requirements:

1. All components and their exact behaviour must be known in normal and abnormal conditions.
2. The connections between all components must be known.
3. All possible influences, fault modes, and their respective fault magnitudes must be known (Definition 25).
4. To perform diagnosis, a robust method must exist that evaluates fault magnitudes and infers the corresponding faults.

Especially the first and third requirement often cannot be satisfied. While models of the normal behaviour may exist, the nature of faults is often unknown and unpredictable and thus cannot be modelled exhaustively. For diagnosis, thresholds must be known for all sensor values which need to be diagnosed. For example, in some automated cooking device different thresholds for the boiling point would need to be saved depending on the altitude due to the ambient pressure. To know and to model such thresholds for industrial cyber-physical systems can be prohibitively expensive. Algorithm DDGD was developed to overcome this drawback and learn system descriptions automatically.

The diagnosis **with calculation of residuals using component health states** can bring some advantages:

1. Only normal behaviour of components must be known, but it may be approximated.
2. The connections between all components must be known. This is no different than without the calculation of residuals.
3. Residuals need to be designed to track the normal behaviour and indicate when faults occur.
4. Diagnosis can be done through tracking the residual values with thresholds.

While the calculation of residuals for diagnosis still requires a connection model, the component models and residual generating functions are far easier to create [152]. For each measurement variable $h(\cdot)$ can be determined in multiple ways: analytical, data-driven, or through naive thresholds (as seen above). For well understood components, where analytical equations are known, these can be integrated into $h(\cdot)$ by using, for example, state observers. Often, however, these functions can only be learned and approximated [191]. Instead of accurate and elaborate quantitative models, normal behaviour can be learned with machine learning algorithms such as artificial neural networks or support vector machines. Equation 1.1 makes it convenient to create simple statistical or threshold-based models, which can optimally be extracted from expert knowledge. Additionally, the evaluation of the residuals through thresholds is an implicit binary discretisation into some signal being ok (its residual being equal to 0) or not ok (it's residual being different from 0). Such a binary view is helpful for using traditional consistency-based diagnosis algorithms [19, 20, 17].

Category	Signal-based	Residual-based
Approach	Analysing fault magnitudes	Analysing symptoms
Fault manifestation	Absolute sensor readings	Binary discretisation into normal and anomalous values
Practicality	Fault models need to calculate the error functions $f_a(\cdot)$ and $f_m(\cdot)$	Needs the availability of some residual generating function $h(\cdot)$
Error propagation	Influences amplified or dampened according to component	No amplification or dampening, but residual contains less information

Table 14.: Comparison between the calculation of absolute errors and residuals

5.7. Example: Causal Models

The structure patterns and their uses for signal-based and residual-based fault propagation shall be illustrated using the one-tank system of the running example. During fault-free operation each component within the one-tank system is governed by a set of known equations. We measure the throughput through a pipe with a sensor after each valve, denoted by $fl_i \in \mathbb{R}$ and we measure the water level in the tank through a sensor denoted by $l_i \in \mathbb{R}$. Each value is calculated by some suitable formula $n_i(\cdot)$. It is therefore possible to state the model of normal operating behaviour and augment the formulae by introducing fault models $f_m(\Xi_m)$ and $f_a(\Xi_a)$.

$$\begin{aligned}
 fl_0 &= n_0(\mathbf{x}_0, \mathbf{u}_0) f_{m_0}(\Xi_{m,0}) + f_{a_0}(\Xi_{a,0}) \\
 l_0 &= n_1(\mathbf{x}_1, \mathbf{u}_1 \cup fl_0) f_{m_1}(\Xi_{m,1}) + f_{a_1}(\Xi_{a,1}) \\
 fl_1 &= n_2(\mathbf{x}_2, \mathbf{u}_2 \cup l_0) f_{m_2}(\Xi_{m,2}) + f_{a_2}(\Xi_{a,2})
 \end{aligned} \tag{5.9}$$

Supposing f_{m_0} is the symptom of the fault in valve 1. In a trivial case, using a binary valve, the first equation would evaluate to 0 with $f_{m_0} = 0$. With fault propagation this fault will over time manifest itself in most other sensor values as well. In this example, at first the level in the tank might fall, since f_{m_0} influences fl_0 and thus changes the input to $n_1(\cdot)$. Due to a falling tank level, the flow through valve v_1 will decrease and so forth. Each fault manifestation directly causally influences the measurable values of other components.

Calculating residuals makes tracking of fault propagation easier. Simplifying Equation 1.1 we define

$$h(\cdot) : |\hat{\alpha}(t) - m(t)| < \tau \tag{5.10}$$

where $\hat{\alpha} \in \mathbb{R}$ is the model's prediction at time t , τ is a threshold, and $m \in \mathbb{R}$ is the measurement.

Regardless of the actual implementation of $h(\cdot)$, the above one-tank model could be

modelled as

$$\begin{aligned}
 r_0^{fl} &= h(\mathbf{x}_0, \mathbf{u}_0, fl_0 f_{m_0}(\Xi_{m,0}) + f_{a_0}(\Xi_{a,0}), \boldsymbol{\tau}_0) \\
 r_0^l &= h(\mathbf{x}_1, r_0^{fl} \cup \mathbf{u}_1, l_0 f_{m_1}(\Xi_{m,1}) + f_{a_1}(\Xi_{a,1}), \boldsymbol{\tau}_1) \\
 r_1^{fl} &= h(\mathbf{x}_2, l_0 \cup \mathbf{u}_2, r_0^l f_{m_2}(\Xi_{m,2}) + f_{a_2}(\Xi_{a,2}), \boldsymbol{\tau}_2)
 \end{aligned} \tag{5.11}$$

where $h(\cdot) \in \mathcal{R}$, with \mathcal{R} being a set of possible residual generating algorithms, and thresholds $\boldsymbol{\tau} \in \mathbb{R}^\tau$

This binary interpretation facilitates the usage of consistency-based diagnosis on the basis of residual-based fault propagation, as the residuals r can be easily evaluated through learned or expert-defined thresholds.

Part IV.
Evaluation

1. Overview

While the algorithms introduced above are applicable to many kinds of cyber-physical systems, this part presents their use in the context of cyber-physical production systems. These make up a significant part of modern machinery [7]. The experiments also show the limitations of the respective algorithms. In particular, within the evaluation four different parts need to be shown: i) How the diagnosis algorithm HySD is performing given data from industrial use-cases. ii) Showing the limitations and capabilities of algorithms DDA-IM, DDRC, and DDGD by presenting the generated system descriptions and showing the diagnosis accuracy, given those system descriptions. iii) Analysing how well the generated system descriptions can be used for diagnosis using algorithm CheckDiag. iv) Analysing the computational complexity and the running time of the presented algorithms.

The evaluation of algorithm HySD proves its ability to answer research questions RQ₁ and RQ₂, since the algorithm provides a usable methodology for diagnosing cyber-physical systems (RQ₁). At the same time, HySD relies on a theory of fault propagation for its system description and thus answers RQ₂. RQ₃ is mainly answered by the algorithms DDRC and DDGD.

In what follows we will first briefly introduce the evaluation methodology. Then empirical results are presented with six different sets of experiments. These evaluate HySD with expert-generated system descriptions, and in conjunction with DDGD, with automatically generated system descriptions. Algorithm DDRC is evaluated qualitatively as its output always requires interpretation and translation from signals to actual components. Further, the generated system descriptions of both algorithms DDRC and DDGD are validated through the novel evaluation algorithm CheckDiag. In the theoretical analysis, the running time and computational complexity are analysed for all algorithms.

2. Methodology

The goal of the evaluation is to prove whether the algorithms introduced in the last chapter are correct, whether they are applicable to real world data, and to find out which limitations need to be accounted for [229].

Looking at algorithm HySD, it needs to be shown that, given an expert generated system description SD, the algorithm is able to integrate residuals \mathcal{R} with the system description, and pass the resulting set of SMT- \mathcal{LRA} expressions to some diagnosis algorithm (see Section 1.7 Part I). For this, HySD needs to convert an expert-defined connection model with respective component models into a logical model in SMT- \mathcal{LRA} . Then it needs to associate observations with the logical expressions, and perform a consistency-check in order to compute diagnoses. Quite the same is true for algorithm HySD_simple. The only difference to HySD is that HySD_simple assumes that a system description in SMT- \mathcal{LRA} , or even propositional logic, is already given. Therefore, HySD_simple contains only a subset of the functionality of algorithm HySD. Nevertheless, HySD_simple is a stand-alone diagnosis algorithm, once a system description is given. Although for the methods in this thesis HySD_simple is usually called from within algorithm DDA-IM, where DDA-IM is a convenient way to convert the output from algorithms DDGD and DDRC into the system description for HySD_simple. Every time a diagnosis is executed in this chapter, we use HySD, when only expert-generated system descriptions are available and we use HySD_simple called from within DDA-IM, when system descriptions are generated automatically.

The algorithms DDGD and DDRC to generate system descriptions are evaluated by checking whether their output leads to correct diagnoses. This is done through the algorithm CheckDiag that will be introduced in Chapter 3.3. Further, their generated system descriptions are used as the input to the two algorithms DDA-IM and HySD_simple. Using those system descriptions, observations are generated using two methods: a) A random number of observations are assigned *faulty*. This leads to an inconsistent system description, which is then used for diagnosis. Here, the output of the diagnosis is only checked, whether it is reasonable. b) Observations are used from the real processes that were also used to evaluate HySD on its own. This shows how diagnosis performance compares to diagnosis using expert-generated system descriptions.

We will now provide an illustration how the concrete evaluation for algorithm HySD is done on the one-tank running example (see Section 1.4 Part I). Especially, it shall be presented, how the evaluation is performed when residuals are included into expert-defined or generated system descriptions. Figure 12 (a) shows an expert defined system model using the structure pattern Sequence (Chapter 5 Part III). Within the figure, an input is connected to a sequence of components (valve 0, pipe 0, sensor 0, tank, pipe 1, valve 1, sensor 1), which then lead to an output. From the structure pattern Sequence, it is possible to create a series of output models of the form

$$\mathbf{y}_i = n(\mathbf{x}_i, \mathbf{y}_{i-1} \cup \mathbf{u}_i) \quad (2.1)$$

where \mathbf{y}_i is the i 'th component's output, \mathbf{u}_i the input, and \mathbf{x}_i the component's state. In the illustration, input and output would be flow values measured in meters per second. The tank has the water level as its state variable. Then, in Figure 12 (b) residuals can be generated on top of the sensors through the health functions

$$r_i = h(\mathbf{x}_i, \mathbf{y}'_{i-1} \cup \mathbf{u}_i, \mathbf{y}_i f_{m_i} + f_{a_i}, \boldsymbol{\tau}_i) \quad (2.2)$$

with specified fault models f_{m_i} and f_{a_i} (Definition 25) and thresholds $\boldsymbol{\tau}$. In Figure 12 (b) these are depicted in the boxes titled *Residual Generation*.

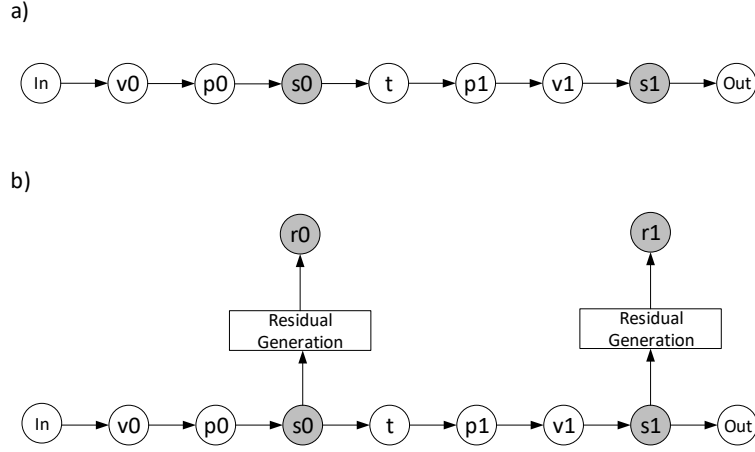


Figure 12.: Expert defined system model of the one-tank running example. The system is constructed using the Sequence structure pattern. Threshold-based residual generation may be integrated into SMT. t is the tank, p are pipes, v are valves, s are sensors, and In and Out are the system inputs and outputs, respectively.

Concretely, the system is characterised through its input flow, output flow, and through the two residuals r_0 and r_1 . The health function $h(\cdot)$ is implemented using simple expert-defined thresholds (Eq.: 1.1). Then it is possible to create system descriptions using the structure patterns and the residuals. This is done by looking at the residuals and tracing the dependency backwards until the next residual is encountered. In Figure 12, this would result in the SMT- \mathcal{LRA} expressions

$$\begin{aligned} v_0 \wedge p_0 &\rightarrow r_0 \leq \tau_0 \\ t \wedge p_1 \wedge v_1 &\rightarrow r_1 \leq \tau_1 \end{aligned} \quad (2.3)$$

Creating and assigning values to this system description is done through algorithm HySD and in the procedure outlined in Section 2.2 in Part III. For the evaluation in the next section using expert-defined system descriptions, we have assumed an expert-created system description of the form of φ_d (see Algorithm 1). This means, an expert has created a system description analytically in SMT- \mathcal{LRA} as shown in Figure 12.

Algorithms DDRC and DDGD are evaluated on their capability to generate useful expressions

such as those shown in Equation 2.3. This is evaluated by using their generated system descriptions as input to algorithm DDA-IM and using observations from real process data. If an accurate diagnosis can be computed, the generated system descriptions are deemed usable. Further, algorithm CheckDiag is employed to quantitatively evaluate the usefulness of the generated system descriptions through some heuristic.

All experiments were executed with Python 3.7 on a 64-bit Windows computer with 16GB of RAM and Intel i7-9750H processor.

3. Empirical Results

This chapter introduces different evaluation data-sets and the results obtained. The experiments are designed such that for each experiment either HySD (when using expert-defined system descriptions) or HySD_simple (when using generated system descriptions) is employed. Therefore, the overall goal of this chapter is to evaluate the performance of those two algorithms. However, when generated system descriptions are used with algorithms DDRC and DDGD, these algorithms are executed before HySD and HySD_simple. In those cases, successfully running HySD and HySD_simple means that they are based on a suitably-generated system description.

This chapter will first introduce the data-sets that were used for evaluation. Then, two dedicated experiments are carried out to show the performance of HySD using some expert-generated system descriptions. Afterwards, experiments are run with DDGD and DDRC, respectively. To show the performance of these algorithms they are run such that they generate a system description, which is then evaluated using HySD_simple. Please note the similarity of HySD and HySD_simple in that they both have in common the methods to generate symptoms, perform SAT solving, and hitting-set calculations. In the end of this chapter, Algorithm CheckDiag is presented, which is used to evaluate the generated system descriptions without performing a diagnosis. Overall, we show the functioning of the diagnosis algorithms, the performance of the system-description-generating algorithms in conjunction with diagnosis algorithms, and the quality of generated system descriptions. The data used for the evaluation is also found in the articles leading to this thesis [69, 76, 75, 70].

3.1. Description of data-sets

Two benchmarks, each with multiple tank systems, and five individual systems were used for the evaluation. In the following, each data-set will be briefly described. All of the systems are primarily multiple tank systems of different complexity, as it is quite common in the diagnosis community to evaluate algorithms on tank systems.

Benchmark of tank systems: The first benchmark data used for evaluation was extracted from systems S₁, S₃, S₄, and S₆ of the benchmark from Balzereit et al. [82]. The benchmark was created using the software OpenModelica 1.13 and was coauthored by the author for use in this thesis and the associated research articles. System S₁ is a one-tank system similar to the running example (Figure 2), where a pump pumps water through a valve into a tank and the tank drains through a second valve. In the three-tank system S₃, water is pumped into two parallel tanks (A and B). While tank A drains into the sink, another tank B is connected to A, which is then connected to a pump leading into the sink. System S₄ contains a tank which is connected to two further tanks and one bypass. All of these are connected to a fourth tank, which drains into the sink. System S₆ contains a tank and two

valves that each are used as the input for some type of fluid. The system works as a mixer, where the two fluids are mixed and then ejected. In the benchmark data, we have limited the evaluation to sensor values measuring flow and tank-level values. Internal variables and correlated features, such as derivatives, were removed. The simulation in OpenModelica was run over 500 time steps and faults were injected at time step 250. The faults injected in these data-sets can be configured within the OpenModelica 1.13 implementation [82] and include valve and pipe cloggings, tank leaks, and different pump speeds. For this thesis, each possible fault has been injected once.

Benchmark of bottling process: The benchmark with four subsystems developed by Ehrhardt et al. [83] describes an industrial bottling plant. The four modules are a mixer, a distill, a filter, and a bottling module. The modules may be used stand-alone or may be connected. For the evaluation in this thesis, it was assumed that the modules are stand-alone and effects from one module do not propagate into another module. The mixer perturbs three substances and stores the mixed products in a tank. The mixer is connected to a distill, which splits substances with different boiling temperatures. The filter subsystem removes residue from the fluid stream. As the last module, the bottling subsystem fills the produced substance into tiny glass bottles. For diagnosis, process data has been pre-processed to generate binary residuals [70] (variables r_i). The faults injected in these data-sets can again be configured within the OpenModelica 1.13 implementation [83] and include valve and pipe cloggings, and tank leaks. Each possible fault has been injected once.

Python Implementation of a four-tank system: The benchmark of the bottling process and the benchmark of the tank systems were both implemented in OpenModelica. Parallel to them, a first-principles implementation of a four tank system was developed in Python. Figure 3 back in the running example of Chapter 1.4 Part I shows the layout of the tank system used. Water flows into valve 0 and is stored in the tank. From the tank, the water is distributed into two additional tanks and flows through a bypass. The different water flows are aggregated into the last tank, which is connected to the output. It is assumed that the flow of water is maintained through external pumps, which induces a stable state. The system was programmed using first-principle differential equations and the formulation as a linear time invariant system with an integrated observer pattern. This made it possible to directly integrate HySD into the Python implementation to form an integrated diagnosis demonstrator. The injected faults are all pipe leakages either as single faults or double faults.

The Tennessee Eastman process: The Tennessee Eastman process (TE) [230] is a well-known evaluation system from the process industry [41, 42]. Within this thesis, two different data-sets of the Tennessee Eastman Process were used. One data-set is extracted from a simulation program of the original implementation from Downs et al. [230] and the other one is a public data set recently published by Manca et al. [231]. The difference between the two data-sets is that the former contains plain data from a simulation controlled through a command-line application and written in Fortran, and the latter contains specially annotated alarm data in the form of comma-separated-value files, which makes it easier for processing, especially in supervised applications. The implementation of Downs et al. contains 20 different injected faults (which they call process disturbances). However, the instrumentation of the simulated process is such that not all faults will be identified exactly. The same is true for the data set from Manca et al., which also omits some of the non-observable parameters. For example, within this evaluation only process disturbances

through IDVs 1, 6, 8, and 13-15 could be evaluated. Overall, the advantage of the Tennessee Eastman process is that it is very closely modelled after its real-world counterpart. As such, as in real industrial settings, not every component is equipped with a sensor for diagnosis. This makes it harder for diagnosis algorithms to find faults and also puts a higher emphasis on a correct system description. The injected faults are those associated to IDVs 1, 6, 8, and 13-15.

The following two data-sets were only used for the qualitative evaluation of algorithm DDRC.

Injection molding machine: The data-set from an injection molding machine describes the production of plastic casings for Raspberry PI systems. To use DDRC, we split the time-series data from the machine into the subsets S and M . In set S , we captured signals such as temperatures, cycle times, pressure values, and speed of various components. In set M , we represented measurements from an optical quality control system. The size of the data-sets was 11 signals for S and 10 signals for M . Faults were injected through randomly setting 20 % of the signals in set M to false, which corresponds to wrong size measurements of the produced parts in the real world.

Compounding process: The production of rubber products requires a batch-wise compounding process in which different ingredients are mixed according to a pre-defined recipe. The ingredients are brought into a mixer that stirs them into one unified mass for the output. For the experiment for algorithm DDRC, we used time-series data of single batches encompassing 6506 measurements of 87 signals. Of these, 35 were identified to be quality control signals from laboratory data, thus forming set M and the rest being contained in set S .

The last data-set was used only with algorithm CheckDiag

ISCAS-85 Benchmark: The ISCAS-85 benchmark [232] is a well-known standard in the field of circuit diagnosis. It describes net lists (connections) of a collection of Boolean circuits. In the past, many diagnosis algorithms have been evaluated on some version of the ISCAS-85 benchmark [44]. Here, the benchmark is used to show how the generated system descriptions of the cyber-physical systems compare to this standard benchmark in circuit diagnosis.

3.2. Algorithm HySD

Algorithm HySD is used for diagnosing cyber-physical systems, given an expert-defined system description and real-valued sensor measurements. Algorithm HySD_simple is used to diagnose cyber-physical systems, given a generated system description through algorithms such as DDGD. Table 15 presents the diagnosis results for algorithms HySD in column *expert SD* and HySD_simple using system descriptions from DDGD in column *generated SD*. The numbers in the column denote the percentage of how often the fault was diagnosed correctly, given 100 diagnosis runs. Column $|SD|$ shows the number of expressions that were contained within the system description. The main goal is to show whether HySD diagnoses faults correctly. Therefore, the system description is first provided analytically. Then HySD is evaluated in conjunction with DDGD to show that DDGD generates suitable system descriptions and that HySD is able to use those system descriptions to obtain correct diagnoses. For the expert-defined system descriptions in 28 of 41 experiments we were

able to diagnose a single fault minimal cardinality diagnosis (68 %), while 83% of all experiments were diagnosed correctly. Using generated system descriptions with DDGD, 81% were diagnosed correctly. This means, with system descriptions generated using Granger causality, diagnosis accuracy only dropped by 1%.

3.2.1. Using expert-defined System Descriptions

This subsection presents the results for algorithm HySD, where the system description was specified by experts a-priori. The experiments show the general functioning of the algorithm and its ability to find faults. Data from two systems was used for evaluation: The four-tank model from first principles written in Python and the simulation of the Tennessee Eastman Process from Downs et al. [230]. These are summarised in the first two experimental systems in Table 15. The other systems will be discussed in the next subsection.

The first experiment in Table 15 shows the different fault modes of the four-tank model (Figure 3) with the injected faults and whether or not the faults were detected. Each fault p_i denotes a fault in the respective pipe in the form of a clogage. An x in the column *expert SD* denotes that the injected fault was among the result set of the diagnosis algorithm. An x^* denotes that exclusively the injected fault was detected, meaning that a perfect result was obtained. It must be noted here that finding only the injected faults depends heavily on the granularity of the underlying data source. For example, if valve 5 stops working because the flow through it is blocked, its flow would immediately go to 0. The sampling frequency is high enough to detect this decrease in the flow rate early enough that the water level in the tanks is not yet significantly affected. However, in large industrial plants sampling rates are often far lower. A faulty component might then only be recognised once its effects have propagated into other observations from other components. HySD was executed for 300 time-steps. As the criterion in Table 15, we evaluated the output of the diagnosis algorithm in the time step 101, which was directly after the fault had been injected. It is evident that due to the SMT logic constraints, every unexpected change in the behaviour of the components would be detected immediately.

The second system evaluated in Table 15 shows the results for six experiments with the Tennessee Eastman Process from Downs et al. The injected faults for IDV 16 through 20 have an insufficient description for validating results. The other faults not described contain fault modes that are not associated with single components and thus cannot be evaluated. We were able to find all faults that had an identifiable component fault as the cause of the process disturbance. The change of input ratios that some IDVs such as IDV 1 exhibit, can only be detected indirectly, since no observations are available at the inputs.

System	Fault mode	expert SD	generated SD	SD
Four Tank	p_0 - Pipe leakage	x^*	NA	12
	p_3 - Pipe leakage	x^*	NA	12
	p_3 - Pipe leakage	x^*	NA	12
	p_5 - Pipe leakage	x^*	NA	12
	p_6 - Pipe leakage	x^*	NA	12
	p_1, p_3 - Pipe leakage	x^*	NA	12
	p_4, p_5 - Pipe leakage	x^*	NA	12
TE - Downs	IDV 1 - Feed ratio changed	-	NA	30
	IDV 6 - Pipe A feed loss	x^*	NA	30
	IDV 8 - Feed ratio changed	x^*	NA	30
	IDV 13 - Reactor kinetics fault	-	NA	30
	IDV 14 - Reactor cooling fault	x^*	NA	30
	IDV 15 - Condenser cooling fault	x^*	NA	30
TE - Manca	Valve A	NA	1.0	47
	Valve C	NA	1.0	47
	Valve E	NA	0.3	47
	Valve Purge	NA	-	47
	Condenser	NA	1.0	47
S1	pump - Slow	x^*	1.0	39
	tank1 - Leakage	x^*	1.0	39
	valve1 - clogged	x^*	1.0	39
	valve0 - clogged	x^*	1.0	39
S3	pump - Slow	-	-	80
	tank1 - Leakage	x^*	1.0	80
	tank2 - Leakage	x^*	1.0	80
	valve2 - clogged	x^*	1.0	80
	valve3 - clogged	x^*	1.0	80
S4	tank2 Leakage	x^*	1.0	117
	pipe4 - clogged	x^*	1.0	117
	valve3 - clogged	x^*	1.0	117
	valve6 - clogged	x^*	1.0	117
S6	tank - Leakage	x^*	1.0	6
	valveLeft - clogged	x	-	6
	valveRight - clogged	x	-	6
Mixer	pipe - clogged	x^*	1.0	38
	pipe - Leakage	x	0.57	38
	pipe - clogged \wedge Leakage	x	0.57	38
Distill	pipe - clogged	x^*	1.0	48
	pipe - Leakage	x	0.66	48
	pipe - clogged \wedge Leakage	x	0.5	48
Filter	pipe - clogged	x	0.66	30
	pipe - Leakage	x^*	1.0	30
	pipe - clogged \wedge Leakage	x	0.5	30
Bottling	pipe - clogged	x^*	1.0	33
	pipe - Leakage	x	0.66	33
	pipe - clogged \wedge Leakage	x	0.66	33

Table 15.: Experimental results using expert defined and generated system descriptions. In column *expert SD*, an x^* denotes a minimum cardinality diagnosis. An x denotes that the correct faulty component was among the result set. A – denotes that the injected fault was not diagnosed. Cells with NA denote the experiment was not applicable. Numbers in column *generated SD* denote the percentage how often the correct diagnosis was achieved over 100 runs. $|SD|$ denotes the size of the system description.

3.2.2. Using Generated System Descriptions - Algorithm DDGD

This subsection evaluates the algorithm DDGD in conjunction with HySD. DDGD (Chapter 4, Part III) generates system descriptions from process data using Granger Causality. HySD is called through the wrapping algorithm DDA-IM, integrates the generated system description with residuals, and performs a hitting set calculation for diagnosis. This sections thus shows that both algorithms DDGD and HySD work in conjunction to generate system descriptions and diagnose faults in cyber-physical systems.

Due to space constraints, we only produce an exemplarily generated system description for the system with the smallest number of components (S_1). System S_1 is a one-tank system in which a pump fills a tank through a valve. The list below shows the generated system description from the DDGD algorithm. Please note that the other generated system descriptions can be found in the Appendix. On the left-hand-side of the implications the components are evident, while the right-hand-side shows the Granger-causal variables from OpenModelica that were converted into residuals.

$$\begin{aligned} & \text{tank} \rightarrow \text{valve0.V} \\ \text{pump} \wedge \text{valve0} & \rightarrow \text{pump.V} \\ & \text{tank} \rightarrow \text{valve1.V} \\ & \text{tank} \rightarrow \text{tank1.level} \\ & \text{tank} \rightarrow \text{pump.V} \\ \text{valve1} & \rightarrow \text{tank1.level} \\ \text{valve1} & \rightarrow \text{pump.V} \\ \text{pump} \wedge \text{valve0} & \rightarrow \text{tank1.level} \\ \text{pump} \wedge \text{valve0} & \rightarrow \text{valve1.V} \\ & \text{valve1} \rightarrow \text{valve0.V} \end{aligned}$$

The data for the Tennessee Eastman Process (TE) was taken from Manca [231]. In the third experiment of Table 15 it is evident that faults in valves A and C, as well as the condenser were well recognised. With 86 %, these are also the predominant faults that occur. Faults in Valve E were less well recognised, while none of the faults within the purge valve were diagnosed correctly. This can mainly be attributed to two reasons: Firstly, for creating the system description, algorithm DDGD also used only few instances of those faults. Thus, fault information is limited. Second, the valves are harder to observe within the process, such that the generated residuals are not as accurate as those for the other types of fault due to the sensor location. Still, over all runs we achieved a diagnosis accuracy of 89% using the generated system descriptions.

The experiments from the third to last rows in column *generated SD* in Table 15 show diagnosis results for all systems using DDGD and HySD of the respective benchmarks. The number 1.0 indicates that all faults were identified correctly in all 100 runs. A number smaller than 1 indicates the percentage of how often the fault was correctly identified. In summary, most faults are correctly recognized. However, especially for system S_6 it is evident that automatically generated models sometimes are too limited. The diagnosis is

wrong, since the generated expressions only describe the behaviour of the tank, but omit the valves. This would not occur, when experts define the system description.

3.2.3. Using Generated System Descriptions - Algorithm DDRC

Algorithm DDRC (Chapter 3, Part III) also generates system descriptions from process data, but uses an even more uninformed approach. This makes it also harder to evaluate, as its output needs to be properly interpreted. While DDGD needs an annotation when faults happened in historical data, DDRC learns system descriptions solely through a correlation analysis between process data (S) and quality data (M). Similar to the last chapter, algorithm DDA-IM is used to diagnose the generated system description of DDRC. The evaluation in this section is done qualitatively on three data-sets: an injection molding machine, system S_3 of the benchmark from Balzereit et al. [82], and the compounding process. For the qualitative evaluation, the correlated signals must be interpreted by an expert and need to be translated into associated components. For example, when a system description generated with DDRC is used with a diagnosis algorithm such as HySD_simple the output may be some signal name such as *flowo*. An expert must now associate the signal with the respective component. Therefore, an x in the result tables denotes that the diagnosed signals belong to the component that was assumed to be faulty. All experiments were executed once for each threshold and with fixed random seed. Some resulting system descriptions can be found in the Appendix.

To set up the system description Φ we used only data from normal operating conditions without any anomalies to create the weak fault models. Faults were injected by assigning a random value ($\{\top, \perp\}$) to all variables $m \in M$, such that 20% of observations are assigned \perp . This results in a variety of faulty components for each experiment run.

As with algorithm DDGD, the goal of the evaluation is to prove that the system description Φ shall be approximately correct to perform diagnosis and that the diagnosis algorithm shall be able to identify the injected fault(s). Our experiments show that a completely uninformed data-driven approach is sufficient to achieve some promising diagnosis results.

Correlation threshold τ	0.1	0.3	0.5	0.7	0.8	0.9
Sensible system description achieved	x	x	x	x	x	x
Diagnosis correct	x	x	x	x	x	x

Table 16.: The results for the injection molding machine using DDRC. An x denotes that, given the threshold, an expressive system description was generated and that it was used to obtain a correct diagnosis using a random number of faulty observations.

Table 16 contains the results for the injection molding machine with different thresholds for correlation. The first row shows, whether the generated system description was suitable for diagnosis. The second row shows whether the diagnosis algorithm was able to determine the injected faults, given the generated system description. This shows that our method works well for the intended use-case of injection molding machines. The correct diagnosis was found in each experiment and the generated system descriptions are approximately correct, given the data-driven approach.

The simulation of the four-tank system (S_3), which was already used as the running example, was created with OpenModelica and contained 1939 signals. Of these, many were

3. Empirical Results

Modelica-internal variables that do not exist in real processes. Thus, 18 of these variables were identified as measurement variables, by filtering the signal names to only those which were associated with temperature, flow, and water level. Given the size of the data-set, we argue that our approach shows good results (Table 17) as can be validated on the running example and by choosing a reasonable value for τ .

Correlation threshold τ	0.1	0.3	0.5	0.7	0.8	0.9
Sensible system description achieved	-	x	x	x	x	x
Diagnosis correct	-	x	x	x	x	x

Table 17.: The results for system S_3 using DDRC. An x denotes that, given the threshold, an expressive system description was generated and that it was used to obtain a correct diagnosis using a random number of faulty observations.

Table 17 contains the results for system S_3 with different thresholds. An x denotes that the system description or the diagnosis was correct. Again, faults were injected through randomly setting some signals in set M to false, which simulates wrongly closed valves or a leaky tank. Except for a correlation threshold of 0.1, all threshold values generated suitable system descriptions. This evaluation shows that also the standard four-tank systems established in the literature, perform quite well given this uninformed approach.

Correlation threshold τ	0.1	0.3	0.5	0.7	0.8	0.9
Sensible system description achieved	x	x	-	-	-	-
Diagnosis correct	x	x	-	-	-	-

Table 18.: The results for the compounding process using DDRC. An x denotes that, given the threshold, an expressive system description was generated and that it was used to obtain a correct diagnosis using a random number of faulty observations.

Table 18 contains the results for the compounding process with different thresholds. In contrast to the first two systems, signals within the compounding process are not highly correlated. This is partly due to some signals being aggregated over time. For example, the laboratory data is only available once a batch is finished and has been stored for some time. As a result, only few distinct expressions exist such that Φ is comparatively small, with about 8 expressions and $\tau = 0.3$. Above a threshold $\tau = 0.5$ only one expression exists, which is insufficient for diagnosis. The injected faults were simulated through randomly setting some signals in set M to false. This simulates wrong measurements of ingredients, viscosity, or other material properties. Thus it can be concluded that in some use-cases the uninformed, correlation-based approach of DDRC is insufficient to generate suitable system descriptions. In case of the compounding process, the main reason is that signals cannot be well-divided into sets S and M .

3.3. Checking model validity - Algorithm CheckDiag

Chapter 3 and 4 in Part III have shown how a system description is generated from data. Using this system description, a diagnosis algorithm is able to decide whether a set of observations OBS exhibits inconsistent behaviour in regard to some model SD . However, learning a system description, especially in the uninformed, signal-based case (Chapter 3 Part III) requires experts to thoroughly check the system description. The primary task

of these experts is to check whether the logical expressions *look usable* for diagnosis. But for experts this is often ill defined. In practice, they check that expressions are not *too long* and that *a suitable amount* of expressions exist. But this is too vague for any theoretical basis and as an objective criterion. This section provides a formalisation to measure the validity of logical system descriptions. In particular, this section provides the definitions and an algorithm called `CheckDiag`, which outputs a score $\omega_q \in [0, 1]$, where a number close to 0 indicates perfect diagnosability (all components can be diagnosed, should they be faulty) and values close to 1 indicate non-diagnosability (faulty components cannot be distinguished). Further, algorithm `CheckDiag` outputs a set of those components, which cannot be distinguished for fault diagnosis.

The theory presented here belongs to the area of diagnosability research. Sampath et al. [233] have published a seminal paper for discrete event systems in 1995. The same thoughts were used by Jiang et al. [234] and Pecheur et al. [235]. Also recently, Bittner et al. [236] have published an extensive formalisation of diagnosability using error bounds for discrete event systems. Grünbaum et al. have presented an approach to validate causal models through methods of quantitative probing [178]. All of these approaches, however, are graph-based and focus on analysing transitions between system states. Another related area of research is the definition of cones within circuit diagnosis by Stern et al. [20]. The formalisation in this section is most similar to the idea behind finding cones in Boolean circuit diagnosis. Within their area the authors define criteria, which let a diagnostician decide whether some faults can be distinguished or whether only the head-circuit of a cone can be diagnosed, while the components (using some form of hierachy) *before* this component are hidden. In this chapter the goal is to extend this idea of cones to the propositional logic expressions of the learned system description SD. In summary, an algorithm is required that validates a generated system description to check whether it is expressive enough to diagnose faults in cyber-physical systems.

3.3.1. Theory of Diagnosability

First, we are going to introduce some necessary definitions to analyse diagnosability. Then, algorithm `CheckDiag` is introduced and it is shown, how it can be used to analyse system description automatically and generate an objective criterion for diagnosability.

Definition 41 (Rule Set). *A rule set \mathcal{R}_s of a system description SD is the set of components of the left-hand side of the implication within some logical expressions φ_i .*

It is first convenient to define the rule set to simplify the system description. The rule set contains the expressions from SD. Underlying is the assumption that by design every expression within SD is associated to a unique measurement or residual. Removing the implication thus does not lead to a loss of information. Within \mathcal{R}_s only the sets of components without connectives are contained.

Given the rule set \mathcal{R}_s with only the names of the components, it becomes possible to analyse the system diagnosability.

Definition 42 (Diagnosability). *Given some sets $F_i, F_j \dots \subset F$ of components, a component c_i is diagnosable, if there exists a single component c_i in at least two sets, with $c_i \in F_i \wedge c_i \in F_j \wedge \dots$*

for any $i, j \in \mathbb{N}$. A component is always diagnosable, if it is the single component associated to a unique residual.

Diagnosability for a component is defined such that the component is not *masked* or within a cone, when using the terminology of Metodi et al. [100]. This is calculated by checking whether the component health can be determined through multiple tuples (multiple expressions) within Φ . At the same time, it must be ensured that no other component is solely observable through the exact same rules. Applying the diagnosability check for all components over the whole set F makes it possible to determine perfect diagnosability.

Definition 43 (Perfect Diagnosability). *Perfect diagnosability exists, when $\forall c_i \in F$ at least two subsets exist, such that $\exists F_i, F_j : c_i \in F_i \wedge c_i \in F_j$, with $i, j \in \mathbb{N}$ and $F_i, F_j \in F$.*

If each component is observable through at least two sets F_i, F_j and no other component is observable through the same observations, the set exhibits perfect diagnosability. In practice this is hard to achieve, especially when the system description is learned from data. For those components which cannot be observed perfectly, the term hidden components is defined.

Definition 44 (Hidden Components). *Hidden components are non-diagnosable components, where for at least two components $c_i, c_j \in F_m$ and $c_i, c_j \in F_l$, with $F_m, F_l \in F$ and $i, j \in \mathbb{N}$.*

These are sets of components where, given the learned system description, no observation can determine which of the components are faulty. Instead, a diagnosis algorithm must assume that once all other components are exonerated, the fault must be located within the set of hidden components. This assumption, however, increases the achievable minimum cardinality diagnosis, as hidden components cannot be further discriminated through any diagnosis algorithm.

3.3.2. Algorithm CheckDiag

The algorithm CheckDiag (Algorithm 6) performs the diagnosability check and outputs \top , when perfect diagnosability exists and outputs the set of hidden components, when components cannot be distinguished from each other given the system description.

The algorithm takes as input the rule-set \mathcal{R}_s and checks iteratively, whether components are diagnosable. From the rule-set, set F is derived. The set is sorted according to cardinality with ascending order, such that rules with the least number components are visited first. Then, for each component it is checked in how many other rules the component is represented. If the component is the only member in its set, it can be diagnosed (this is the trivial case). Components that can be distinguished in two or more sets are set as diagnosable.

By dividing the number of hidden components within the output set F through the number of components within the input set \mathcal{R}_s , it is possible to calculate a quotient. The quotient can be used to obtain a quantitative measurement about the expressiveness of a learned system description. In other words, it determines how many components can be successfully diagnosed.

Algorithm 6: CheckDiag: Check for diagnosability

Data: \mathcal{R}_s
Result: F

```

1  $F \leftarrow \mathcal{R}_s$  ;
2  $F_{sorted} \leftarrow F.sortBy(smallest)$  ;
3 foreach  $card = [1..max(|F_{sorted}|)]$  do
4    $f \leftarrow F_{sorted}.setsWithCardinality(card)$  ;
5   foreach  $f_i \in f$  do
6     foreach  $c_{f_i} \in f_i$  do
7       if  $\exists c, \text{ where } c \in (f \setminus f_i)$  then
8          $F \leftarrow F \setminus c_{f_i}, \forall f_i \in F$  ;
9          $F_{sorted} \leftarrow F.sortBy(smallest)$  ;
10 if  $F = \emptyset$  then
11    $\text{return}(\top)$  ;
12  $\text{return}(F)$  ;
    
```

$$\omega_q = \frac{\sum_{f'_i \in F} |f'_i|}{\sum_{f_j \in \mathcal{R}_s} |f_j|} \quad (3.1)$$

3.3.3. Example: CheckDiag

The execution of the algorithm is visualised in Tables 19 and 20. Table 19 shows the execution when SD is perfectly diagnosable. The leftmost column shows several rules with components labelled $COMPS = \{a, b, c, d, e\}$. The arrow indicates that component b can be explained through rules 1 and 3. Since it is diagnosable, it is removed. In the second column, through the removal of component b , the algorithm determines that components a and d can also be diagnosed and removes them. Using this procedure, the algorithm continues until all rules are reduced to the empty set.

$a \wedge b$	a	\emptyset	\emptyset
$a \wedge b \wedge c$	$a \wedge c$	c	\emptyset
$b \wedge d$	$\xrightarrow{\{b\}}$ d	$\xrightarrow{\{a,d\}}$ \emptyset	$\xrightarrow{\{c,e\}}$ \emptyset
$c \wedge e$	$c \wedge e$	$c \wedge e$	\emptyset
$a \wedge e$	$a \wedge e$	e	\emptyset
$a \wedge d \wedge e$	$a \wedge d \wedge e$	e	\emptyset

Table 19.: Example of a perfectly diagnosable logical system description

Table 20 provides an example, where perfect diagnosability cannot be established. Here, the algorithm starts by removing component c and d as they are diagnosable through the last two rules. The same is done with component e . But components a and b cannot be further distinguished. Recall here that in the last two chapters we defined that each logical expression has an associated observation on its right-hand-side. The result of CheckDiag is interpreted that, given some observations, it cannot be determined whether component

a or component b is faulty. Certainly, when a system description contains many of these cases, diagnosability of the whole system is limited.

$a \wedge b$	$a \wedge b$	$a \wedge b$
$a \wedge b \wedge c$	$a \wedge b$	$a \wedge b$
$c \wedge e$	$\xrightarrow{\{c,d\}}$ e	$\xrightarrow{\{e\}}$ \emptyset
$d \wedge e$	e	\emptyset

Table 20.: Example of a logical system description with hidden components $\{a, b\}$

3.3.4. Trivial Estimation of Diagnosability

Combing through large system descriptions can in some cases take a long time. Therefore, this section introduces a computationally less intensive approach for estimating the expressiveness of system descriptions. For this, generated rules \mathcal{R}_s are evaluated according to their validity using a heuristic. It is assumed that to be specific, a rule must encompass only a small number of components and the number of rules must be high enough. If the number of rules is small, while each rule contains many components, diagnosis accuracy would suffer, as many possible hitting sets would be calculated. The trivial estimation can be calculated through the heuristic

$$spec = \frac{\sum_{r \in \mathcal{R}_s} |r|}{|\mathcal{R}_s|} \quad (3.2)$$

with $|r|$ being the number of components per rule and $|\mathcal{R}_s|$ being the size of the rule-set. Consequently, Equation 3.2 provides a score $spec \in \mathbb{R}^+$ that would optimally be close to 1 and deviates further, if the rule set gets larger and less well-formed.

Overall, the validity of a set of tuples \mathcal{R}_s can be formulated as the optimization function

$$spec_{opt} = \left(\sum_{r \in \mathcal{R}_s} \min(|r|) \right) \wedge \max(|\mathcal{R}_s|) \quad (3.3)$$

where $|\mathcal{R}|$ is the size of the set of the rule set \mathcal{R} and r is the number of components c within one rule.

Both formulae 3.1 and 3.3 can be reformulated to calculate the Pareto optimum with optimization algorithm over the parameters of the DDGD and DDRC algorithms.

3.3.5. Results

CheckDiag evaluated the system descriptions from the Tennessee Eastman process from Manca et al., systems S1, S3, S4, S6, the bottling process, and the ISCAS-85 benchmark. The goal is that algorithms DDGD and DDRC only generate system description, that achieve a validity quotient $\omega_q = 0$. If this is not possible, the goal is to use parameter tuning in order to reduce ω_q as far as possible. The model validity quotient shows for an arbitrary rule set \mathcal{R}_s , which components are diagnosable (or hidden). Therefore, any propositional

logic system description can be evaluated. This is shown through the evaluation of the ISCAS-85 benchmark. Tables 21 and 22 show the calculated quotient, as well as the sizes of set COMPS and the hidden components for each investigated system. As can be seen in the example in Table 20, even hidden components can still point a practitioner in the right direction as it reduces the potential search space. In many cyber-physical systems the amount of hidden components is due to their limited instrumentation. While Boolean circuits can be analysed quite well, processes such as the tank systems have rather few sensors, given their number of components. It can also be seen that system S6 contains only one component, although an expert would certainly assert that more components can be distinguished within the system.

System	COMPS	# hidden	ω_q
TE	4	0	0.0
S1	4	2	0.5
S3	6	2	0.33
S4	5	2	0.4
S6	1	0	0.0
Mixer	7	3	0.429
Distill	8	5	0.625
Filter	6	3	0.5
Bottling	5	2	0.4

Table 21.: Model validity for all diagnosed systems

System	COMPS	# hidden	ω_q
c17	9	9	1
c432	187	33	0.1764
c499	75	22	0.2933
c880	417	23	0.0552
c1355	555	22	0.0396
c1908	888	30	0.0338
c2670	1286	35	0.0272
c3540	1697	133	0.0784
c5315	2362	20	0.0084
c6288	2416	2384	0.9867
c7552	3611	131	0.0363

Table 22.: Model validity for systems of the ISCAS-85 benchmark

Table 22 shows the CheckDiag applied to the well-known benchmark of ISCAS-85 circuits. Full-observability was assumed so that all intermediate variables were evaluated. It is evident that for most circuits only a small number of components are hidden. The known netlists and derived propositional logic system descriptions can thus be assumed to be well-suited for diagnosis.

3.3.6. Comparing signal-based to component-based Granger Causality

In Chapter 5 in Part III it was shown that differences exist between signal-based and residual-based approaches. This subsection demonstrates the difference between system descriptions generated through using residuals as observations in the case of algorithm DDGD and using signals only in the case of algorithm DDRC.

System	Method	Average rule size	# of rules
TE	DDRC - signal	47	2
S3	DDRC - signal	16.5	18
TE	DDGD - residual	2.82	47
S3	DDGD - residual	1.13	16

Table 23.: Comparison between rules generated through signal-based and component-based Granger Causality

Table 23 shows the differences between expressions generated by DDGD and DDRC for the two systems S3 and TE, exemplarily. While DDGD works on information about component

3. Empirical Results

health states (residuals), DDRC uses a signal-based approach. DDRC is faster than DDGD, but since DDRC is uninformed it relies heavily on the correct division of the data into process data and quality data. Overall, the table shows that DDGD obtains smaller expressions, but that more expressions are generated.

4. Theoretical Analysis

The chapter analyses the developed algorithms HySD, HySD_simple, DDGD, DDRC, DDA-IM, and CheckDiag analytically. The first section analyses the theoretical complexity of the algorithms and shows how efficient the algorithms can be implemented on classical computers. The second section analyses the running time of the algorithms asymptotically.

The computationally hardest problems contained within this thesis are the MaxSAT and hitting-set calculations in algorithms HySD and HySD_simple. To generate fault hypotheses, we use the MaxSAT problem, which then passes the hypotheses to the hitting-set algorithm for discrimination. Both of these problems are NP-complete: The known NP-complete SAT [237] problem can be reduced to MaxSAT and vertex cover [238] can be reduced to the hitting-set problem, respectively. Only the worst-case complexity is presented here, although more efficient algorithms in the average case may exist. Some of those more efficient algorithms use, for example, kernelization as described by Abu-Khzam et al. [238] or parametrization [239, 240]. The advantage with algorithm HySD is, that as long as their interface adheres to the diagnosis interface presented in Section 1.7 in Part I they can be easily replaced and thus make the method computationally more efficient.

Table 24 presents the computational complexity and running time of the developed algorithms in this thesis. The next two sections will explain the different algorithms in detail.

Algorithm	Complexity	running time	O-Notation
HySD	NP-complete	Exponential	$O(2^{\log(n)n})$
HySD_simple	NP-complete	Exponential	$O(2^{\log(n)n})$
DDGD	Polynomial	Quadratic	$O(n^2)$
DDRC	Polynomial	Quadratic	$O(n^2)$
DDA-IM	NP-complete	Exponential	$O(2^{\log(n)n})$
CheckDiag	Polynomial	Cubic	$O(n^3)$

Table 24.: Comparison of the computational complexity and running time of the algorithms developed in this thesis

4.1. Computational Complexity

Algorithm HySD contains four main parts driving the theoretical complexity: 1) The creation of SMT expressions (lines 1-3, and 7 of Algorithm 1). Generating the SMT expressions is done in linear time by converting residuals, thresholds, and manual specifications into the SMT format. This results in a computational complexity of $O(6n)$. 2) Obtaining the observations α (line 5 of Algorithm 1). This is done by reading from a database or table, which is also done in linear time $O(n)$. 3) Checking the satisfiability (line 9 of Algorithm 1). For the satisfiability checking we use the z3 SMT-solver [222] which converts the SMT

expressions into a MaxSAT problem. MaxSAT is known to be NP-complete [241], although specialised formulations may lead to some improvements in running time and complexity [239]. 4) The diagnosis discrimination. Running the diagnosis discrimination is also NP-complete, though, for the systems investigated in this article, the absolute running time is better than solving the MaxSAT problem. Therefore, the computationally hardest problem is solving the MaxSAT which thus determines the overall complexity of the algorithm. Please note that diagnosis and hypothesis generation algorithms within HySD can be exchanged, such that the computational complexity may be improved [22]. The same is true for Algorithm HySD_simple. The difference to HySD is only that fewer linear steps are necessary.

Algorithm DDRC exhibits a polynomial complexity. It calculates the Spearman correlation, which is quadratic, and then goes once through each cell in the result matrix and evaluates it against some threshold with complexity $O(n)$.

Algorithm DDGD also has a polynomial computational complexity. The complexity is mainly determined through the calculation of the Granger Causality. Within the algorithm, each input value is compared to every other value at least once, but usually multiple times, resulting in a complexity of $O(n^2)$.

DDA-IM solves the same NP-complete problem of diagnosis the same as Algorithm HySD, through its call to HySD_simple. The core of the algorithm is a calculation of the hitting-set and a solution to the MaxSAT problem to obtain the diagnosis results.

Algorithm CheckDiag exhibits polynomial complexity, as it only analyses sets derived from the rule set \mathcal{R}_s , without recursion or calls to NP-complete algorithms.

4.2. Running-time Evaluation

Figure 13 presents the measured running time of algorithm HySD for the experiments in the third to last rows in Table 15 in Chapter 3. These encompass the experiments of the two tanks benchmarks, and the Tennessee Eastman Process. Figure 13 presents the running time against the number of expressions within the system description ($|SD|$). As expected, the running time increases with an increasing number of expressions. However, Figure 13 does not clearly show the expected exponential increase in running time. This is mainly due to the small problem instances, although these represent real systems. It must be noted that the expected exponential behaviour stems from the fact that the applied diagnosis algorithms solve an NP-complete problem. However, the main functionality of the algorithms introduced in this thesis, namely the transformation of observations and system descriptions from cyber-physical systems to classical consistency-based diagnosis algorithms, is computationally easier than the diagnosis itself. In the following we will analyse each of the algorithms in detail.

For Algorithm HySD, the functions `makeSMT_xxx()` take models and observations and convert them into $SMT - \mathcal{LRA}$ expressions in linear time, thus having a running time of $O(n)$, with n being the number of inputs. The same is true for functions `getReadings()`, `generateHypotheses()`, and `generateSymptoms()`. MaxSAT solving, as well as the diagnosis `diagnose()`, require exponential running time in the worst case. They have to compare every possible component (propositional symbol) to every other component in each propositional logic formula. This results in the exponential running time of Algorithms HySD

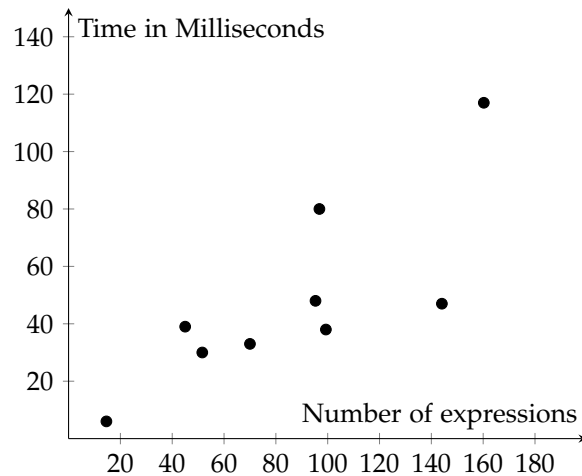


Figure 13.: Running time in milliseconds of algorithm HySD compared to the problem size

and HySD_simple of $O(2^{\log(n)n})$. In absolute terms, the running time for the experiments used in this thesis with the given hardware was below one second. However, the problem instances (system descriptions) were quite small from an evaluation standpoint. Contrarily, the empirical evaluation has shown that many typical cyber-physical systems are already covered by such small system descriptions.

Algorithm DDRC computes the Spearman correlation and then goes through the result matrix cell by cell. The worst case running time is thus quadratic $O(n^2)$. The running time of algorithm

DDGD is quadratic as well $O(n^2)$ in the worst case due to the multivariate Granger Causality test in which each sensor value is tested against every other sensor value (or health state).

Solving the diagnosis problem in Algorithm DDA-IM is NP-complete and thus exhibits exponential running time due to the same reasons as in Algorithm HySD, resulting in $O(2^{\log(n)n})$ through the calculation of the hitting-set.

In the worst case CheckDiag has a cubic running time $O(n^3)$, if all expressions in the system descriptions are of different lengths and no two components can be easily separated. In the average case, however, many components should be easily diagnosable already at the start of the algorithm, which thus leads to a significant reduction in running time.

Part V.

Conclusion and Future Work

1. Discussion

Within this thesis, a general method for diagnosing faults in cyber-physical systems using six novel algorithms was presented. So far, many existing diagnosis approaches only work within narrow fields of application or rely on specialised and hard to obtain expert knowledge. The approach in this thesis, using the algorithms HySD, DDRC, and DDGD, allows scaling of the use of expert knowledge: using DDRC that only requires two data sets for learning the system description and thresholds from historical data for residual generation, using DDGD with known fault annotations, or using manually crafted models for highly specialised applications. This scaling is facilitated through the use of propositional and SMT logic. SMT logic easily integrates plain, propositional logic expressions, while also allowing the integration and evaluation of residuals and health functions. Different traditional consistency-based diagnosis algorithms can be applied and exchanged for checking the consistency of the logical expressions and computing the diagnoses. This allows practitioners to select the diagnosis algorithms by running time, by code availability, or by some other criterion. Further, the method allows the integration of the results of machine learning algorithms: Instead of computing a residual through a threshold equation it would easily be possible to train some machine learning model for some signals and only evaluate the output of the machine learning algorithm within the logical framework. Overall, all the main developed algorithms HySD, DDRC, and DDGD work within the domain of explainable artificial intelligence, which allows experts to trace each execution step and get insights about the execution of the algorithms.

While the presented methodology in this thesis is broadly applicable in the domain of cyber-physical systems, it also has several limitations. The crux of the broad applicability is that in some domains specialised algorithms might perform better. To illustrate, using injection molding machines, a machine learning algorithm can be trained based on historical data and expert annotations that outputs faults, given process data. Therefore, highly accurate, specialised machine learning algorithms are expected to generally perform better against the presented method. But while machine learning algorithms would require advanced techniques from the domain of transfer learning to be adaptable to other use-cases, the presented methodology extends easily and while maintaining explainability.

Another drawback is that so far a method is missing to generate provably correct models from process data. Algorithms DDRC and DDGD learn system descriptions from process data with minimal constraints. But while their generated models can be heuristically checked for diagnosability, there is so far no method which measures how well the generated model matches the real, physical system. When avoiding learning system descriptions, one faces the problem of obtaining correct and extensive models from process experts. Further, algorithm DDRC requires that the data is available in two sets S and M , which some industrial applications cannot provide. Algorithm HySD is able to process elaborate SMT- \mathcal{LRA} system descriptions. But obtaining these is a difficult process, which is often done through process engineering and consultancy work [242]. Another drawback stems from the construction of the cyber-physical systems themselves. The entire presented methodology relies on

the availability of useful observations from the cyber-physical system. If observations are missing or corrupted, diagnosis accuracy and expressiveness of the result suffer. In many cases, these problems can be circumvented through approximating missing process data or through integrating noise processing into the residual generation. Further, the possibility of sensor faults could be integrated into the system descriptions by integrating sensors in the definition of the components.

While the presented theory of fault propagation is certainly useful to create the system description required by HySD, there is still some manual effort involved in finding the right equations, identifying sensors, and integrating them into SMT- \mathcal{LRA} . Mistakes can quickly lead to system descriptions that are missing important behaviour or interdependencies, which then limit the overall diagnosis capability.

All experiments have been performed on benchmarks involving process industrial use-cases, mostly in the form of tank systems. Although the approach is general enough that it may be transferred to other domains, this has so far not been proven empirically. Therefore, suitable benchmark data is highly necessary. Some drawback might exist, for example, when a cyber-physical system only contains a small number of continuous values, while most of the signals are Boolean. Then fault diagnosis approaches such as timed automata [37] and other discrete event systems may be preferable.

Algorithm HySD is currently focused on identifying single-faults. Underlying is the assumption that within some system it is uncommon that a fault is due to more than one faulty component. But some systems may exhibit this behaviour. Slight changes to the output post-processing of HySD, however, would allow this functionality. Intermittent faults with a temporal dependency and strong-fault models cannot be used yet. While the latter would be technically possible, especially through the capabilities of SMT- \mathcal{LRA} , the former would require significant changes to the reasoning and would possibly involve the use of temporal logics [65].

2. Conclusion

In the course of this thesis, six different novel algorithms were introduced. Each of these contributes to the goal of obtaining a general method for diagnosing cyber-physical systems in the real world. The main algorithm is the novel diagnosis algorithm HySD, which was developed to identify faults in cyber-physical systems. Back in Chapter 1 Part I, it was identified that thus far no practical and lightweight approach exists for diagnosing cyber-physical systems. While it was also shown that consistency-based diagnosis algorithms have been well-researched and powerful algorithms exist, many of these were used in narrow domains such as Boolean circuit diagnosis. Given past research, however, there was an intuition that those consistency-based diagnosis algorithms should be adaptable within the domain of cyber-physical systems. Several authors had attempted this in the past, for example, Kühnert et al. [174], Grastien et al. [64], Narasimhan et al. [63], Bunte et al. [48], and several others. The novelty in the presented approach lies in a holistic formulation (in SMT logic) to obtain residuals and use them with consistency-based diagnosis, thus obtaining a practical approach for diagnosis, while making use of established foundations. Further, it was shown how diagnosis models can in many cases be learned from data. Although those models cannot yet be proved to be correct, it was shown that in many use-cases they generate approximate and usable propositional logic expressions. Therefore, users of the presented diagnosis algorithms can choose one of three options to perform diagnosis: i) Running the uninformed algorithm DDRC to obtain a system description and then performing diagnosis using HySD. ii) Running the slightly more informed algorithm DDGD to obtain a system description and then performing diagnosis using HySD. iii) Using experts to create a system description through SMT expressions and then running HySD. For the underlying theory of fault propagation, Chapter 5 in Part III has provided a theoretical foundation for experts to study dependencies in cyber-physical systems. This enables them to create more advanced component models than the threshold equations used in this thesis.

Overall, the research questions were solved as follows. RQ₁ was solved through the development of algorithm HySD, by choosing, analysing, and adapting SMT logic to fuse residuals with consistency-based diagnosis, and by describing a practical method to integrate and evaluate process signals. RQ₂ was answered through the theoretical analysis of fault propagation and the comparison between signal-based and residual-based fault diagnosis. RQ₃ was solved through the algorithms DDRC and DDGD.

While the introduced algorithms solve an acute and practical problem in industry, particularly in production processes, they have still some limitations. One major drawback is that there is still quite some expert knowledge involved. Using DDRC, the model needs to be checked by experts. Using DDGD, fault annotations are necessary to learn Granger causalities. And algorithm HySD requires the specification of residual evaluation functions, regardless of how the system description was obtained. Thus, the performance of all of the algorithms still relies on accurate expert knowledge to generate good results. It must be noted, however, that the amount of required expert knowledge compared to previous

2. Conclusion

approaches was reduced. Still, the presented methodology is expected to perform worse than dedicated and optimised applications for narrow use-cases when enough data is available. Still, the evaluation has shown that using HySD / HySD_simple for fault diagnosis and algorithms DDGD and DDRC for learning system descriptions can successfully be applied for cyber-physical systems in industrial production contexts.

3. Outlook

Several questions still remain open. These questions are structured within four main research directions: i) Generating better system descriptions, ii) Integrating expert knowledge easier and more structured. iii) Combining the diagnosis approach with expert systems, such as EUREKA [15], iv) Solve the NP-complete hitting-set calculations more efficient through the use of quantum computers. Each of these directions should now be briefly evaluated.

The evaluation has shown that, while the generated system descriptions using the algorithms DDRC and DDGD are approximately correct, the correctness cannot yet be proven. Future work should focus on four tasks: First, a method which allows tuning of the meta-parameters of Granger Causality and Spearman correlation, maybe through suitable pre- and postprocessing, is required. Second, although Granger causality is considered one of the best methods [93, 184], several other algorithms exist to recognise dependencies in data. It would be beneficial to theoretically and empirically analyse the correctness of all suitable algorithms. Third, so far only propositional logic system descriptions are generated. Others have shown how SMT expressions can be learned from data [87]. Those SMT-learning approaches should be integrated into DDGD. This would free experts of the need to explicitly specify residuals. Fourth, a method should be developed which allows proving whether a generated model is correct. This could be a comparison between inferred causality and genuine causality (in the words of Pearl [219]). Of the four mentioned tasks this might be the hardest to achieve, as it requires the existence or formulation of a ground truth, and a provably correct simulation of the diagnosis model's predictions compared to this ground truth.

Practically integrating expert knowledge is another main research direction. So far, algorithm HySD requires an expert to formulate a system description in SMT logic or propositional logic augmented with Python program code. For many industry experts, who are specialised in their respective fields, this is no easy task. It is therefore highly important to find concepts to integrate their knowledge [243]. Further, so far there is no automatic approach to integrate knowledge from design documentation such as piping and instrumentation diagrams [243], ontologies [48], or other sources [244]. It would be highly convenient to use some of the existing design documentation of some cyber-physical system to automatically derive minimal structure patterns and then create SMT logic templates for the use in algorithm HySD. Another possible extension would be to integrate machine learning algorithms into the SMT and propositional logic expressions. This would remove the need to explicitly formulate threshold expressions to evaluate residuals. Instead, residuals could be replaced through the output of machine learning algorithms indicating the status of sensors and components. Expert knowledge could also be formulated in some extension of temporal logic. Some attempts have been published before [65], but in practical use-cases it is important that time spans are quantified. So far, research in this direction is missing. Another practical extension would involve integrating HySD with automated reconfiguration tools such as AutoConf [245]. This would allow users to obtain

new process parameters, once faults have been detected to continue production (albeit often with reduced throughput). Nowadays, faults often lead to complete production stops so some configuration of valid, alternative parameters would save companies a significant amount of resources.

The third research direction is fusing the presented approach with expert systems. One prominent expert system is EUREKA [15], which was developed at Xerox in the 1990s to diagnose faults in printers. Nowadays, several expert systems exist for diagnosing faults in cyber-physical systems [16]. But they are often proprietary and rely almost exclusively on the input from experts. Future work should involve combining these systems with automated diagnosis methods.

The fourth research direction involves efficiently solving the NP-complete problem of hitting-set computation, or, in fact, any problem to which satisfiability and vertex cover are reducible. With the advent of industrially viable quantum computers, many authors have started to make progress in this field, primarily Leipold et al. [246, 247], Perdomo-Ortiz et al. [77, 78, 72], Bian et al. [248], or Feldman et al. [249]. But these again mostly focus on the well-understood domain of Boolean circuit diagnosis. Extending this research into the domain of real cyber-physical systems should bring significant advantages in running time for many industrial applications.

Part VI.
Appendix

1. System Descriptions of the investigated Systems

Listed are all system descriptions with residuals r_j on the right-hand side of the implication and the associated components as conjunctions on the right-hand side. Assigning a truth value to the residuals enables diagnosis algorithms to reason about possible faults within the components.

System description of running DDGD on system S_1

```
valve1 → valve0.V
valve1 → tank1.ports[1].m
pump ∧ valve0 → pump.port
valve1 → valve0.m
pump ∧ valve0 → tank1.level
pump ∧ valve0 → valve1.V
    tank → tank1.level
valve1 → pipe4.port
valve1 → valve0.port
valve1 → pump.m
    tank → valve1.V
valve1 → pipe4.flowModel.m
pump ∧ valve0 → pump.V
valve1 → pump.V
pump ∧ valve0 → valve1.port
valve1 → tank1.level
valve1 → pipe.flowModel.m
    tank → valve1.m
pump ∧ valve0 → pipe1.port
    tank → pipe2.port
valve1 → pipe.port
valve1 → pump.port
    tank → pipe1.flowModel.m
pump ∧ valve0 → pipe2.flowModel.m
    tank → sink.ports[1].m
    tank → pipe1.port
    tank → valve1.port
pump ∧ valve0 → pipe.flowModel.m
valve1 → source.ports[1].m
pump ∧ valve0 → sink.ports[1].m
pump ∧ valve0 → pipe1.flowModel.m
    tank → tank1.ports[3].m
pump ∧ valve0 → pipe.port
pump ∧ valve0 → valve1.m
pump ∧ valve0 → tank1.ports[3].m
pump ∧ valve0 → pump.m
pump ∧ valve0 → pipe2.port
pump ∧ valve0 → source.ports[1].m
    tank → pipe2.flowModel.m
```

1. System Descriptions of the investigated Systems

System description of running DDGD on system S6

```
tank → valveLinear1.port
tank → sink.ports[1].m
tank → valveLinear1.m
tank → valveLinear1.V
tank → pipe3.flowModel.m
tank → pipe3.port
```

System description of running DDGD on system Bottling

```
pumpP401 ∧ tankB402 → source0.pipe
pumpP401 ∧ tankB402 → source0.port
tankB401 ∧ pumpP401 ∧ tankB402 → source0.port
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → sink0.pipe.flowModel.m
pumpP401 ∧ tankB402 → sink0.pipe.flowModel.m
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → der(bottling)
pumpP401 ∧ tankB402 → sink0.port
pumpP401 ∧ tankB402 → source0.source.ports[1].m
tankB401 ∧ pumpP401 ∧ tankB402 → sink0.volumeFlowRate.port
pumpP401 ∧ tankB402 → der(bottling)
pumpP401 ∧ tankB402 → sink0.volumeFlowRate.port
tankB401 ∧ pumpP401 ∧ tankB402 → sink0.port
tankB401 ∧ pumpP401 ∧ tankB402 → source0.pipe
pumpP401 ∧ tankB402 → source0.volumeFlowRate.port
tankB401 ∧ pumpP401 ∧ tankB402 → source0.source.ports[1].m
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → sink0.volumeFlowRate.port
tankB401 ∧ pumpP401 ∧ tankB402 → sink0.pipe.flowModel.m
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → sink0.pipe.port
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → source0.source.ports[1].m
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → source0.volumeFlowRate.port
pumpP401 ∧ tankB402 → bottling
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → source0.port
tankB401 ∧ pumpP401 ∧ tankB402 → source0.volumeFlowRate.port
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → sink0.port
tankB401 ∧ pumpP401 ∧ tankB402 → der(bottling)
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → bottling
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → sink0.sink.ports[1].m
tankB401 ∧ pumpP401 ∧ tankB402 → bottling
tankB401 ∧ pumpP401 ∧ tankB402 → sink0.pipe.port
pumpP401 ∧ tankB402 → sink0.sink.ports[1].m
pumpP401 ∧ tankB402 → sink0.pipe.port
tankB401 ∧ pumpP401 ∧ tankB402 ∧ valveDiscrete ∧ valveV403 → source0.pipe
tankB401 ∧ pumpP401 ∧ tankB402 → sink0.sink.ports[1].m
```

System description of running DDGD on system Filter

```
tankB101 ^ pumpP101 ^ tankB102 → sink0.pipe.port
pumpP101 ^ tankB102 → sink0.volumeFlowRate.port
pumpP101 ^ tankB102 → sink0.pipe.flowModel.m
pumpP101 ^ tankB102 → sink0.pipe.port
tankB101 ^ pumpP101 ^ tankB102 → source0.port
tankB101 ^ pumpP101 ^ tankB102 → source0.pipe
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → sink0.volumeFlowRate.port
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → source0.pipe
tankB101 ^ pumpP101 ^ tankB102 → source0.source.ports[1].m
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → sink0.pipe.flowModel.m
pumpP101 ^ tankB102 → source0.volumeFlowRate.port
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → source0.port
tankB101 ^ pumpP101 ^ tankB102 → sink0.port
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → sink0.sink.ports[1].m
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → sink0.port
tankB101 ^ pumpP101 ^ tankB102 → source0.volumeFlowRate.port
tankB101 ^ pumpP101 ^ tankB102 → sink0.sink.ports[1].m
tankB101 ^ pumpP101 ^ tankB102 → sink0.pipe.flowModel.m
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → sink0.pipe.port
pumpP101 ^ tankB102 → sink0.sink.ports[1].m
pumpP101 ^ tankB102 → sink0.port
tankB101 ^ pumpP101 ^ tankB102 → filter
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → source0.source.ports[1].m
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → filter
pumpP101 ^ tankB102 → filter
tankB101 ^ pumpP101 ^ tankB102 → sink0.volumeFlowRate.port
pumpP101 ^ tankB102 → source0.pipe
pumpP101 ^ tankB102 → source0.source.ports[1].m
tankB101 ^ pumpP101 ^ filterF101 ^ valveDiscrete ^ valveV101 ^ tankB102 → source0.volumeFlowRate.port
pumpP101 ^ tankB102 → source0.port
```

1. System Descriptions of the investigated Systems

System description of running DDGD on system Distill

```

pump ^ tankB101 → sink1.sink.ports[1].m
pump ^ tankB101 ^ tankB102 → source0.port
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → der(still)
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → source0.volumeFlowRate.port
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → source0.source.ports[1].m
pump ^ tankB101 ^ tankB102 → sink1.port
pump ^ tankB101 → sink0.sink.ports[1].m
pump ^ tankB101 → sink1.volumeFlowRate.port
pump ^ tankB101 ^ tankB102 → still
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink1.volumeFlowRate.port
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink0.port
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink0.pipe.port
pump ^ tankB101 ^ tankB102 → sink1.volumeFlowRate.port
pump ^ tankB101 ^ tankB102 → sink0.sink.ports[1].m
pump ^ tankB101 → sink0.port
pump ^ tankB101 ^ tankB102 → sink0.port
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink1.port
pump ^ tankB101 ^ tankB102 → source0.pipe
pump ^ tankB101 ^ tankB102 → source0.source.ports[1].m
pump ^ tankB101 → sink0.pipe.flowModel.m
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → source0.port
pump ^ tankB101 ^ tankB102 → source0.volumeFlowRate.port
pump ^ tankB101 → sink0.pipe.port
pump ^ tankB101 → source0.port
pump ^ tankB101 ^ tankB102 → sink0.volumeFlowRate.port
pump ^ tankB101 → source0.source.ports[1].m
pump ^ tankB101 → sink0.volumeFlowRate.port
pump ^ tankB101 ^ tankB102 → sink0.pipe.port
pump ^ tankB101 → still
pump ^ tankB101 → sink1.port
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink0.pipe.flowModel.m
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink1.sink.ports[1].m
pump ^ tankB101 → sink1.pipe.port
pump ^ tankB101 → sink1.pipe.flowModel.m
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink1.pipe.port
pump ^ tankB101 ^ tankB102 → sink1.pipe.flowModel.m
pump ^ tankB101 → source0.volumeFlowRate.port
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → still
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink1.pipe.flowModel.m
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink0.sink.ports[1].m
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → sink0.volumeFlowRate.port
pump ^ tankB101 ^ tankB102 → sink1.sink.ports[1].m
pump ^ tankB101 → der(still)
pump ^ tankB101 ^ tankB102 → der(still)
pump ^ tankB101 ^ tankB102 ^ valveDiscrete ^ valveV101 ^ V101 ^ V102 ^ outputReservoir → source0.pipe
pump ^ tankB101 ^ tankB102 → sink0.pipe.flowModel.m
pump ^ tankB101 ^ tankB102 → sink1.pipe.port
pump ^ tankB101 → source0.pipe
pump ^ tankB101 → sink0.pipe
```

System description of running DDGD on system Mixer

```
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → mixer
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source2.source.ports[1].m
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source1.source.ports[1].m
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source2.source.ports[1].m
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source0.volumeFlowRate.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source1.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source1.volumeFlowRate.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.pipe.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source0.pipe
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source0.source.ports[1].m
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.pipe.flowModel.m
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source0.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source0.pipe
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source1.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → der(mixer)
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.volumeFlowRate.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source2.volumeFlowRate.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.pipe.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source2.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source1.pipe
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source1.volumeFlowRate.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source0.source.ports[1].m
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.sink.ports[1].m
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.volumeFlowRate.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.sink.ports[1].m
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → mixer
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source2.pipe
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source0.volumeFlowRate.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source0.port
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → der(mixer)
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source1.pipe
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source2.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source1.source.ports[1].m
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → source2.volumeFlowRate.port
tankB201 ^ pumpP201 ^ tankB202 ^ tankB203 ^ valveV201 ^ valveV202 ^ valveV203 → source2.pipe
pumpP201 ^ valveV201 ^ valveV202 ^ valveV203 → sink0.pipe.flowModel.m
```

1. System Descriptions of the investigated Systems

System description of running DDGD on the Tennessee Eastman Process according to Manca et al.

$vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcReactorD}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{TempFeedA}$
 $vE \wedge \text{condenser} \rightarrow \text{ConcReactorA}$
 $vC \wedge \text{condenser} \rightarrow \text{ConcPurgeD}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcUnderflowStripD}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{PressureStripper}$
 $vA \wedge vE \wedge \text{condenser} \rightarrow \text{ConcUnderflowStripG}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{FlowReactorFeed}$
 $vA \wedge vC \wedge \text{condenser} \rightarrow \text{CondFeedC}$
 $\text{condenser} \rightarrow \text{FlowD}$
 $vA \wedge vC \wedge \text{condenser} \rightarrow \text{FlowE}$
 $vE \rightarrow \text{UnderflowStripper}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{FlowStripperSteam}$
 $vE \wedge \text{condenser} \rightarrow \text{LevelStripper}$
 $vE \wedge \text{condenser} \rightarrow \text{ConcPurgeG}$
 $vE \wedge \text{condenser} \rightarrow \text{TempCoolingOutCond}$
 $vA \wedge vE \wedge vC \rightarrow \text{ConcPurgeA}$
 $vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcReactorE}$
 $vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcPurgeE}$
 $vE \wedge vC \wedge \text{condenser} \rightarrow \text{TempFeedE}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{TempCoolingInletCondenser}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{PressureSeparator}$
 $vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcUnderflowStripH}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcReactorB}$
 $\text{condenser} \rightarrow \text{FlowCRC}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{UnderflowSeparator}$
 $\text{condenser} \rightarrow \text{FlowCoolingInletCondenser}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{PressureReactor}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcReactorC}$
 $vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcPurgeB}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{FlowC}$
 $vE \wedge \text{condenser} \rightarrow \text{TempSeparator}$
 $vE \wedge \text{condenser} \rightarrow \text{tempCoolingOutR}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{ConcPurgeC}$
 $vE \wedge \text{condenser} \rightarrow \text{TempStripper}$
 $vC \wedge \text{condenser} \rightarrow \text{ConcUnderflowStripF}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{CondFeedC}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{CondFeedD}$
 $vE \rightarrow \text{LevelSeparator}$
 $vE \wedge \text{condenser} \rightarrow \text{ConcPurgeH}$
 $\text{condenser} \rightarrow \text{FlowCoolingInletReactor}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{CondFeedA}$
 $vA \wedge vE \wedge vC \wedge \text{condenser} \rightarrow \text{CondFeedE}$
 $vE \wedge \text{condenser} \rightarrow \text{CompressorWork}$
 $vA \wedge vC \rightarrow \text{CondFeedC}$
 $vA \wedge vE \wedge \text{condenser} \rightarrow \text{FlowPurge}$

System description of running DDGD on system S_3 (i)

```
valve1 → pipe6.port
tank2 → pipe1.port
pump1 ∧ valve2 → pipe7.flowModel.m
tank2 → pipe7.port
tank2 → tank2.level
tank1 → pipe2.flowModel.m
tank2 → pipe6.port
valve3 → pipe2.port
pump1 ∧ valve2 → valveLinear3.port
tank2 → valveLinear3.port
tank1 → pipe6.flowModel.m
valve1 → valveLinear3.m
valve1 → sink.ports[2].m
pump1 ∧ valve2 → sink.ports[2].m
tank2 → pipe2.flowModel.m
valve3 → tank.ports[2].m
valve3 → pipe1.flowModel.m
tank1 → pipe6.port
tank2 → valveLinear1.m
pump1 ∧ valve2 → valveLinear1.m
pump1 ∧ valve2 → valveLinear1.port
valve3 → valveLinear1.port
tank1 → valveLinear3.m
pump1 ∧ valve2 → pump1.V
tank1 → pipe1.port
pump1 ∧ valve2 → valveLinear3.m
pump1 ∧ valve2 → der(tank2.level)
tank1 → pipe7.flowModel.m
pump1 ∧ valve2 → pipe1.flowModel.m
pump1 ∧ valve2 → pump1.port
tank2 → pipe7.flowModel.m
valve3 → tank2.level
valve3 → tank1.ports[2].m
tank2 → valveLinear1.port
pump1 ∧ valve2 → pipe7.port
pump1 ∧ valve2 → tank.ports[1].m
pump1 ∧ valve2 → tank2.level
valve1 → valveLinear3.port
valve1 → pipe7.flowModel.m
```

1. System Descriptions of the investigated Systems

System description of running DDGD on system S₃ (ii)

tank1 → *pipe7.port*
valve1 → *tank2.ports[2].m*
tank1 → *tank1.ports[2].m*
valve1 → *pipe6.flowModel.m*
valve1 → *pipe7.port*
tank1 → *sink.ports[2].m*
tank2 → *pipe2.port*
tank1 → *valveLinear1.port*
valve1 → *tank1.level*
tank1 → *tank2.ports[2].m*
pump1 ∧ *valve2* → *pipe2.flowModel.m*
tank2 → *tank1.ports[2].m*
tank1 → *tank.ports[2].m*
pump1 ∧ *valve2* → *tank.ports[2].m*
valve1 → *tank.level*
tank2 → *sink.ports[2].m*
tank2 → *pipe1.flowModel.m*
valve3 → *valveLinear1.m*
tank1 → *valveLinear1.m*
pump1 ∧ *valve2* → *pipe6.port*
valve1 → *tank2.level*
pump1 ∧ *valve2* → *pipe6.flowModel.m*
pump1 ∧ *valve2* → *pump1.m*
tank2 → *tank2.ports[2].m*
pump1 ∧ *valve2* → *tank2.ports[2].m*
valve3 → *pipe2.flowModel.m*
tank1 → *pipe1.flowModel.m*
pump1 ∧ *valve2* → *der(tank.level)*
tank2 → *tank.ports[2].m*
tank1 → *tank1.level*
pump1 ∧ *valve2* → *tank.level*
valve3 → *pipe1.port*
tank1 → *pipe2.port*
tank2 → *valveLinear3.m*
pump1 ∧ *valve2* → *pipe1.port*
pump1 ∧ *valve2* → *pipe2.port*
pump1 ∧ *valve2* → *valveLinear3.V*
pump1 ∧ *valve2* → *tank1.ports[2].m*
tank1 → *tank2.level*
tank1 → *valveLinear3.port*
tank2 → *pipe6.flowModel.m*

System description of running DDGD on system S4 (i)

```
valve3 → valveLinear5.V
valve6 → tank4.ports[2].m
valve4 ∧ pipe4 → tank2.ports[3].m
tank2 → tank4.ports[3].m
valve6 → pipe5.port
valve4 ∧ pipe4 → valveLinear5.m
valve4 ∧ pipe4 → valveLinear5.V
tank2 → tank4.ports[1].m
valve6 → overflow4.port
valve3 → volumeFlowRate
tank2 → tank2.level
valve4 ∧ pipe4 → overflow2.flowModel.m
valve4 ∧ pipe4 → sink.ports[2].m
tank2 → pipe6.port
valve3 → pipe2.port
tank2 → valveLinear6.m
valve6 → valveLinear5.m
valve6 → valveLinear5.port
valve3 → valveLinear5.port
valve3 → tank1.level
valve6 → valveDiscrete1.m
tank2 → valveLinear6.V
valve3 → tank4.ports[1].m
valve3 → valveLinear6.m
valve3 → tank2.ports[2].m
valve6 → tank4.ports[1].m
valve3 → tank2.ports[1].m
valve4 ∧ pipe4 → tank3.level
valve4 ∧ pipe4 → volumeFlowRate2.port
valve6 → tank4.ports[4].m
tank2 → valveDiscrete1.V
tank2 → pipe5.flowModel.m
valve6 → tank4.level
valve6 → volumeFlowRate4.port
valve6 → volumeFlowRate4.V
valve3 → pipe6.flowModel.m
tank2 → valveLinear4.port
tank2 → valveLinear5.m
valve3 → tank3.level
valve4 ∧ pipe4 → volumeFlowRate
valve4 ∧ pipe4 → pipe5.flowModel.m
valve4 ∧ pipe4 → tank4.ports[2].m
valve4 ∧ pipe4 → overflow2.port
valve6 → valveLinear4.port
valve6 → overflow4.flowModel.m
valve3 → tank4.level
valve3 → tank3.ports[2].m
valve3 → tank2.level
tank2 → pipe5.port
valve3 → tank1.ports[2].m
tank2 → valveLinear5.port
valve4 ∧ pipe4 → tank3.ports[2].m
valve3 → valveLinear6.port
valve6 → tank2.ports[2].m
valve3 → der(tank4.level)
valve3 → valveLinear4.m
valve3 → valveLinear5.m
valve3 → sink.ports[1].m
tank2 → pipe4.port
```

1. System Descriptions of the investigated Systems

System description of running DDGD on system S₄ (ii)

```
tank2 → pipe4.port
tank2 → valveLinear4.m
valve4 ∧ pipe4 → valveLinear5.port
valve6 → pipe4.port
tank2 → valveLinear5.V
valve4 ∧ pipe4 → sink.ports[4].m
valve6 → tank2.level
valve4 ∧ pipe4 → volumeFlowRate4.V
valve3 → der(tank2.level)
valve6 → der(tank3.level)
valve3 → valveDiscrete1.V
valve6 → valveDiscrete1.V
tank2 → valveLinear6.port
valve4 ∧ pipe4 → volumeFlowRate2.V
valve6 → valveDiscrete1.port
valve3 → pipe6.port
valve6 → valveLinear5.V
valve3 → valveLinear4.V
valve4 ∧ pipe4 → tank4.ports[4].m
valve6 → pipe4.flowModel.m
valve6 → der(tank2.level)
valve6 → tank3.level
valve6 → sink.ports[2].m
valve4 ∧ pipe4 → pipe5.port
valve6 → pipe5.flowModel.m
valve3 → tank4.ports[3].m
valve3 → valveLinear4.port
valve3 → valveDiscrete1.port
tank2 → sink.ports[1].m
valve3 → pipe4.port
valve6 → valveLinear4.V
valve3 → valveLinear2.V
tank2 → pipe4.flowModel.m
tank2 → tank2.ports[2].m
valve4 ∧ pipe4 → overFlow4.flowModel.m
valve4 ∧ pipe4 → tank2.level
valve4 ∧ pipe4 → overFlow4.port
tank2 → tank4.ports[2].m
valve3 → pipe2.flowModel.m
tank2 → valveDiscrete1.m
valve6 → valveLinear4.m
valve3 → valveDiscrete1.m
valve4 ∧ pipe4 → volumeFlowRate4.port
tank2 → pipe6.flowModel.m
valve4 ∧ pipe4 → tank4.level
valve3 → pipe5.flowModel.m
tank2 → tank3.level
valve3 → valveLinear6.V
tank2 → volumeFlowRate
valve3 → tank4.ports[2].m
tank2 → valveDiscrete1.port
valve3 → pipe5.port
valve6 → tank3.ports[2].m
valve6 → volumeFlowRate
tank2 → tank3.ports[2].m
valve3 → valveLinear2.m
valve3 → valveLinear2.port
tank2 → valveLinear4.V
valve3 → pipe4.flowModel.m
```

Shortened system description of running DDRC on system S₃ with correlation threshold 0.9

```
overflow1.flowModel.Is[1] ^ overflow1.flowModel.vs[1] ^ overflow1.flowModel.vs[2] → overflow1.port_b.m_flow
overflow2.flowModel.Is[1] ^ overflow2.flowModel.vs[1] ^ overflow2.flowModel.vs[2] → overflow2.port_b.m_flow
overflow3.flowModel.Is[1] ^ overflow3.flowModel.vs[1] ^ overflow3.flowModel.vs[2] → overflow3.port_b.m_flow
overflow4.flowModel.Is[1] ^ overflow4.flowModel.vs[1] ^ overflow4.flowModel.vs[2] → overflow4.port_b.m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) ^ der(tank3.V) ^ der(tank3.m) → pipe4.port_b.m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) → pipe5.port_b.m_flow
overflow1.flowModel.Is[1] ^ overflow1.flowModel.vs[1] ^ overflow1.flowModel.vs[2] → tank1.ports[5].m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) ^ der(tank3.V) ^ der(tank3.m) → tank2.ports[2].m_flow
overflow2.flowModel.Is[1] ^ overflow2.flowModel.vs[1] ^ overflow2.flowModel.vs[2] → tank2.ports[3].m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) ^ der(tank3.V) → tank3.ports[2].m_flow
overflow3.flowModel.Is[1] ^ overflow3.flowModel.vs[1] ^ overflow3.flowModel.vs[2] → tank3.ports[3].m_flow
overflow4.flowModel.Is[1] ^ overflow4.flowModel.vs[1] ^ overflow4.flowModel.vs[2] → tank4.ports[4].m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) ^ der(tank3.V) ^ der(tank3.m) → valveLinear4.port_b.m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) → valveLinear5.port_b.m_flow
overflow1.flowModel.Is[1] ^ overflow1.flowModel.vs[1] ^ overflow1.flowModel.vs[2] → volumeFlowRate1.port_b.m_flow
overflow2.flowModel.Is[1] ^ overflow2.flowModel.vs[1] ^ overflow2.flowModel.vs[2] → volumeFlowRate2.port_b.m_flow
overflow3.flowModel.Is[1] ^ overflow3.flowModel.vs[1] ^ overflow3.flowModel.vs[2] → volumeFlowRate3.port_b.m_flow
overflow4.flowModel.Is[1] ^ overflow4.flowModel.vs[1] ^ overflow4.flowModel.vs[2] → volumeFlowRate4.port_b.m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) ^ der(tank3.V) ^ der(tank3.m) → volumeFlowRate_tank2_tank4.port_b.m_flow
der(tank2.U) ^ der(tank2.V) ^ der(tank2.m) ^ der(tank3.U) ^ der(tank3.V) → volumeFlowRate_tank3_tank4.port_b.m_flow
```


2. Code Availability

Figure 14 presents the architecture of the Python code used for the evaluation. The code itself is available under <https://github.com/DaemonNet/hysd>. To run the code, ensure that the associated data is located in the necessary file hierarchy. The data can be obtained from within the respective benchmarks [82, 83] and needs to be in the format of comma-separated-value files. The entry point to the program is file `Executor.py`. The file loads the data and, depending on its configuration, runs algorithms DDRC and DDGD to generate system descriptions, or uses an expert-defined system description that must be specified by the user. The system descriptions are then passed to HySD or DDA-IM and `HySD_simple`. The three diagnosis algorithms are summarised in the figure under the label HySD. All intermediate and final files are located in the folder *Output* upon successful execution. A summary is provided in the output file *diag.txt*.

It should be noted that generating system descriptions automatically may take a long time as the data sets contain several gigabytes of data.

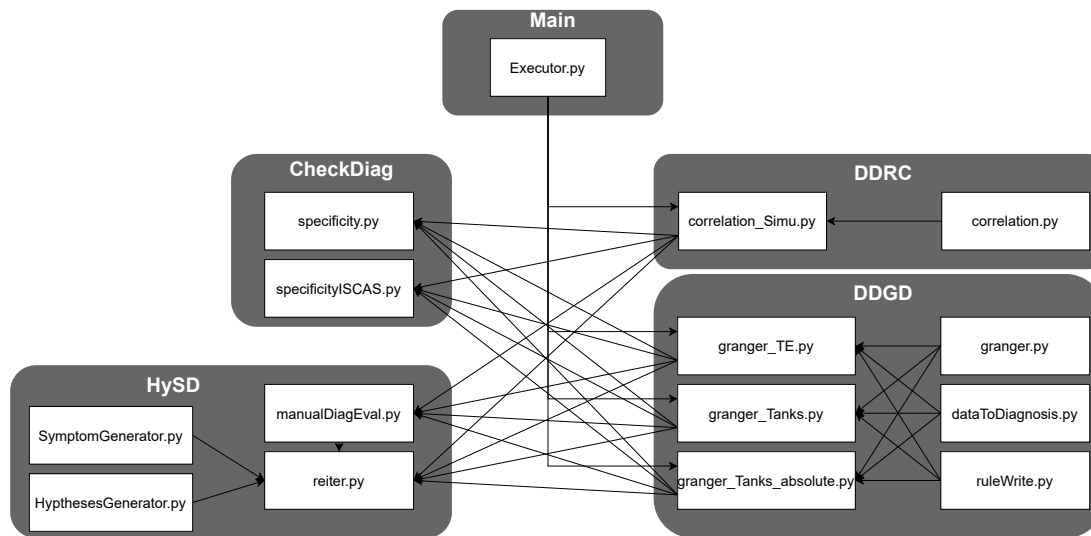


Figure 14.: Architecture of the evaluation system. Black boxes denote the respective algorithms and white boxes denote the respective source files. Arrows show the dependencies.

Bibliography

- [1] Christos Koulamas and Athanasios Kalogeras. "Cyber-physical systems and digital twins in the industrial internet of things [cyber-physical systems]". In: *Computer* 51.11 (2018), pp. 95–98 (cit. on p. 3).
- [2] Mariagrazia Dotoli et al. "An overview of current technologies and emerging trends in factory automation". In: *International Journal of Production Research* 57.15-16 (2019), pp. 5047–5067 (cit. on p. 3).
- [3] Venkat Venkatasubramanian et al. "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods". In: *Computers & chemical engineering* 27.3 (2003), pp. 293–311 (cit. on p. 3).
- [4] Venkat Venkatasubramanian, Raghunathan Rengaswamy, and Surya N Kavuri. "A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies". In: *Computers & chemical engineering* 27.3 (2003), pp. 313–326 (cit. on p. 3).
- [5] Venkat Venkatasubramanian et al. "A review of process fault detection and diagnosis: Part III: Process history based methods". In: *Computers & chemical engineering* 27.3 (2003), pp. 327–346 (cit. on p. 3).
- [6] Julius Pfrommer, Thomas Usländer, and Jürgen Beyerer. "KI-Engineering–AI Systems Engineering". In: *at-Automatisierungstechnik* 70.9 (2022), pp. 756–766 (cit. on p. 3).
- [7] "ARTEMIS Strategic Research Agenda". In: (2016) (cit. on pp. 3, 9, 101).
- [8] Die Bundesregierung. "Strategie Künstliche Intelligenz der Bundesregierung". In: (2018) (cit. on p. 3).
- [9] Jacobs, J.C. and Kagermann, H. and Spath, D. "The Future of Work in the Digital Transformation ". In: (2018) (cit. on p. 3).
- [10] "Electronic Components and Systems Strategic Research Agenda". In: (2018) (cit. on p. 3).
- [11] VDI/VDE Gesellschaft Mess- und Automatisierungstechnik. "Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation". In: (2013) (cit. on p. 3).
- [12] Hamed Khorasgani, Gautam Biswas, and Daniel Jung. "Structural methodologies for distributed fault detection and isolation". In: *Applied Sciences* 9.7 (2019), p. 1286 (cit. on pp. 3, 4, 7–9, 37, 51, 57).
- [13] Meir Kalech, Roni Stern, and Ester Lazebnik. "Minimal Cardinality Diagnosis in Problems with Multiple Observations". In: *Diagnostics* 11.5 (2021), p. 780 (cit. on pp. 3, 8, 39, 49).
- [14] Peter Struss and Benjamin Ertl. "Diagnosis of bottling plants—first success and challenges". In: *20th International Workshop on Principles of Diagnosis*. 2009, pp. 83–90 (cit. on pp. 3, 9, 55).

- [15] Daniel G Bobrow and Jack Whalen. "Community knowledge sharing in practice: the Eureka story". In: *Reflections: The SoL Journal* 4.2 (2002), pp. 47–59 (cit. on pp. 3, 18, 131, 132).
- [16] Alexander Diedrich, Patrick Deutschmann, and Caroline Junker. "ServiceNavigator - A Bayesian Assistance System for Diagnosing Industrial Production Systems". In: *2022 5th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)[submitted]*. IEEE. 2022 (cit. on pp. 3, 18, 55, 132).
- [17] Raymond Reiter. "A theory of diagnosis from first principles". In: *Artificial intelligence* 32.1 (1987), pp. 57–95 (cit. on pp. 3, 4, 6, 12, 37, 39–42, 48, 54, 70, 96).
- [18] Gerald J Sussman. "A computational model of skill acquisition". In: (1973) (cit. on pp. 3, 6, 47).
- [19] Johan De Kleer and Brian C Williams. "Diagnosing multiple faults". In: *Artificial intelligence* 32.1 (1987), pp. 97–130 (cit. on pp. 4, 6, 9, 41, 42, 48, 66, 96).
- [20] Roni Stern, Meir Kalech, and Orel Elimelech. "Hierarchical diagnosis in strong fault models". In: *Twenty Fifth International Workshop on Principles of Diagnosis*. 2014 (cit. on pp. 4, 9, 56, 66, 96, 115).
- [21] Alexander Feldman, Gregory Provan, and Arjan van Gemund. "The Lydia approach to combinational model-based diagnosis". In: *Proc. DX* 9 (2009), pp. 403–408 (cit. on p. 4).
- [22] Alexander Feldman, Gregory Provan, and Arjan Van Gemund. "Approximate model-based diagnosis using greedy stochastic search". In: *Journal of Artificial Intelligence Research* 38 (2010), pp. 371–413 (cit. on pp. 4, 10, 39, 43, 49, 66, 122).
- [23] Xiangfu Zhao et al. "TreeMerge: Efficient Generation of Minimal Hitting-Sets for Conflict Sets in Tree Structure for Model-Based Fault Diagnosis". In: *IEEE Transactions on Reliability* (2021) (cit. on pp. 4, 8, 49).
- [24] Rolf Isermann and Peter Balle. "Trends in the application of model-based fault detection and diagnosis of technical processes". In: *Control engineering practice* 5.5 (1997), pp. 709–719 (cit. on pp. 4, 38).
- [25] P.J. Mosterman and G. Biswas. "Diagnosis of continuous valued systems in transient operating regions". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 29.6 (Nov. 1999), pp. 554–565. ISSN: 1083-4427, 1558-2426. DOI: 10.1109/3468.798059 (cit. on pp. 4, 35, 50, 52, 57, 75).
- [26] Johan De Kleer. "How circuits work". In: *Artificial intelligence* 24.1-3 (1984), pp. 205–280 (cit. on p. 4).
- [27] Kenneth D Forbus. "Qualitative process theory". In: *Artificial intelligence* 24.1-3 (1984), pp. 85–168 (cit. on pp. 4, 39, 52).
- [28] Tolga Kurtoglu and Irem Y Tumer. "A graph-based fault identification and propagation framework for functional design of complex systems". In: (2008) (cit. on p. 4).
- [29] Arthur Schopenhauer. *Ueber die vierfache Wurzel des Satzes vom zureichenden Grunde*. FA Brockhaus, 1864 (cit. on p. 4).
- [30] Drew McDermott and Jon Doyle. "Non-monotonic logic I". In: *Artificial intelligence* 13.1-2 (1980), pp. 41–72 (cit. on pp. 4, 31).

- [31] Drew McDermott. "Nonmonotonic logic II: Nonmonotonic modal theories". In: *Journal of the ACM (JACM)* 29.1 (1982), pp. 33–57 (cit. on pp. 4, 31).
- [32] Johan De Kleer and John Seely Brown. "A qualitative physics based on confluences". In: *Artificial intelligence* 24.1-3 (1984), pp. 7–83 (cit. on pp. 4, 52, 56, 89, 90, 94).
- [33] Kenneth D Forbus. "Qualitative Process Theory." en. In: *Artificial Intelligence* (1984), p. 187 (cit. on pp. 4, 37).
- [34] Patrick Rodler. "Memory-limited model-based diagnosis". In: *Artificial Intelligence* (2022), p. 103681 (cit. on pp. 4, 8, 10, 41, 49).
- [35] Patrick Rodler. "How should I compute my candidates? A taxonomy and classification of diagnosis computation algorithms". In: *33rd International Workshop on Principle of Diagnosis – DX 2022*. 2022 (cit. on pp. 4, 6).
- [36] Alexey Ignatiev. "Towards Trustable Explainable AI." In: *IJCAI*. 2020, pp. 5154–5158 (cit. on p. 5).
- [37] Alexander Maier. "Online passive learning of timed automata for cyber-physical production systems". In: *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*. IEEE. 2014, pp. 60–66 (cit. on pp. 5, 50, 58, 128).
- [38] Marie-Odile Cordier et al. "A comparative analysis of AI and control theory approaches to model-based diagnosis". In: *ECAI*. 2000, pp. 136–140 (cit. on p. 5).
- [39] Alban Grastien. "Diagnosis of hybrid systems by consistency testing". In: *24th International Workshop on Principles of Diagnosis (DX-13)*. Citeseer. 2013, pp. 9–14 (cit. on pp. 5, 54, 56, 57).
- [40] Constanze Hasterok and Janina Stompe. "PAISE®–process model for AI systems engineering". In: *at-Automatisierungstechnik* 70.9 (2022), pp. 777–786 (cit. on p. 5).
- [41] Sriram Narasimhan and Gautam Biswas. "Model-based diagnosis of hybrid systems". In: *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and humans* 37.3 (2007), pp. 348–361 (cit. on pp. 5, 10, 50, 52, 108).
- [42] Hamed Khorasgani and Gautam Biswas. "Structural Fault Detection and Isolation in Hybrid Systems". In: *IEEE Transactions on Automation Science and Engineering* (2017) (cit. on pp. 5, 12, 36, 49, 51, 52, 108).
- [43] Allen L Brown. "Qualitative knowledge, causal reasoning, and the localization of failures". In: *MIT* (1974) (cit. on pp. 6, 47).
- [44] Tolga Kurtoglu et al. "First international diagnosis competition-DXC'09". In: *Proc. DX 9* (2009), pp. 383–396 (cit. on pp. 6, 109).
- [45] Alexander Feldman, Gregory Provan, and Arjan Van Gemund. "Solving strong-fault diagnostic models by model relaxation". In: *Twenty-First International Joint Conference on Artificial Intelligence*. 2009 (cit. on p. 7).
- [46] Guangquan Zhao et al. "Research advances in fault diagnosis and prognostic based on deep learning". In: *2016 Prognostics and system health management conference (PHM-Chengdu)*. IEEE. 2016, pp. 1–6 (cit. on p. 7).
- [47] Miao He and David He. "Deep learning based approach for bearing fault diagnosis". In: *IEEE Transactions on Industry Applications* 53.3 (2017), pp. 3057–3065 (cit. on p. 7).
- [48] Andreas Bunte, Benno Stein, and Oliver Niggemann. "Model-based diagnosis for cyber-physical production systems based on machine learning and residual-based diagnosis models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 2727–2735 (cit. on pp. 7, 129, 131).

- [49] Jens Otto, Birgit Vogel-Heuser, and Oliver Niggemann. "Automatic parameter estimation for reusable software components of modular and reconfigurable cyber-physical production systems in the domain of discrete manufacturing". In: *IEEE Transactions on Industrial Informatics* 14.1 (2017), pp. 275–282 (cit. on p. 7).
- [50] Peter Struss and Oskar Dressler. "' Physical Negation" Integrating Fault Models into the General Diagnostic Engine." In: *IJCAI*. Vol. 89. 1989, pp. 1318–1323 (cit. on p. 7).
- [51] Diomidis H Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. Quality Press, 2003 (cit. on p. 7).
- [52] R Feynman. "Report of the presidential commission on the space shuttle challenger accident". In: *Appendix F* (1986) (cit. on p. 7).
- [53] Nassim Nicholas Taleb. *The Black Swan*. Random House Publishing Group, 2010 (cit. on p. 7).
- [54] Judea Pearl. *Causality*. Cambridge university press, 2009 (cit. on pp. 7, 10, 32–34, 51, 53, 56, 89).
- [55] Joseph Y Halpern. *Actual causality*. MiT Press, 2016 (cit. on pp. 7, 10).
- [56] Alexander Bochman. "Actual Causality in a Logical Setting." In: *IJCAI*. 2018, pp. 1730–1736 (cit. on pp. 7, 34).
- [57] Jan Maciej Kościelny, Michał Syfert, and Paweł Wnuk. "Diagnostic Row Reasoning Method Based on Multiple-Valued Evaluation of Residuals and Elementary Symptoms Sequence". In: *Energies* 14.9 (2021), p. 2476 (cit. on pp. 8, 39, 49).
- [58] Paul J Blazek and Milo M Lin. "Explainable neural networks that simulate reasoning". In: *Nature Computational Science* 1.9 (2021), pp. 607–618 (cit. on p. 9).
- [59] Fei Tao et al. "Digital twins and cyber-physical systems toward smart manufacturing and industry 4.0: Correlation and comparison". In: *Engineering* 5.4 (2019), pp. 653–661 (cit. on pp. 9, 58).
- [60] Oliver Niggemann et al. "A Generic DigitalTwin Model for Artificial Intelligence Applications". In: *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2021, pp. 55–62 (cit. on pp. 9, 18, 19, 53, 57, 58, 63, 66).
- [61] Louise Travé-Massuyès and Teresa Escobet. "BRIDGE: Matching Model-Based Diagnosis from FDI and DX Perspectives". In: *Fault Diagnosis of Dynamic Systems*. Springer, 2019, pp. 153–175 (cit. on pp. 9, 49, 56).
- [62] Johan De Kleer and James Kurien. "Fundamentals of model-based diagnosis". In: *IFAC Proceedings Volumes* 36.5 (2003), pp. 25–36 (cit. on pp. 9, 10).
- [63] Sriram Narasimhan and Lee Brownston. "HyDE-a general framework for stochastic and hybrid modelbased diagnosis". In: *Proc. DX* 7 (2007), pp. 162–169 (cit. on pp. 9, 129).
- [64] Alban Grastien. "Diagnosis of hybrid systems with SMT: opportunities and challenges". In: *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. IOS Press. 2014, pp. 405–410 (cit. on pp. 9, 49, 54, 56, 129).
- [65] Ingo Pill and Thomas Quaritsch. "Behavioral diagnosis of LTL specifications at operator level". In: *Twenty-Third International Joint Conference on Artificial Intelligence*. Citeseer. 2013 (cit. on pp. 9, 128, 131).

- [66] A Prasad Sistla and Edmund M Clarke. “The complexity of propositional linear temporal logics”. In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 733–749 (cit. on p. 9).
- [67] Ruocheng Guo et al. “A survey of learning causality with data: Problems and methods”. In: *ACM Computing Surveys (CSUR)* 53.4 (2020), pp. 1–37 (cit. on p. 10).
- [68] Zhiwei Gao, Carlo Cecati, and Steven X Ding. “A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches”. In: *IEEE Transactions on Industrial Electronics* 62.6 (2015), pp. 3757–3767 (cit. on pp. 12, 36, 37, 50).
- [69] Alexander Diedrich, Alexander Maier, and Oliver Niggemann. “Model-based Diagnosis of Hybrid Systems using Satisfiability Modulo Theory”. In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19), 2019 (cit. on pp. 17, 57, 62, 77, 84, 107).
- [70] Alexander Diedrich and Oliver Niggemann. “On Residual-based Diagnosis of Physical Systems”. In: *Engineering Applications of Artificial Intelligence* 109 (2022), p. 104636. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2021.104636>. URL: <https://www.sciencedirect.com/science/article/pii/S0952197621004450> (cit. on pp. 17, 35, 39, 62, 107, 108).
- [71] Alexander Diedrich, Björn Böttcher, and Oliver Niggemann. “Exposing Design Mistakes During Requirements Engineering by Solving Constraint Satisfaction Problems to Obtain Minimum Correction Subsets.” In: *ICAART (2)*. 2016, pp. 280–287 (cit. on pp. 17, 18).
- [72] Alexander Diedrich et al. “Applying Simulated Annealing to Problems in Model-based Diagnosis”. In: (2016) (cit. on pp. 17, 18, 27, 132).
- [73] Alexander Diedrich and Oliver Niggemann. “Diagnosing Hybrid Cyber-Physical Systems using State-Space Models and Satisfiability Modulo Theory”. In: *29th International Workshop on the Principles of Diagnosis* (2018) (cit. on pp. 17, 18).
- [74] Alexander Diedrich, Kaja Balzereit, and Oliver Niggemann. “First Approaches to Automatically Diagnose and Reconfigure Hybrid Cyber-Physical Systems”. In: *ML4CPS 2020 – Machine Learning for Cyber Physical Systems Conference* (2019) (cit. on pp. 17, 18).
- [75] Alexander Diedrich and Oliver Niggemann. “Diagnosing Systems through Approximated Information [submitted]”. In: *13th Annual Conference of the Prognostics and Health Management Society* (2021) (cit. on pp. 17, 18, 62, 107).
- [76] Alexander Diedrich, Franziska Buchholz, and Oliver Niggemann. “Learning a Causal System Description for Diagnosing Physical Systems”. In: *Proceedings of the 33rd International Workshop on Principles of Diagnosis, Toulouse, France*. 2022 (cit. on pp. 18, 62, 107).
- [77] Alejandro Perdomo-Ortiz et al. “On the readiness of quantum optimization machines for industrial applications”. In: *arXiv preprint arXiv:1708.09780* (2017) (cit. on pp. 18, 132).
- [78] Alejandro Perdomo-Ortiz et al. “Readiness of quantum optimization machines for industrial applications”. In: *Physical Review Applied* 12.1 (2019), p. 014004 (cit. on pp. 18, 19, 132).

- [79] Andreas Bunte, Alexander Diedrich, and Oliver Niggemann. "Semantics enable standardized user interfaces for diagnosis in modular production systems". In: *International Workshop on the Principles of Diagnosis (DX)*. 2016 (cit. on pp. 18, 19).
- [80] Andreas Bunte, Alexander Diedrich, and Oliver Niggemann. "Integrating semantics for diagnosis of manufacturing systems". In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2016, pp. 1–8 (cit. on pp. 18, 19).
- [81] Kaja Balzereit et al. "Generating Causal Hypotheses for Explaining Black-Box Industrial Processes". In: *2023 5th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2022 (cit. on pp. 18, 19, 78).
- [82] Kaja Balzereit et al. "An Ensemble of Benchmarks for the Evaluation of AI Methods for Fault Handling in CPPS". In: *19th IEEE International Conference on Industrial Informatics*. Nov. 2021 (cit. on pp. 18, 19, 107, 108, 113, 147).
- [83] Jonas Ehrhardt et al. "An AI benchmark for Diagnosis, Reconfiguration & Planning". In: *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2012 (cit. on pp. 18, 19, 108, 147).
- [84] Leonardo de Moura and Nikolaj Bjørner. "Satisfiability modulo theories: An appetizer". In: *Brazilian Symposium on Formal Methods*. Springer. 2009, pp. 23–36 (cit. on pp. 23, 28).
- [85] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010 (cit. on pp. 23, 25, 30).
- [86] Richard M Karp. "On the computational complexity of combinatorial problems". In: *Networks* 5.1 (1975), pp. 45–68 (cit. on p. 24).
- [87] Samuel Kolb et al. "Learning SMT (LRA) constraints using SMT solvers". In: *IJ-CAI International Joint Conference on Artificial Intelligence*. Vol. 2018. ijcai.org. 2018, pp. 2333–2340 (cit. on pp. 28, 53, 131).
- [88] Daniel Jung and Christofer Sundström. "A Combined Data-Driven and Model-Based Residual Selection Algorithm for Fault Detection and Isolation". In: *IEEE Transactions on Control Systems Technology* 27.2 (Mar. 2019), pp. 616–630. ISSN: 1063-6536, 1558-0865, 2374-0159. DOI: 10.1109/TCST.2017.2773514 (cit. on pp. 30, 37, 51).
- [89] Raymond Reiter. "A logic for default reasoning". In: *Artificial intelligence* 13.1-2 (1980), pp. 81–132 (cit. on p. 31).
- [90] Jack Minker. "An overview of nonmonotonic reasoning and logic programming". In: *The Journal of Logic Programming* 17.2-4 (1993), pp. 95–126 (cit. on pp. 31, 47).
- [91] Antonis C Kakas, Robert A. Kowalski, and Francesca Toni. "Abductive logic programming". In: *Journal of logic and computation* 2.6 (1992), pp. 719–770 (cit. on p. 31).
- [92] Tim Finin and Gary Morris. "Abductive reasoning in multiple fault diagnosis". In: *Artificial Intelligence Review* 3.2-3 (1989), pp. 129–158 (cit. on pp. 31, 48).
- [93] Nassim Nicholas Taleb. "Antifragile: Things that gain from disorder". In: vol. 3. Random House, 2012, p. 199 (cit. on pp. 32, 83, 131).
- [94] Amjad Ibrahim et al. "Practical causal models for cyber-physical systems". In: *NASA Formal Methods: 11th International Symposium, NFM 2019, Houston, TX, USA, May 7–9, 2019, Proceedings* 11. Springer. 2019, pp. 211–227 (cit. on p. 32).

- [95] Joseph Y Halpern. "A modification of the Halpern-Pearl definition of causality". In: *arXiv preprint arXiv:1505.00162* (2015) (cit. on p. 32).
- [96] Sainyam Galhotra, Romila Pradhan, and Babak Salimi. "Explaining black-box algorithms using probabilistic contrastive counterfactuals". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 577–590 (cit. on p. 34).
- [97] Joseph Y Halpern. "A Modification of the Halpern-Pearl Definition of Causality". en. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015), p. 12 (cit. on pp. 34, 89).
- [98] Ye Yuan et al. "Data driven discovery of cyber physical systems". In: *Nature communications* 10.1 (2019), pp. 1–9 (cit. on pp. 35, 51, 58, 83).
- [99] S. Narasimhan et al. "Building observers to address fault isolation and control problems in hybrid dynamic systems". In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.o. Vol. 4. ISSN: 1062-922X. Oct. 2000, 2393-2398 vol.4. DOI: 10.1109/ICSMC.2000.884349* (cit. on p. 36).
- [100] Amit Metodi et al. "A novel sat-based approach to model based diagnosis". In: *Journal of Artificial Intelligence Research* 51 (2014), pp. 377–411 (cit. on pp. 39, 43, 65, 116).
- [101] Johan De Kleer and John Seely Brown. "Theories of causal ordering". In: *Artificial intelligence* 29.1 (1986), pp. 33–61 (cit. on pp. 41, 52).
- [102] John McCarthy. "Artificial intelligence, logic and formalizing common sense". In: *Philosophical logic and artificial intelligence*. Springer, 1989, pp. 161–190 (cit. on pp. 42, 48).
- [103] Brian C Williams and Robert J Ragno. "Conflict-directed A* and its role in model-based embedded systems". In: *Discrete Applied Mathematics* 155.12 (2007), pp. 1562–1595 (cit. on pp. 42, 49).
- [104] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107 (cit. on p. 42).
- [105] Arend Rensink. "Representing first-order logic using graphs". In: *International Conference on Graph Transformation*. Springer. 2004, pp. 319–335 (cit. on p. 44).
- [106] Marc Hallin and Madan L Puri. "Rank tests for time series analysis a survey". In: (1991) (cit. on p. 44).
- [107] Steven L Bressler and Anil K Seth. "Wiener–Granger causality: a well established methodology". In: *Neuroimage* 58.2 (2011), pp. 323–329 (cit. on pp. 45, 83).
- [108] Ali Shojaie and Emily B Fox. "Granger causality: A review and recent advances". In: *Annual Review of Statistics and Its Application* 9 (2022), pp. 289–319 (cit. on p. 45).
- [109] Bethany Lusch, Pedro D Maia, and J Nathan Kutz. "Inferring connectivity in networked dynamical systems: Challenges using Granger causality". In: *Physical Review E* 94.3 (2016), p. 032220 (cit. on pp. 45, 53, 83).
- [110] John McCarthy. *Programs with common sense*. 1959 (cit. on p. 47).
- [111] Raymond Reiter. "Nonmonotonic reasoning". In: *Exploring artificial intelligence*. Elsevier, 1988, pp. 439–481 (cit. on p. 47).

- [112] Jon Doyle. "A truth maintenance system". In: *Artificial intelligence* 12.3 (1979), pp. 231–272 (cit. on p. 47).
- [113] Johan De Kleer. "An assumption-based TMS". In: *Artificial intelligence* 28.2 (1986), pp. 127–162 (cit. on p. 47).
- [114] Alexander Bochman. *A logical theory of causality*. MIT Press, 2021 (cit. on p. 47).
- [115] Serge Sonfack Souchio, Laurent Geneste, and Bernard Kamsu Foguem. "Combining expert-based beliefs and answer sets". In: *Applied Intelligence* (2022), pp. 1–12 (cit. on p. 47).
- [116] Johan De Kleer. *Local Methods for Localizing Faults in Electronic Circuits*. Tech. rep. Massachusetts Institute of Technology, Cambridge Artificial Intelligence Laboratory, 1976 (cit. on p. 48).
- [117] Stephen Smale. "On the mathematical foundations of electrical circuit theory". In: *The Collected Papers of Stephen Smale: Volume 2*. World Scientific, 2000, pp. 951–968 (cit. on p. 48).
- [118] Johan De Kleer and Brian C Williams. "Diagnosis with Behavioral Modes." In: *IJCAI*. Vol. 89. 1989, pp. 1324–1330 (cit. on p. 48).
- [119] Johan De Kleer. "Diagnosing intermittent faults". In: *18th International Workshop on Principles of Diagnosis*. 2007, pp. 45–51 (cit. on p. 48).
- [120] Johan De Kleer. "Diagnosing Multiple Persistent and Intermittent Faults." In: *IJCAI*. 2009, pp. 733–738 (cit. on p. 48).
- [121] David Poole. "Normality and Faults in Logic-Based Diagnosis." In: *IJCAI*. Vol. 89. Citeseer. 1989, pp. 1304–1310 (cit. on p. 48).
- [122] Francesco Dall’Occo, Marcello Dalpasso, and Michele Favalli. "Techniques for SAT-Based Boolean Reasoning on Multiple Faults Affecting Logic Cells and Interconnects in Digital ICs". In: *Electronics* 11.3 (2022), p. 382 (cit. on p. 49).
- [123] Alexander Feldman et al. "A SAT-Based Encoding for Finding a Minimal Automated Test Pattern Generation Test-Suite". In: () (cit. on p. 49).
- [124] Roni Stern et al. "How many diagnoses do we need?" In: *Artificial Intelligence* 248 (2017), pp. 26–45 (cit. on p. 49).
- [125] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. "Model Based Diagnosis of Multiple Observations with Implicit Hitting Sets". In: *arXiv preprint arXiv:1707.01972* (2017) (cit. on p. 49).
- [126] Alexey Ignatiev et al. "Model-Based Diagnosis with Multiple Observations." In: *IJCAI*. 2019, pp. 1108–1115 (cit. on p. 49).
- [127] Ion Matei et al. "A Hybrid Qualitative and Quantitative Diagnosis Approach". In: *Annual Conference of the PHM Society*. Vol. 11. 1. 2019 (cit. on pp. 49, 52, 56).
- [128] Ion Matei et al. "Classification based diagnosis: Integrating partial knowledge of the physical system". In: *2019 Annual Conference of the PHM Society, Scottsdale, USA: PHM Society. doi*. Vol. 10. 2019 (cit. on p. 49).
- [129] Dean Cazes and Meir Kalech. "Model-Based Diagnosis with Uncertain Observations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 03. 2020, pp. 2766–2773 (cit. on p. 49).

- [130] Edi Muškardin, Ingo Pill, and Franz Wotawa. "CatIO-A Framework for Model-Based Diagnosis of Cyber-Physical Systems". In: *International Symposium on Methodologies for Intelligent Systems*. Springer. 2020, pp. 267–276 (cit. on p. 49).
- [131] Barry Dowdeswell, Roopak Sinha, and Stephen G MacDonell. "Finding faults: A scoping study of fault diagnostics for Industrial Cyber-Physical Systems". In: *Journal of systems and software* 168 (2020), p. 110638 (cit. on p. 49).
- [132] Yigit A Yucesan, Arinan Dourado, and Felipe AC Viana. "A survey of modeling for prognosis and health management of industrial equipment". In: *Advanced Engineering Informatics* 50 (2021), p. 101404 (cit. on p. 49).
- [133] Liyu Wang et al. "Automatic modeling and fault diagnosis of car production lines based on first-principle qualitative mechanics and semantic web technology". In: *Advanced Engineering Informatics* 49 (2021), p. 101248 (cit. on p. 49).
- [134] Gregory Provan. "Model abstractions for diagnosing hybrid systems". In: *Proceedings of the 20th International Workshop on Principles of Diagnosis, DX-09, Stockholm, Sweden*. Citeseer. 2009, pp. 321–328 (cit. on p. 49).
- [135] Matthew Klenk. "Qualitative Reasoning with Modelica Models". en. In: *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)* (2014), p. 7 (cit. on p. 50).
- [136] Saif Siddique Butt, Robert Prabel, and Harald Aschemann. "Fault detection and isolation of a hybrid synchronous machine using parity equations". In: *2012 17th International Conference on Methods Models in Automation Robotics (MMAR)*. Aug. 2012, pp. 178–183. DOI: 10.1109/MMAR.2012.6347892 (cit. on pp. 50, 92).
- [137] Peter Struss. "Fundamentals of model-based diagnosis of dynamic systems". In: *IJCAI*. 1997, pp. 480–485 (cit. on p. 50).
- [138] Feng Lin. "Diagnosability of discrete event systems and its applications". In: *Discrete Event Dynamic Systems* 4.2 (1994), pp. 197–212 (cit. on p. 50).
- [139] Matthew J Daigle et al. "A comprehensive diagnosis methodology for complex hybrid systems: A case study on spacecraft power distribution systems". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 40.5 (2010), pp. 917–931 (cit. on pp. 50, 51).
- [140] Matthew Daigle, Xenofon Koutsoukos, and Gautam Biswas. "A discrete event approach to diagnosis of continuous systems". In: *Proceedings of the 18th International Workshop on Principles of Diagnosis*. 2007, pp. 259–266 (cit. on pp. 50, 56, 57).
- [141] Alban Grastien, Patrik Haslum, Sylvie Thiébaux, et al. "Conflict-Based Diagnosis of Discrete Event Systems: Theory and Practice." In: *KR*. 2012 (cit. on p. 50).
- [142] Jussi Rintanen, Alban Grastien, et al. "Diagnosability Testing with Satisfiability Algorithms." In: *IJCAI*. 2007, pp. 532–537 (cit. on p. 50).
- [143] Gautam Biswas et al. "An Approach To Mode and Anomaly Detection with Spacecraft Telemetry Data". In: *Int. J. Prognostics Health Manage* 7 (2016), pp. 1–18 (cit. on pp. 50, 57).
- [144] Steven X Ding. *Model-based fault diagnosis techniques: design schemes, algorithms, and tools*. Springer Science & Business Media, 2008 (cit. on p. 51).
- [145] Wolfgang Borutzky. "A Hybrid Bond Graph Model-based-Data Driven Method for Failure Prognostic". In: *Procedia Manufacturing* 42 (2020), pp. 188–196 (cit. on p. 51).

- [146] Gang Niu, Yajun Zhao, and Van Tung Tran. "Fault detection and isolation based on bond graph modeling and empirical residual evaluation". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 229.3 (2015), pp. 417–428 (cit. on p. 51).
- [147] Jan Lunze. "A method to get analytical redundancy relations for fault diagnosis". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 1006–1012 (cit. on pp. 51, 57).
- [148] Aadil Sarwar Khan et al. "Distributed fault detection and isolation in second order networked systems in a cyber-physical environment". In: *ISA transactions* 103 (2020), pp. 131–142 (cit. on pp. 51, 56, 57).
- [149] Maiying Zhong et al. "Event-triggered parity space approach to fault detection for linear discrete-time systems". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2021) (cit. on p. 51).
- [150] Jingsong Wu et al. "Event-Triggered Diagnostic Observer Design Using the Performance Tradeoff Approach". In: *IEEE Access* 10 (2022), pp. 17484–17494 (cit. on p. 51).
- [151] Indranil Roychoudhury et al. "Efficient simulation of hybrid systems: A hybrid bond graph approach". In: *Simulation* 87.6 (2011), pp. 467–498 (cit. on p. 51).
- [152] Om Prakash and AK Samantaray. "Model-based diagnosis and prognosis of hybrid dynamical systems with dynamically updated parameters". In: *Bond Graphs for Modelling, Control and Fault Diagnosis of Engineering Systems*. Springer, 2017, pp. 195–232 (cit. on pp. 51, 96).
- [153] Janos Gertler. "Fault detection and isolation using parity relations". In: *Control engineering practice* 5.5 (1997), pp. 653–661 (cit. on p. 51).
- [154] Richard S Mcah, Gregory M Stanley, and Dennis M Downing. "Reconciliation and rectification of process flow and inventory data". In: *Industrial & Engineering Chemistry Process Design and Development* 15.1 (1976), pp. 175–183 (cit. on p. 51).
- [155] JE Potter and MC Suman. "Thresholdless redundancy management with arrays of skewed instruments". In: *Integrity in Electronic Flight Control Systems* (1977) (cit. on p. 51).
- [156] Gustavo Perez-Zuniga et al. "Near-Optimal Decentralized Diagnosis via Structural Analysis". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2022) (cit. on p. 51).
- [157] Nathalie Verdière and Sébastien Orange. "Diagnosability and detectability of multiple faults in nonlinear models". In: *Journal of Process Control* 69 (2018), pp. 1–7 (cit. on p. 51).
- [158] Albert Oromi et al. "Robust Fault Diagnosis using a Data-based Approach and Structural Analysis". In: *Proceedings of 11th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*. 2022 (cit. on p. 51).
- [159] Arman Mohammadi, Mattias Krysander, and Daniel Jung. "Analysis of grey-box neural network-based residuals for consistency-based fault diagnosis". In: *Proceedings of 11th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*. 2022 (cit. on p. 51).
- [160] Mehdi Bayouhd, Louise Travé-Massuyes, and Xavier Olive. "Hybrid systems diagnosis by coupling continuous and discrete event techniques". In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 7265–7270 (cit. on p. 51).

- [161] Brian C Williams. "1984 - The Use of Continuity in a Qualitative Physics". en. In: *Association for the Advancement of Artificial Intelligence* (1984), p. 5 (cit. on pp. 52, 90).
- [162] Olivier Raiman. "Order of magnitude reasoning". In: *Readings in qualitative reasoning about physical systems*. Elsevier, 1990, pp. 318–322 (cit. on p. 52).
- [163] Brian C Williams and Johan de Kleer. "Qualitative reasoning about physical systems: a return to roots". In: *Artificial Intelligence* 5.1 (1991), pp. 1–9 (cit. on p. 52).
- [164] Emil Krabbe Nielsen et al. "Causality validation of multilevel flow modelling". In: *Computers & Chemical Engineering* 140 (2020), p. 106944 (cit. on p. 52).
- [165] Amin Jaber, Jiji Zhang, and Elias Bareinboim. "Causal identification under Markov equivalence". In: *Twenty-Eighth International Joint Conference on Artificial Intelligence*. 2019 (cit. on p. 52).
- [166] Bernhard Schölkopf et al. "Toward causal representation learning". In: *Proceedings of the IEEE* 109.5 (2021), pp. 612–634 (cit. on p. 52).
- [167] Miguel Martinez-Garcia et al. "Visually interpretable profile extraction with an autoencoder for health monitoring of industrial systems". In: *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*. IEEE. 2019, pp. 649–654 (cit. on p. 52).
- [168] Manuel Arias Chao et al. "Fusing physics-based and deep learning models for prognostics". In: *Reliability Engineering & System Safety* 217 (2022), p. 107961 (cit. on p. 52).
- [169] Olga Fink et al. "Potential, challenges and future directions for deep learning in prognostics and health management applications". In: *Engineering Applications of Artificial Intelligence* 92 (2020), p. 103678 (cit. on pp. 52, 56, 57).
- [170] Ji Q Wu et al. "Automated causal inference in application to randomized controlled clinical trials". In: *Nature Machine Intelligence* 4.5 (2022), pp. 436–444 (cit. on p. 52).
- [171] Zhenhui Li et al. "Discovery of causal time intervals". In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM. 2017, pp. 804–812 (cit. on p. 52).
- [172] Jonathan Laurent, Jean Yang, and Walter Fontana. "Counterfactual Resimulation for Causal Analysis of Rule-Based Models." In: *IJCAI*. 2018, pp. 1882–1890 (cit. on p. 52).
- [173] Imran Ahmed, Gwanggil Jeon, and Francesco Piccialli. "From artificial intelligence to explainable artificial intelligence in industry 4.0: a survey on what, how, and where". In: *IEEE Transactions on Industrial Informatics* 18.8 (2022), pp. 5031–5042 (cit. on p. 52).
- [174] Christian Kühnert and Jürgen Beyerer. "Data-driven methods for the detection of causal structures in process technology". In: *Machines* 2.4 (2014), pp. 255–274 (cit. on pp. 52, 129).
- [175] Ping Duan et al. "Direct causality detection via the transfer entropy approach". In: *IEEE transactions on control systems technology* 21.6 (2013), pp. 2052–2066 (cit. on p. 53).
- [176] Yoichi Chikahara and Akinori Fujino. "Causal Inference in Time Series via Supervised Learning." In: *IJCAI*. 2018, pp. 2042–2048 (cit. on p. 53).
- [177] Raha Moraffah et al. "Causal inference for time series analysis: Problems, methods and evaluation". In: *Knowledge and Information Systems* (2021), pp. 1–45 (cit. on p. 53).

- [178] Daniel Grünbaum, Maike L Stern, and Elmar W Lang. “Quantitative probing: Validating causal models using quantitative domain knowledge”. In: *arXiv preprint arXiv:2209.03013* (2022) (cit. on pp. 53, 115).
- [179] Mohit Kumar et al. “Learning MAX-SAT from contextual examples for combinatorial optimisation”. In: *Artificial Intelligence* 314 (2023), p. 103794 (cit. on p. 53).
- [180] Alex Tank, Emily B Fox, and Ali Shojaie. “Granger causality networks for categorical time series”. In: *arXiv preprint arXiv:1706.02781* (2017) (cit. on p. 53).
- [181] Alex Tank et al. “Neural granger causality”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.8 (2021), pp. 4267–4279 (cit. on pp. 53, 83).
- [182] Tao Yuan and S Joe Qin. “Root cause diagnosis of plant-wide oscillations using Granger causality”. In: *Journal of Process Control* 24.2 (2014), pp. 450–459 (cit. on p. 53).
- [183] Karim Nadim, Ahmed Ragab, and Mohamed-Salah Ouali. “Data-driven dynamic causality analysis of industrial systems using interpretable machine learning and process mining”. In: *Journal of Intelligent Manufacturing* (2023) (cit. on p. 53).
- [184] Nitin K Singh and David M Borrok. “A Granger causality analysis of groundwater patterns over a half-century”. In: *Scientific reports* 9.1 (2019), pp. 1–8 (cit. on pp. 53, 83, 131).
- [185] George Sugihara et al. “Detecting causality in complex ecosystems”. In: *science* 338.6106 (2012), pp. 496–500 (cit. on p. 53).
- [186] Judea Pearl. “The do-calculus revisited”. In: *arXiv preprint arXiv:1210.4852* (2012) (cit. on p. 53).
- [187] James F Allen. “Towards a general theory of action and time”. In: *Artificial Intelligence* 23.2 (1984), pp. 123–154 (cit. on p. 53).
- [188] Juhan Ernits and Richard Dearden. “Towards diagnosis modulo theories”. In: *22nd International Workshop on Principles of Diagnosis (DX-11)*. 2011, pp. 249–256 (cit. on p. 53).
- [189] Scott Poll et al. “Second international diagnostics competition–DXC’10”. In: *Proceedings of the 21st International Workshop on Principles of Diagnosis*. 2010 (cit. on p. 54).
- [190] Andreas Eggers, Martin Fränzle, and Christian Herde. “SAT modulo ODE: A direct SAT approach to hybrid systems”. In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2008, pp. 171–185 (cit. on p. 54).
- [191] Gregory Provan. “Abstraction-Refinement Methods for Model-Based Diagnosis”. In: *Proceedings of the Workshop on the Principles of Diagnosis* (2018) (cit. on pp. 54, 96).
- [192] Martin Fränzle, Holger Hermanns, and Tino Teige. “Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems”. In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2008, pp. 172–186 (cit. on p. 54).
- [193] André Platzer. “Differential dynamic logic for hybrid systems”. In: *Journal of Automated Reasoning* 41.2 (2008), pp. 143–189 (cit. on p. 54).
- [194] André Platzer. *Logical foundations of cyber-physical systems*. Springer, 2018 (cit. on p. 54).

-
- [195] Anita Devi. "Spectrum Based Fault Localization". In: *International Journal of Computer Trends and Technology (IJCTT)*—volume 4 (), p. 1698 (cit. on p. 55).
- [196] Rui Abreu, Peter Zoetewij, and Arjan JC Van Gemund. "Spectrum-based multiple fault localization". In: *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society. 2009, pp. 88–99 (cit. on p. 55).
- [197] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry. "Towards verified artificial intelligence". In: *arXiv preprint arXiv:1606.08514* (2016) (cit. on p. 55).
- [198] Ingo Pill and Franz Wotawa. "Spectrum-based fault localization for logic-based reasoning". In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2018, pp. 192–199 (cit. on p. 55).
- [199] Agnar Aamodt and Enric Plaza. "Case-based reasoning: Foundational issues, methodological variations, and system approaches". In: *AI communications* 7.1 (1994), pp. 39–59 (cit. on p. 55).
- [200] Eyke Hullermeier. "Credible case-based inference using similarity profiles". In: *IEEE Transactions on Knowledge and Data Engineering* 19.6 (2007), pp. 847–858 (cit. on p. 55).
- [201] Marta Fullen, Peter Schüller, and Oliver Niggemann. "Defining and validating similarity measures for industrial alarm flood analysis". In: *Industrial Informatics (INDIN), 2017 IEEE 15th International Conference on*. IEEE. 2017, pp. 781–786 (cit. on p. 55).
- [202] P Struss. "Model-based On-Board Diagnosis and Tools for the Developer of On-Board Systems-VMBD and IDD: Two Projects of the European Car Industries". In: *MONET Newsletter Issue 2 - Automotive Supplement* (2002) (cit. on p. 55).
- [203] Gautam Biswas et al. "An application of data driven anomaly identification to spacecraft telemetry data". In: *Prognostics and Health Management Conference*. 2016 (cit. on pp. 55, 57).
- [204] Fernando Garramiola et al. "A review in fault diagnosis and health assessment for railway traction drives". In: *Applied Sciences* 8.12 (2018), p. 2475 (cit. on p. 55).
- [205] Fernando Garramiola et al. "A hybrid sensor fault diagnosis for maintenance in railway traction drives". In: *Sensors* 20.4 (2020), p. 962 (cit. on p. 55).
- [206] Tat Nghia Nguyen, Thomas Downar, and Richard Vilim. "A probabilistic model-based diagnostic framework for nuclear engineering systems". In: *Annals of Nuclear Energy* 149 (2020), p. 107767 (cit. on p. 55).
- [207] Daniel Jung. "Structural Methods for Distributed Fault Diagnosis of Large-Scale Systems". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 2690–2695 (cit. on p. 55).
- [208] Erik Frisk and Mattias Krysander. "Residual selection for consistency based diagnosis using machine learning models". In: *IFAC-PapersOnLine* 51.24 (2018), pp. 139–146 (cit. on p. 55).
- [209] Youssef Mahmoud Youssef and Daniel Ota. "A general approach to health monitoring & fault diagnosis of unmanned ground vehicles". In: *2018 International Conference on Military Communications and Information Systems (ICMCIS)*. IEEE. 2018, pp. 1–8 (cit. on p. 55).
- [210] Youssef Youssef and Paul Plöger. "A Non-intrusive Fault Diagnosis System for Robotic Platforms." In: *DX@ Safeprocess*. 2018 (cit. on p. 55).

- [211] Qiujie Wang, Tao Jin, and Mohamed A Mohamed. "An innovative minimum hitting set algorithm for model-based fault diagnosis in power distribution network". In: *IEEE Access* 7 (2019), pp. 30683–30692 (cit. on p. 55).
- [212] Pin Lyu et al. "A novel RSG-based intelligent bearing fault diagnosis method for motors in high-noise industrial environment". In: *Advanced Engineering Informatics* 52 (2022), p. 101564 (cit. on p. 55).
- [213] Mehdi Mousavi et al. "A new fault diagnosis approach for heavy-duty gas turbines". In: *IEEE/ASME Transactions on Mechatronics* (2022) (cit. on p. 55).
- [214] Louise Travé-Massuyès. "Bridging control and artificial intelligence theories for diagnosis: A survey". In: *Engineering Applications of Artificial Intelligence* 27 (2014), pp. 1–16 (cit. on p. 56).
- [215] Daniel Jung et al. "Combining model-based diagnosis and data-driven anomaly classifiers for fault isolation". In: *Control Engineering Practice* 80 (2018), pp. 146–156 (cit. on p. 56).
- [216] Daniel Jung et al. "A combined diagnosis system design using model-based and data-driven methods". In: *Control and Fault-Tolerant Systems (SysTol), 2016 3rd Conference on*. IEEE. 2016, pp. 177–182 (cit. on p. 56).
- [217] Andreas Lundgren and Daniel Jung. "Data-driven fault diagnosis analysis and open-set classification of time-series data". In: *Control Engineering Practice* 121 (2022), p. 105006 (cit. on p. 56).
- [218] Daniel Jung. "Automated Design of Grey-Box Recurrent Neural Networks For Fault Diagnosis using Structural Models and Causal Information". In: *Learning for Dynamics and Control Conference*. PMLR. 2022, pp. 8–20 (cit. on p. 56).
- [219] Judea Pearl. "Causal Diagrams for Empirical Research". In: *California Digital Library* (1995), p. 36 (cit. on pp. 57, 131).
- [220] Jens Eickmeyer et al. "Data Driven Modeling for System-Level Condition Monitoring on Wind Power Plants." In: *DX*. 2015, pp. 43–50 (cit. on p. 67).
- [221] Firas Bayram, Bestoun S Ahmed, and Andreas Kassler. "From concept drift to model degradation: An overview on performance-aware drift detectors". In: *Knowledge-Based Systems* (2022), p. 108632 (cit. on p. 67).
- [222] Leonardo de Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340 (cit. on pp. 70, 121).
- [223] Silke Merkelbach et al. "Using vibration data to classify conditions in disk stack separators". In: *Vibroengineering PROCEDIA* 46 (2022), pp. 21–26 (cit. on p. 82).
- [224] Kalyan Perumalla, Srikanth Yoginath, and Juan Lopez. "Detecting Sensors and Inferring their Relations at Level-0 in Industrial Cyber-Physical Systems". In: *2019 IEEE International Symposium on Technologies for Homeland Security (HST)*. Nov. 2019, pp. 1–5. DOI: 10.1109/HST47167.2019.9032891 (cit. on p. 83).
- [225] Hirotogu Akaike. "Information theory and an extension of the maximum likelihood principle". In: *Selected papers of hirotugu akaike*. Springer, 1998, pp. 199–213 (cit. on p. 85).
- [226] Richard P Feynman, Robert B Leighton, and Matthew Sands. "The feynman lectures on physics; vol. i". In: *American Journal of Physics* 33.9 (1965), pp. 750–752 (cit. on p. 90).

- [227] Henrik Niemann and Niels Kjølstad Poulsen. "Fault tolerant control—a residual based set-up". In: *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 8470–8475 (cit. on p. 93).
- [228] Ron J Patton and Jie Chen. "Observer-based fault detection and isolation: Robustness and applications". In: *Control Engineering Practice* 5.5 (1997), pp. 671–682 (cit. on p. 95).
- [229] Karl R Popper. "Science as falsification". In: *Conjectures and refutations* 1.1963 (1963), pp. 33–39 (cit. on p. 103).
- [230] James J Downs and Ernest F Vogel. "A plant-wide industrial process control problem". In: *Computers & chemical engineering* 17.3 (1993), pp. 245–255 (cit. on pp. 108, 110).
- [231] Gianluca Manca. "Tennessee-Eastman-Process" Alarm Management Dataset. 2020. DOI: 10.21227/326k-qr90. URL: <https://dx.doi.org/10.21227/326k-qr90> (cit. on pp. 108, 112).
- [232] Mark C Hansen, Hakan Yalcin, and John P Hayes. "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering". In: *IEEE Design & Test of Computers* 16.3 (1999), pp. 72–80 (cit. on p. 109).
- [233] Meera Sampath et al. "Diagnosability of discrete-event systems". In: *IEEE Transactions on automatic control* 40.9 (1995), pp. 1555–1575 (cit. on p. 115).
- [234] Shengbing Jiang et al. "A polynomial algorithm for testing diagnosability of discrete-event systems". In: *IEEE Transactions on Automatic Control* 46.8 (2001), pp. 1318–1321 (cit. on p. 115).
- [235] Charles Pecheur, Alessandro Cimatti, and Ro Cimatti. "Formal verification of diagnosability via symbolic model checking". In: *Workshop on Model Checking and Artificial Intelligence (MoChArt-2002), Lyon, France*. 2002 (cit. on p. 115).
- [236] Benjamin Bittner et al. "Diagnosability of fair transition systems". In: *Artificial Intelligence* 309 (2022), p. 103725 (cit. on p. 115).
- [237] Larry Stockmeyer. "Classifying the computational complexity of problems". In: *The journal of symbolic logic* 52.1 (1987), pp. 1–43 (cit. on p. 121).
- [238] Faisal N Abu-Khzam. "A kernelization algorithm for d-hitting set". In: *Journal of Computer and System Sciences* 76.7 (2010), pp. 524–531 (cit. on p. 121).
- [239] Mohamed El Halaby. "On the Computational Complexity of MaxSAT." In: *Electron. Colloquium Comput. Complex.* Vol. 23. 2016, p. 34 (cit. on pp. 121, 122).
- [240] Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. "On the parameterized complexity of polytree learning". In: *arXiv preprint arXiv:2105.09675* (2021) (cit. on p. 121).
- [241] Russell Impagliazzo, Pavel Pudlák, and Jiri Sgall. "Lower bounds for the polynomial calculus and the Gröbner basis algorithm". In: *Computational Complexity* 8.2 (1999), pp. 127–144 (cit. on p. 122).
- [242] Lydia Kaiser et al. "Automatic verification of modeling rules in systems engineering for mechatronic systems". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 55867. American Society of Mechanical Engineers. 2013, Vo2BT02A009 (cit. on p. 127).

- [243] Esteban Arroyo et al. "Automatic derivation of qualitative plant simulation models from legacy piping and instrumentation diagrams". In: *Computers & Chemical Engineering* 92 (2016), pp. 112–132 (cit. on p. 131).
- [244] Lameya Afroze et al. "Domain Knowledge Injection Guidance for Predictive Maintenance". In: *Machine Learning for Cyber-physical Systems* (2023) (cit. on p. 131).
- [245] Kaja Balzereit and Oliver Niggemann. "AutoConf: A New Algorithm for Reconfiguration of Cyber-Physical Production Systems". In: *IEEE Transactions on Industrial Informatics* (2022) (cit. on p. 131).
- [246] Hannes Leipold and Federico M Spedalieri. "Quantum Annealing with Special Drivers for Circuit Fault Diagnostics". In: *arXiv preprint arXiv:2203.09560* (2022) (cit. on p. 132).
- [247] Hannes Leipold, Federico M Spedalieri, and Eleanor Rieffel. "Tailored Quantum Alternating Operator Ansatzes for Circuit Fault Diagnostics". In: *Algorithms* 15.10 (2022), p. 356 (cit. on p. 132).
- [248] Zhengbing Bian et al. "Mapping constrained optimization problems to quantum annealing with application to fault diagnosis". In: *Frontiers in ICT* (2016), p. 14 (cit. on p. 132).
- [249] Alexander Feldman, Johan de Kleer, and Ion Matei. "A Quantum Algorithm for Computing All Diagnoses of a Switching Circuit". In: *arXiv preprint arXiv:2209.05470* (2022) (cit. on p. 132).