



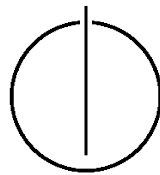
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

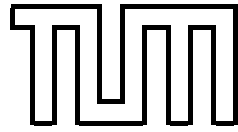
Dissertation in Informatik

**An Actual Causality Framework for  
Accountable Systems**

Amjad Ibrahim







FAKULTÄT FÜR INFORMATIK  
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Lehrstuhl IV - Software and Systems Engineering

# **An Actual Causality Framework for Accountable Systems**

*Amjad Ibrahim*

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Helmut Seidl

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Alexander Pretschner
2. Univ.-Prof. Dr. Joseph Halpern,  
Cornell University, USA

Die Dissertation wurde am 03.11.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 04.03.2021 angenommen.



---

## Acknowledgments

One day, we'll be what we want.  
The journey hadn't yet begun,  
the road hadn't ended.  
The wise men hadn't yet reached  
their exile.  
The exiled men hadn't yet attained  
their wisdom.

سنكون يوماً ما نريدُ  
لا الرحلةُ ابتدأتُ ، ولا الدربُ أنتهى  
لم يبلغ الحكماءُ غربتهم  
كما لم يبلغ الغرباءُ حكمتهم

Mural. Unfortunately, It Was Paradise: Selected Poems,  
by Mahmoud Darwish et al., University of California Press.

جدارية لمحمود درويش (٢٠٠٠)

Throughout my life, I always come back to this verse as I close chapters, “*did I attain my wisdom? Was it the journey*” I still do not know. However, I know that this particular adventure was unique; it was very challenging, still rewarding in many ways. It made me a new person in many aspects, close to what I want to be; more humble, precise, critical, resilient and persistent.

I owe my resilience and persistence to my late mother; to her, goes my utmost gratitude. Mom: you once joked, “a doctorate title goes well with your name, Amjad!” I like to think that this step would make you proud and happy; I did it for you, mom; and my father, who took part in shaping this thesis through our long conversations on the topic and around it.

My deepest gratitude goes to my supervisor, Prof. Dr. Alexander Pretschner. Alex, it is not a coincidence that you have been my teacher for very long. From the very first lecture I attended with you in 2013 until now, I am learning from you. Thank you for preparing me for this, granting me the chance, challenging me, guiding me, and having my back every time things did not work as expected. Without all our discussions, I certainly would not have been writing this part of this thesis.

Prof. Joseph Halpern, this whole thesis is an acknowledgment of the work you have done. I would also like to express my gratefulness for your insightful comments and support over the past three years. Thank you very much for your warm hospitality in Ithaca.

Along this journey, I met a diverse and friendly group at the Chair of Software and Systems Engineering. To all my friends at the chair, thank you for sharing pain, tiredness, laughs, food, ice cream, beer, and wine with me. I am grateful to the Bachelor and Master's students who helped in building parts of this thesis, especially Simon, Tobias, Stavros, and others. I am also grateful to Severin and Claudius for proofreading parts of this thesis.

Finally, a heartfelt thanks to my dear family in “the lady of earth, the mother of all beginnings, the mother of all endings,” *Palestine*; my father, sisters, and little nephews, you were always a part of this adventure. To my bigger family of friends in Palestine and Germany and especially those in the intersection of the two circles, thanks for your support.



---

## Zusammenfassung

Mit dem stetigen Einzug von digitalen Systemen in alle Facetten des täglichen Lebens wächst die Notwendigkeit deren Handeln erklärbar zu machen. Da moderne Systeme Personen schaden, Assets kompromittieren oder über Kreditwürdigkeit entscheiden können, müssen sie accountable sein. Accountability eines Systems zu ermöglichen im Sinne der Entwicklung von forensischen Fähigkeiten, um Unfälle zu erklären und um mögliche fehlerverhaltende Parteien verantwortlich zu machen, ist einer der Säulen dieser Doktorarbeit. Die drei Konzepte — Accountability, Erklärbarkeit, Verantwortlichkeit — basieren auf Kausalität; sie benötigen ein passendes Verständnis von Kausalität. Actual Causality, ein Konstrukt das seine Wurzeln in der Philosophie der Wissenschaft hat, ist formalisiert in Halpern und Pearls Kausalitätstheorie. Allerdings wurde — soweit uns bekannt — die Theorie und Operationalisierung der Actual Causality nicht explizit in automatisierter Form für soziotechnische Zwecke wie Accountability verwendet. Stattdessen wurden sie nur adaptiert und verwendet im Zusammenhang mit domänenspezifischen Konzepten wie der Lineage einer Databankanfrage. Dies ist eine Einschränkung, die teilweise auf die Berechnungskomplexität von Kausalität zurückzuführen ist. Obwohl Actual Causality in verschiedenen Disziplinen angewandt wird, fehlt trotzdem eine Methodik, um eine effiziente automatisierte Unterstützung zu entwickeln, welche Accountability moderner Systeme ermöglicht trotz ihrer Komplexität.

In dieser Dissertation entwickeln wir ein umfassendes Framework, automatisiert durch eine Toolchain, um das Problem der Operationalisierung von Tatsächlicher Kausalität für Zwecke der Accountability zu adressieren. Das Framework fokussiert auf die Effizienz und Skalierbarkeit verschiedener Varianten automatisierter Kausalitätsschlussfolgerungen, sowie die Effektivität der Schlussfolgerungen durch praktische und domänenspezifische Ansätze für kausale Modellierung und Kontextualisierung. Da Wiederverwendung gefördert wird, werden Schwierigkeiten bei der Einbettung von Kausalitätsschlussfolgerungen in neue Domänen reduziert. Auf der einen Seite wird eine Reihe an generellen Algorithmen beigetragen um das Schlussfolgern zu automatisieren, was die Wiederverwendung über mehrere Domänen ermöglicht, auf der anderen Seite wird die Wiederverwendung von domänenspezifischen Methodiken und Wissensquellen unterstützt um Modellierung und Kontextualisierung zu operationalisieren. Das führt dazu, dass der Fokus stärker auf die Integration dieser Lösungen zur Ermöglichung von Accountability in spezifischen Systemen wechselt, welche auch durch unser Framework adressiert werden, da es zur Lösung von Accountability-bezogene Probleme für eine diverse Menge von Systemen genutzt werden kann.

Wir demonstrieren, wie man mit dem Framework als Ganzem interagiert, indem wir uns auf Fallbeispiele aus dem Bereich der Microservice-basierten Systeme, sowie auf Beispiele von Flugzeugunglücken, Versagen von cyber-physikalischen Systemen und Systemen der künstlichen Intelligenz konzentrieren, da wir überzeugt sind, dass diese Systeme repräsentativ für moderne, komplexe Systeme sind. Weiterhin zeigen wir, dass das Framework

---

generisch genug ist um Accountability und verwandte Konzepte wie Responsibility einzubinden und dass es mit Toolunterstützung in verschiedene Domänen eingebunden werden kann. Des Weiteren präsentieren wir verschiedene Experimente um darzulegen, dass unsere Ansätze für Kausalitätsschlussfolgerungen effizienter und allgemeingültiger sind als andere aktuelle Techniken.



---

## Abstract

With the rapid deployment of digital systems into all aspects of daily life, the need to understand their actions grows. As modern systems might harm people, compromise assets, or decide loan adequacy, they ought to be accountable. Enabling *accountability* of a system in the sense of developing its forensic capabilities to *explain* mishaps and possibly hold misbehaving parties *responsible* for violations, is the pillar of this doctoral thesis. The three concepts—accountability, explanation, and responsibility—are inherently causal; they require a notion of causality to enable them. Actual causality, a construct rooted in the philosophy of science, is well formalized within the Halpern and Pearl theory of causality. However, to the best of our knowledge, explicit actual causality theories and operationalizations have not been utilized, in an automated fashion, to enable socio-technical purposes such as accountability. Instead, they are adapted and used only in correspondence with domain-specific concepts such as a lineage of a database query. A restriction that is partially attributed to the computational complexity of causality. While used across different disciplines, actual causality still lacks a methodology to devise efficient automated assistance that enables modern systems’ accountability, given their complexity.

In this thesis, we develop a unifying framework, automated by a set of tools, to address the problem of operationalizing actual causality for purposes related to accountability. The framework tackles the efficiency, and scalability of different notions of automated causal reasoning. Also, it considers the effectiveness of this reasoning through practical domain-specific approaches to causal modeling and contextualization. Thus, the framework diminishes the barrier to embedding causality reasoning in new domains because it promotes *reuse*. On the one hand, it contributes a set of general algorithms to automate reasoning so that it can be reused among different domains. It supports, on the other hand, the reuse of domain-specific methodologies and knowledge sources to operationalize modeling and contextualization. As a result, the focus then shifts to how all these solutions are integrated to serve the goal of enabling accountability in a specific system. To that end, our framework can be used to solve accountability-based problems for a wide range of systems.

With case studies from such areas as aircraft accidents, Cyber-physical systems’ faults, microservice-based systems security attacks, and artificial intelligence systems, which we deem representative of modern complex systems, we demonstrate the utilization of the framework as a whole. We show that it is generic enough to accommodate accountability and related concepts, such as responsibility, and with tool support, it is amenable to be incorporated into different domains. We present multiple experiments to show that our approaches to causal reasoning are more efficient and general than the state-of-the-art techniques.



---

# Outline of the Thesis

## CHAPTER 1: INTRODUCTION

This chapter introduces the topic and the context of this thesis. It discusses problems, gaps, goals, and contributions of this work.

## CHAPTER 2: BACKGROUND

This chapter presents the formal foundations required to understand the concept of actual causality.

## CHAPTER 3: EFFICIENTLY CHECKING ACTUAL CAUSALITY WITH SAT SOLVING

This chapter presents a novel encoding of binary causality checking queries into SAT. Several other encodings are derived and evaluated as part of the chapter. Parts of this chapter have previously appeared in the publication [98], co-authored by the author of this thesis.

## CHAPTER 4: ACTUAL CAUSALITY COMPUTATIONS AS OPTIMIZATION PROBLEMS

This chapter presents a formulation of different notions of actual causality computations over binary models as optimization problems. Parts of this chapter have previously appeared in the publication [97], co-authored by the author of this thesis.

## CHAPTER 5: ACTUAL CAUSALITY CHECKING BEYOND BINARY MODELS

This chapter presents a generalization of the concepts in the previous chapters, and proposes a method to answer checking causal queries of numeric models.

## CHAPTER 6: CAUSAL MODEL EXTRACTION FROM ATTACK TREES TO ATTRIBUTE INSIDER ATTACKS

This chapter tackles issues around causal modeling in the context of malicious insiders. Parts of this chapter have previously appeared in publications [99, 95], co-authored by the author of this thesis.

## CHAPTER 7: AUTOMATED GENERATION OF ATTACK GRAPHS AND CAUSAL MODELS FOR MICROSERVICES

This chapter presents an automated approach to generate models of attacks in microservice-based systems. Parts of this chapter have previously appeared in the publication [94], co-authored by the author of this thesis.

## CHAPTER 8: MODEL-DRIVEN CONTEXTUALIZATION FOR MICROSERVICES

This chapter presents an automated approach to use models of attacks to monitor relevant events in microservice-based systems.

---

## CHAPTER 9: A FRAMEWORK FOR OPERATIONALIZING ACTUAL CAUSALITY

This chapter presents a unifying framework for operationalizing actual causality. Parts of this chapter have previously appeared in publication [96], co-authored by the author of this thesis.

## CHAPTER 10: RELATED WORK

This chapter reviews the related work in the fields of accountability, actual causality reasoning, and other related fields. Parts of this chapter have been published in the following publications [167, 98, 97, 96, 94, 95, 99], co-authored by the author of this thesis.

## CHAPTER 11: CONCLUSIONS

This chapter concludes the work in the thesis. It summarizes the contributions proposed throughout the chapters. We state the results of the thesis and the lessons learned during the development of this work. Afterward, we discuss limitations and avenues for future work.

## APPENDIX A: EVALUATED MODELS

This appendix provides a detailed overview of our evaluated dataset. Parts of this appendix have previously appeared in publications [98, 96], co-authored by the author of this thesis.

*N.B.: Multiple chapters of this dissertation are based on different publications authored or co-authored by the author of this dissertation. Such publications are mentioned in the short descriptions above. Due to the obvious content overlapping, quotes from such publications within the respective chapters are not marked explicitly.*

# Contents

<b>Acknowledgements</b>	<b>v</b>
<b>Zusammenfassung</b>	<b>vii</b>
<b>Abstract</b>	<b>ix</b>
<b>Outline of the Thesis</b>	<b>xi</b>
<b>Contents</b>	<b>xiii</b>
<b>I Introduction and Background</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Gaps, Problem Statement, and Research Questions . . . . .	8
1.1.1 Actual Causality Reasoning . . . . .	9
1.1.2 Causal Modeling and Contextualization . . . . .	10
1.2 Goal and Benefits . . . . .	11
1.3 Solution . . . . .	12
1.4 Contributions . . . . .	12
1.5 Summary of Results . . . . .	15
1.6 Structure . . . . .	15
<b>2 Background</b>	<b>17</b>
2.1 Overview . . . . .	17
2.2 Causal Models . . . . .	18
2.3 Reasoning about Causality . . . . .	20
2.4 Responsibility as an Extension . . . . .	23
2.5 Discussion . . . . .	23
2.6 Summary . . . . .	25
<b>II Computational Aspects of Actual Causality Reasoning</b>	<b>27</b>
<b>3 Efficiently Checking Actual Causality with SAT Solving</b>	<b>29</b>
3.1 Introduction . . . . .	29

3.2	Brute-Force Based Causality Checking . . . . .	30
3.3	SAT Based Causality Checking . . . . .	31
3.3.1	Checking AC2 . . . . .	32
3.3.2	Checking AC3 . . . . .	35
3.3.3	Example . . . . .	37
3.3.4	Optimized AC3 Check with SAT . . . . .	38
3.4	Graph-Based Optimizations . . . . .	40
3.4.1	Reduce Number of Potential Variables for $\vec{W}$ . . . . .	40
3.4.2	Remove Irrelevant Sub-formulae . . . . .	41
3.5	Evaluation . . . . .	42
3.5.1	Technical Implementation . . . . .	42
3.5.2	Methodology and Evaluated Models . . . . .	43
3.5.3	Discussion and Results . . . . .	43
3.6	Proofs . . . . .	47
3.6.1	Negation Lemma . . . . .	47
3.6.2	AC2 Encoding Proof . . . . .	49
3.6.3	AC3 Encoding Proof . . . . .	50
3.6.4	Optimized AC3 Encoding Proof . . . . .	52
3.7	Summary . . . . .	53
<b>4</b>	<b>Actual Causality Computations as Optimization Problems</b>	<b>55</b>
4.1	Introduction . . . . .	55
4.2	Checking and Semi-inference Queries as Optimization Problems . . . . .	56
4.2.1	The Objective in Causality Checking . . . . .	56
4.2.2	ILP Formulation . . . . .	57
4.2.3	MaxSAT Encoding . . . . .	58
4.2.4	Results Interpretation . . . . .	59
4.2.5	Example . . . . .	59
4.3	Causality Inference with ILP . . . . .	60
4.3.1	<i>Why<sub>ILP</sub></i> Algorithm . . . . .	62
4.3.2	Example . . . . .	64
4.4	Evaluation . . . . .	65
4.4.1	Evaluating Checking and Semi-Inference . . . . .	65
4.4.2	Evaluating Inference . . . . .	68
4.5	Proofs . . . . .	70
4.6	Summary . . . . .	72
<b>5</b>	<b>Actual Causality Checking Beyond Binary Models</b>	<b>75</b>
5.1	Introduction . . . . .	75
5.2	Approach . . . . .	76
5.2.1	Requirements For Causality Checking . . . . .	76
5.2.2	Building blocks . . . . .	77

---

5.2.3	Algorithm . . . . .	79
5.2.4	Example . . . . .	82
5.3	Evaluation . . . . .	86
5.3.1	Model Size . . . . .	86
5.3.2	Performance . . . . .	87
5.4	Summary . . . . .	89
 <b>III Domain-specific Causal Modeling and Contextualization</b>		<b>91</b>
<b>6</b>	<b>Causal Model Extraction from Attack Trees to Attribute Insider Attacks</b>	<b>93</b>
6.1	Introduction . . . . .	93
6.2	Preliminaries . . . . .	95
6.2.1	Foundations of Attack Trees . . . . .	95
6.2.2	Malicious Insiders Example . . . . .	96
6.3	Attack Trees to Causal Models . . . . .	97
6.3.1	Suspect Attribution . . . . .	98
6.3.2	Attributed Attack Tree Transformation . . . . .	101
6.3.3	Adding Preemption Relations . . . . .	102
6.3.4	Tool Support . . . . .	103
6.4	Evaluation . . . . .	104
6.4.1	The Efficiency of the Extraction . . . . .	105
6.4.2	The Validity of the Approach . . . . .	106
6.4.3	The Effectiveness of the Model . . . . .	107
6.5	Summary . . . . .	108
<b>7</b>	<b>Automated Generation of Attack Graphs and Causal Models for Microservices</b>	<b>109</b>
7.1	Introduction . . . . .	109
7.2	Preliminaries . . . . .	111
7.3	Approach . . . . .	112
7.3.1	Attack Graph Generation for Dockers . . . . .	112
7.3.2	Extracting Causal Models from Attack Graphs . . . . .	117
7.4	Evaluation . . . . .	118
7.4.1	Experiment Setup . . . . .	118
7.4.2	Scalability evaluation . . . . .	119
7.4.3	Effectiveness of the Graphs . . . . .	121
7.5	Summary . . . . .	123
<b>8</b>	<b>Model-driven Contextualization for Microservices</b>	<b>125</b>
8.1	Introduction . . . . .	125
8.2	The Approach . . . . .	127
8.2.1	Monitoring Configuration . . . . .	127

8.2.2	Monitoring Orchestrator . . . . .	128
8.2.3	The overall Architecture . . . . .	128
8.3	Evaluation . . . . .	130
8.3.1	Effectiveness . . . . .	130
8.3.2	Efficiency . . . . .	132
8.4	Summary . . . . .	133
<b>IV A Framework for Accountable Systems</b>		<b>135</b>
<b>9 A Framework for Operationalizing Actual Causality</b>		<b>137</b>
9.1	Introduction . . . . .	137
9.2	A Framework of Actual Causality . . . . .	139
9.2.1	Causal Modeling . . . . .	139
9.2.2	Contextualization . . . . .	141
9.2.3	Causal Reasoning . . . . .	142
9.2.4	The Technical Framework of Actual Causality . . . . .	143
9.3	The Actual Causality Canvas . . . . .	144
9.4	Use Cases . . . . .	146
9.4.1	Explainable AI . . . . .	146
9.4.2	Überlingen mid-air Collision <sup>1</sup> . . . . .	149
9.4.3	Malicious Insiders <sup>2</sup> . . . . .	154
9.4.4	Drone Crash Diagnosis . . . . .	156
9.5	Evaluation . . . . .	158
9.6	Summary . . . . .	159
<b>V Related Work and Conclusions</b>		<b>161</b>
<b>10 Related Work</b>		<b>163</b>
10.1	Accountability . . . . .	163
10.2	Causality . . . . .	164
10.2.1	Actual Causality Reasoning According to HP . . . . .	164
10.2.2	Non-HP Causality Reasoning . . . . .	167
10.3	Causal Models for Accountability . . . . .	169
10.3.1	Insider Threat and Threat Models . . . . .	169
10.3.2	Safety, Fault Trees, and WBA . . . . .	171
10.4	Model-driven Contextualization . . . . .	171
10.5	Summary of the Gaps . . . . .	172

---

<sup>1</sup>A shortened version of this case-study can be found in Appendix A

<sup>2</sup>This is a shortened version of the example in Chapter 6. The reiteration here is meant to put accountability of microservices-based system in perspective with other domains.



<b>11 Conclusions</b>	<b>173</b>
11.1 Thesis Overview . . . . .	173
11.2 Main Results . . . . .	174
11.3 Limitations . . . . .	176
11.4 Future Work . . . . .	176
<b>Bibliography</b>	<b>179</b>
<b>Index</b>	<b>195</b>
<b>List of Figures</b>	<b>195</b>
<b>List of Tables</b>	<b>197</b>
<b>A Evaluated Models</b>	<b>199</b>
A.1 Introduction . . . . .	199
A.2 Description of the Evaluated Models . . . . .	200
A.2.1 Rock-Throwing . . . . .	200
A.2.2 Forest Fire . . . . .	200
A.2.3 Prisoners . . . . .	201
A.2.4 Assassin . . . . .	201
A.2.5 Railroad . . . . .	202
A.2.6 Abstract Model 1 & 2 . . . . .	203
A.2.7 Leakage in Subsea Production System . . . . .	204
A.2.8 Überlingen Accident Investigation . . . . .	206
A.2.9 Binary Tree . . . . .	208
A.2.10 Abstract Model 1 Combined with Binary Tree . . . . .	208
A.2.11 Steal Master Key . . . . .	208



## **Part I**

# **Introduction and Background**



# 1 Introduction

*This chapter introduces the topic and the context of this thesis. It discusses problems, gaps, goals, and contributions of this work.*

Modern digital systems increasingly influence people's lives. The benefits brought by information and cyber-physical systems made them almost indispensable, not only to the industry but to society as well. However, the behavior of such systems at run-time does not always align with the expectations of all the stakeholders. This broad insight manifests in a wide spectrum of, almost inevitable, *unwanted events* ranging from minor software bugs to fatal accidents of aircraft, or cyber-attacks on critical infrastructures. Generally, we consider an unwanted event to be any violation of legal, self-imposed, or contractually agreed-on obligations of a system behavior [13]. As we argue in this introduction, unwanted events for modern digital systems *cannot* be entirely excluded by design. Because these systems are becoming an integral part of our society, we consider it then mandatory to provide mechanisms that help us to understand what caused these events, both for eliminating the underlying (technical) problem and for assigning blame.

With the ease of composing technical components, systems nowadays tend not to maintain fixed boundaries anymore. Leveraging application programming interfaces (APIs), a *component* (or a *system*) can, in principle, directly interact with other components. The virtual composition of systems through interfaces plays an instrumental role in the current technological world. For example, modern cars are an assembly of up to *a hundred or more* electronic control units that run and communicate over a controller area network bus [45]. For cyber-physical systems such as drones, manufacturers provide developers with APIs to build custom applications that run on their drones. In cloud-based information systems, recent paradigms of complete infrastructure automation are increasingly dominating the field. As such, emerging cloud systems can automatically onboard or offboard services on demand and create or remove service replicas based on load. In all such examples, the interfaces between components that are possibly designed and developed by different entities are software contracts that govern the interaction. We want to remind that *any* interface description provides a behavior abstraction. Regardless of the level of abstraction, there must be parts of the behavior that are left unspecified when composing a system. The composition of systems through abstract contracts, in turn, means that the boundary of such systems usually *cannot* be specified without leaving open several degrees of freedom in terms of how to "legally" use the system. This renders the goal of preventing all the illegal states of a system, at design time, impractical.

The previous remark emphasizes ascribing unwanted behavior of a system to technical faults —possibly due to interface misuse. Other problems can be attributed to human interaction with systems. Similar to technical faults, the total elimination of human-related problems is also infeasible most of the time. Without further complication, the infeasibility can be a simple consequence of the human role in the system. For example, let us consider the case of *insiders* in security-critical systems (e.g., online-banking platform).<sup>1</sup> Preventive measures (e.g., access control), in the context of insiders, have a high likelihood of failing because insiders ought to possess the necessary privileges to perform their jobs; however, they can abuse them. The necessity of these privileges for the daily job of the insider hampers preventing such actions from happening. Especially with security in mind, trade-offs between qualities of the system (e.g., security vs. functionality) often arise during the design phase. Consequently, unwanted events at run-time may occur as a side effect of such inevitable trade-offs.

Arguably, the mere sense of dissatisfaction or distrust in a system is considered an unwanted event. The concerned stakeholders may question the output of a system, especially if it is tasked with making critical decisions or predictions for humans using complex approaches. Systems with artificial intelligent agents (e.g., machine learning applications) fall within such category of systems. The regard of whether a customer object to an automated decision is indeed a violation of obligations is debatable; however, for the scope of this thesis, we consider it at least an instance of a suspicious event that requires clarification. Explaining learning systems' results does not only contribute to raising the trust in such systems but also a mandatory requirement for law and standard compliance [141, 143].<sup>2 3</sup>

With the above, we briefly discussed different factors that contribute to the inevitable occurrence of unwanted events. In certain situations, our society deals with a similar result by promoting *accountability*.

### An Overview of Accountability

Accountability is an interdisciplinary concept that is being studied in philosophy, law, social science, political science, and computer science [53, 200, 105, 13, 153]. Although it is related to our work, however, the mission of defining and comparing different notions of accountability is beyond the focus of this thesis. Nevertheless, as part of our introduction, we present a rather technical definition of accountability that facilitates enabling systems to be accountable. We start by considering the definition given by Beckers et al. of accountability as “a capability of a socio-technical system to help answer questions regarding the

---

<sup>1</sup>According to the Cyber Security Intelligence Index (2016), malicious insiders carried out 60% of all cyberattacks in 2015 [169].

<sup>2</sup>For instance, the European Union General Data Protection Regulation (GDPR) explicitly mentions accountability as a principle relating to processing of personal data (Art. 5), and the right to obtain explanations of automated processes (Recital 71). See <https://gdpr-info.eu/>; last accessed July 2020.

<sup>3</sup>“Be accountable to people” is Google’s fourth AI Principle; See <https://ai.google/principles/>; last accessed July 2020.

---

cause of an occurred unwanted event” [13]. From a practical perspective, the notion of unwanted events differs across systems, domains, applications, and qualities. In the scope of this thesis, we relate unwanted events to a specific quality attribute of a system, such as security, safety, accuracy, or functionality compliance. A quality attribute is an interest that pertains to the system’s development or operation. The *ISO/IEC-9126* (Software engineering — Product quality) and *ISO/IEC 25010* (Systems and software Quality Requirements and Evaluation) standards present structured classifications of these interests to characteristics and sub-characteristics [55, 56]. For example, according to *ISO/IEC-9126*, *functionality* (or *functional suitability* in *ISO/IEC 25010*) is a characteristic that includes *functional completeness*, *functional correctness*, and *functional appropriateness* [55], and *security* is another characteristic with sub-characteristics such as *confidentiality and accountability* [55, 56].

Refining the definition to include a quality attribute narrows down the spectrum of unwanted events to the events specific to some qualities of a system. Further, as we shall see in this thesis, defining violations based on their relation to a quality attribute enables utilizing artifacts such as methodologies, models, or tools from the respective attribute’s domain. According to a specific quality attribute, the benefits obtained by enabling accountability become clear. For example, accountability for diagnosability mechanism allows developers to localize faults and fix them in a timely fashion [63]. An accountability mechanism for performance would aid in locating bottlenecks in system deployment.

To that end, we define accountability *as a property of a system that helps to identify the causes of (unwanted) events related to a quality attribute*. The process of enabling accountability entails *developing system’s (forensic) capabilities in identifying miss-behaving parties responsible for specific violations*.

Figure 1.1 depicts a conceptual model of accountability. We consider a *system* to be a composition of *components* with different natures. A non-exhaustive list of component types includes hardware devices, software artifacts, or trained models. The interaction among different components realizes the *functionality* of the system, which is one of its *quality attributes* (as defined in *ISO/IEC-9126*). Although *accountability* is a quality concern itself, we consider it as a property of a *system* that aids it in attributing violations of one or more of its quality attributes specified by the *requirements*. The typed arrow (*supports*) means that *accountability* enables the attribution of events related to a quality attribute, hence supporting the achievement of the attribute.

We call a system *accountable* that can help answer questions regarding the root cause of some (usually undesired) event. For that, accountable systems require a technical ability that we refer to as an *accountability mechanism*. Many scholars have stressed the role of causal reasoning in enabling accountability mechanisms [200, 53, 52, 75, 119, 141, 143]. We believe that an accountability mechanism requires at least two properties: 1) the system must provide *evidence*, usually in the form of logs, that helps to understand the reasons of the unwanted event 2) a mechanism to argue *causality*. Accordingly, we refine the accountability entity in Figure 1.2 with the ingredients needed to enable such a capability. An *accountability mechanism* is a technical sub-system that enables answering *questions* regarding a specific *event*. The *answer* is supposed to be the *cause* of that *event*. As we see in this thesis, a *cause* is

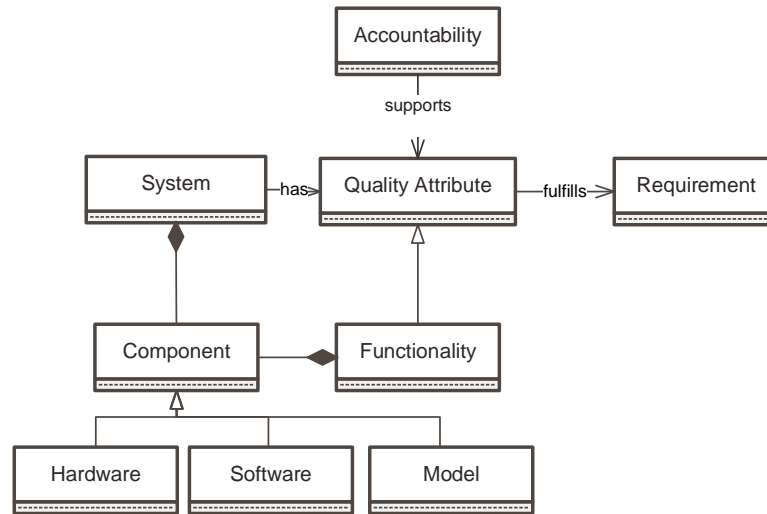


Figure 1.1: A Conceptual Model of Accountability

typically inferred in reference to a *context*, a description of the situation that contained both the *event* and the *cause*. In other words, an *accountability mechanism* acts as a *black-box* that is used to record and explain an incident after it happens. This thesis focuses on the causality reasoning part of accountability mechanisms.

## An Overview of Causality

Relating *causes* and *effects* to each other is a natural part of human cognition. We use causal reasoning to explain particular past events, to predict, possibly control, future events, or to attribute moral responsibility and legal liability [89]. Causality has produced centuries of interdisciplinary theorization. The first documented theories can be traced to ancient philosophers such as Plato (Phaedo, 360 BC) and Aristotle (Posterior Analytics), reaching to current theories by computer scientists such as Pearl [158] and Halpern [75]. Focusing on a goal-driven categorization of causality, we distinguish two notions: *type* (general) causality and *actual* (token) causality [79, 155]. Type causality is concerned with general causal relations and aims to forecast the *effect* of a cause [126]. Reasoning with this notion aids in expanding predictive fields such as medicine [108] and economy. In contrast, actual causality theories focus on explaining an observed event, that is, inferring *causes from effects* [75]. Thus, such theories are useful in assigning blame, explaining, or preventing similar events in the future [89]. According to its retrospective attribution, actual causality



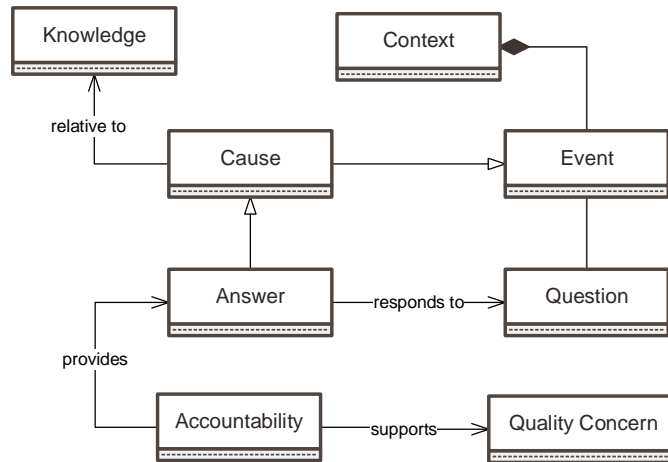


Figure 1.2: A Model of an Accountability Mechanism<sup>4</sup>

is central for enabling explanation-based socio-technical constructs such as accountability in cyber-physical systems [106, 95], in information systems [53, 52, 95], explainability in artificial intelligence [141], and responsibility attribution [33].

Formalizing a precise definition of a cause is challenging for both notions of causality. Attempts in that direction go back to the eighteenth century when Hume introduced the idea of counterfactual reasoning as a method for defining a cause [92]. Simply put, counterfactual reasoning concludes that event  $A$  is a cause of event  $B$  if  $B$  does not occur if  $A$  does not occur. However, this simple reasoning cannot be used with interdependent, multi-factorial, and complex causes [126]. Recently, in computer science, there have been some successful and influential efforts, by Joseph Halpern and Judea Pearl, at proposing a model-based definition of an *actual cause* while addressing the problematic issues in philosophy literature [75]. In essence, their definition—the Halpern-Pearl definition of actual causality (HP in the following)—provides a formal account of when we can call one or more events a cause of another event in a way that captures the human intuition. There have been three versions of HP: the original (2001), updated (2005), and modified (2015) versions, the latter of which we use in this thesis.

A fundamental contribution of HP is that it opens the door for embedding the ability to reason about actual causality into modern digital systems because of its formal foundation.

<sup>4</sup>Intuitively, a *cause* may exist even without specified *knowledge*; however, in this thesis, we always refer to the cause in relation to knowledge.

To a large extent, the definition aspires to formalize the human’s intuition about the causes of effects. Thus, we believe that HP is suitable to explain the causality in accountability scenarios that include, in addition to technical factors, human interaction. Further, HP deals with a few challenges that have faced naive counterfactual reasoning approaches such as cases of preemption, absence of events, bogus prevention, causation by omission and commission, irrelevance [122, 75]. We discuss the features of HP in more detail in Chapter 2.

HP includes *three* artifacts: a *model* [158, 75], a *context*, and a *reasoning mechanism* [79, 81, 74]. Briefly, *causal modeling* refers to abstracting and representing general knowledge (hence *knowledge* in Figure 1.2) of cause and effect relations. A *causal model* describes the different factors contributing to a phenomenon and the way they influence each other. The structural equation model is probably the most known structure to express causal knowledge [155]. *Context setting*, on the other hand, refers to the description of a particular event. It can be thought of as an instance of the general model, or an assignment of values for events as depicted in Figure 1.2. Lastly, the definitions contain logical descriptions (formal definitions and conditions) that can be used to build *reasoning* mechanisms.

### 1.1 Gaps, Problem Statement, and Research Questions

Aiming to enable accountability in modern systems, this thesis focuses on actual causality reasoning. As a technique of knowledge representation and reasoning, actual causality is well formalized as a result of the HP definition [74]. However, to the best of our knowledge, explicit actual causality theories such as HP and operationalizations are not utilized, in an automated fashion, to enable socio-technical purposes such as accountability and explanation. Actual causality reasoning, while used across different disciplines, currently lacks a systematic methodology and mainly tools to enable accountability in modern systems (*Gap 1*, analysis of the related work that identified this gap is presented in Section 10.1 and Section 10.2.1).

Applications of HP, so far, depend on relations to domain-specific technical artifacts such as a lineage of a database query [134], or a counter-example of a model checker [123], which hinders the extension of these applications to the domains we consider for accountability. We are interested in scenarios and systems with a human or social context. For similar contexts, Halpern demonstrated the definition using simple philosophical examples [79]. Although insightful, these scenarios do not stress the potential practical problems that would arise in the technical implementation of the definition, such as scalability, efficiency, knowledge sources, and logging granularity. Such challenges manifest when inspecting prior technical implementations of the definition, in which trade-offs, for instance, between completeness and efficiency, are observed.<sup>5</sup> To that end, this research, as a whole, is directed towards answering the following research question:

---

<sup>5</sup>We refer to the comprehensive support of all the concepts around HP without any restrictions (e.g., one-equation model, singleton causes) as completeness

*RQ1: How can actual causality theories be effectively and efficiently operationalized to enable accountability in modern systems? (answered in Chapter 9)*

In the following, we discuss the different aspects of causality and accountability operationalization. We elaborate on the problems, gaps, and research questions of each aspect.

### 1.1.1 Actual Causality Reasoning

Because of its formal foundation, HP enables automated causality reasoning. We distinguish two notions of reasoning: *checking* and *inference*. *Checking* refers to verifying if a candidate cause is an actual cause of an effect, i.e., answering the question “is  $\vec{X}$  a cause of  $\varphi$ ?” *Inference* involves finding a cause without any candidates, i.e., answering the question “why  $\varphi$ ?” Using HP, causality checking is, in general,  $D_1^P$ -complete and  $NP$ -complete for singleton (one-event) causes [74]; the difference is due to a minimality requirement in the definition (details in Chapter 2). Intuitively, *inference* is at least as hard. This complexity resulted in multiple restrictions made by researchers and practitioners on the causal model or the causal query.

The standard restriction among all the utilizations of HP is the usage of *binary* causal models, i.e., models that contain Boolean variables only. We discuss this issue again in Chapter 10. Then, within different domains, further restrictions are assumed on the model. For instance, in the domain of databases, models contain single equations only based on the lineage of the query in [134], or no explicit equations in [18, 177]. Similar simplifications are made for Boolean circuits in [34]. Also, in the context of software and hardware verification, causal models contain no equations [16]. Similarly, causal queries are restricted. For instance, while the cause is assumed to be a singleton (one event) in some domains [16, 177], the effect is restricted to a specific type like monotone queries. Understandably, the mentioned approaches impose these restrictions on the general theory to deal with the complexity and because they do not compromise their aim. On the other hand, the general approaches (without restrictions) reported results that do not scale to large models [89]. Large models of causal factors are likely to occur especially when generated automatically from other sources for purposes of accountability and explainability [96, 95, 141]. Further, models of real-world accidents are sufficiently large to require efficient approaches. For instance, a model of the 2002 mid-air collision in Germany consists of 95 factors [187] (discussed in detail in Section 9.4.2), and another model of the 2006 Amazonas collision consists of 137 factors [184]; such models are expected to grow in size with data-driven causal discovery approaches. To the best of our knowledge, there exist no comprehensive, efficient, and scalable algorithms for reasoning about actual causality, especially with large causal models (*Gap 2*, analysis of the related work that identified this gap is presented in Section 10.2). As part of operationalizing actual causality, we aspire to answer the following questions.

*RQ2: How can actual causality be checked efficiently and effectively in acyclic binary causal models without any further restrictions on the model or the query? (answered in Chapter 3 and Chapter 4)*

*RQ3: How can actual causality be inferred efficiently and effectively in acyclic binary causal models without any further restrictions on the model or the query? (answered in Chapter 4)*

*RQ4: To what extent can actual causality be checked or inferred without restricting causal model languages? (answered in Chapter 5)*

### 1.1.2 Causal Modeling and Contextualization

Causality, based on HP, is model relative. Therefore, the action of creating a causal model is a crucial step towards an effective causality inference. Halpern and Pearl themselves have shown several times the difficulties of coming up with a proper model and its considerable influence on the result of cause evaluation. Having a system-wide comprehensive causal reasoning ability, i.e., one that explains all the events in a system, is an expensive, possibly unrealistic, goal to achieve. In this thesis, we assume that for accountability goals, including only the factors (formalized in models) that lead to unwanted events, is sufficient. Thus, our approach to causal modeling requires creating models of anticipated problems (e.g., security attacks, or safety hazards). This may seem intuitive; however, this is a fundamental difference between our work and other approaches that require behavioral models (normal behavior of the world) of the system [66, 64, 123].

To the best of our knowledge, no previous work has proposed practical methods to create effective HP causal models for accountability purposes (*Gap 3*, analysis of the related work that identified this gap is presented in Section 10.3). Since such methods can only be designed domain specifically, we emphasize reusing the knowledge represented in domain-specific artifacts and transforming them into causal models that support accountability analysis. Thus, we study the interaction between causal models and other sources of knowledge. Instances of before-mentioned sources include security threat modeling techniques such as attack trees [179] and attack graphs [182], hazard modeling techniques such as fault trees [26] and why-because-graphs [119], and tabular models learned by machine learning algorithms such as decision trees. As such, we phrase the corresponding research question.

*RQ5: How can existing domain-specific models enable practical causal modeling for purposes of accountability? (answered by examples in Chapter 6 and Chapter 7)*

Contextualization refers to the description of the circumstances around a particular event as an assignment of values to variables. From an operational perspective, context-setting roughly includes system monitoring and logging. We refer to monitoring as the mechanism of observing specific events and keeping a record, i.e., a log of these events. Usually, technical components such as programs or sensors generate log statements that describe the occurrence of an event at different levels of granularity. Clearly, logging brings an overhead to systems. The overhead starts at the design time by incurring some development activities for logging and continues at run-time with more disk operations. Lastly, during the log analysis phase, we end up with a massive number of events logged [59]. Hence, we need to think about the logs' properties— the content, the granularity, the frequency, and the

format– beforehand. To the best of our knowledge, the current state of research lacks practical, practitioner-friendly, (semi-) automated methodologies to define logging and monitoring requirements specifically for domains related to accountability (*Gap 4*, analysis of the related work that identified this gap is presented in Section 10.4).

*RQ6: How can we balance the trade-off between logging exhaustiveness and practical analysis of logs?* (answered by an example in Chapter 8)

## 1.2 Goal and Benefits

Accountability mechanisms, which employ automated causal reasoning to determine responsible parties for specific observations, are needed. Thus, the overall goal of this thesis is to enable accountability in modern systems by constructing such mechanisms. From our perspective, this entails proposing a general framework to *operationalize* actual causality reasoning. We realize our framework with practical, efficient, effective, and sound algorithms, methodologies, and tools to address problems of *causal modeling, contextualization, and reasoning*. Figure 1.3 shows a preliminary view of these tasks; throughout the thesis, we will define each part of this architecture. Consequently, such activities can be orchestrated by different stakeholders to achieve accountability, and its related concepts like responsibility [33], explainability [141], and fairness [141].

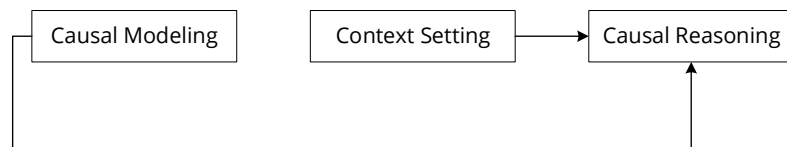


Figure 1.3: An Abstract Architecture of the Solution; reasoning is dependent on the output of causal modeling and contextualization.

Although we aim for a general framework of accountability, its benefits are only clear when considered in a specific domain of applications. For instance, since cyber-physical systems such as drones or airplanes may harm people when they fail, building such systems to be *accountable* is a necessity [95, 120, 104, 167]. In security-critical systems, if preventive measures fail, especially with attacks from the inside [101], accountability, in the sense of attack attribution, is potentially a deterrent measure. In the case of a drone crash, it is imperative to find and address the root cause to prevent future mishaps; in aircraft

accidents, accountability is part of the judicial process to assign liability and responsibility. In the third domain of applications, systems that are trained to make predictions, decisions, and classifications of humans based on data, often require methods to explain their results. For example, when a bank uses machine learning to determine a customer’s loan adequacy, accountability, in the sense of explaining system decisions, contributes to the trust, transparency, compliance, and debugging of these systems [141].

### 1.3 Solution

As mentioned above, in this thesis, we propose an approach to enable systems to reason about the causes of unwanted behavior, thus, be accountable. We realize our aim through the design, implementation, and evaluation of a chain of methods that tackle three different problems related to *causal modeling, contextualization, and reasoning*. In the course of this, we study different domains of applications where accountability can be partially or fully utilized. We derive a methodology to enable accountability for purposes of security, safety, and explainability. Our approach aids in understanding what is attribute specific (e.g., security, safety) and what is domain-specific (e.g., insiders in information systems), and what is general. Abstractly speaking, to achieve this goal, we use two central ideas.

1. Expressing the problem of actual causality reasoning as a combinatorial optimization problem. Consequently, we employ state-of-the-art techniques to compute causality and reflect on the properties of the solution from the causality perspective.
2. Studying the correspondence between causal models and domain-specific models to identify potential sources of knowledge. Then, we propose methods to create and contextualize causal models either by transformation or automatic generation.

### 1.4 Contributions

In this thesis, we describe the different ingredients that are required to achieve our aim of enabling accountability in systems. The focus, however, is put on the technical and practical utilization of the actual causality foundation as a cornerstone to achieve our goal. To that end, the contributions of this research are the following:

1. **Causal Reasoning.** To close *Gap 2*, we conceptualize a novel robust framework to reason about actual causality in acyclic models. The framework comprises the following set of algorithms that we designed, implemented, and evaluated:
  - a sound encoding of causality checking queries in binary models using Boolean satisfiability problem (Chapter 3),
  - several sound reformulations of the SAT encoding to deal with limitations of the original approach, and to support essential concepts from the causality theory such as the degree of responsibility (Chapter 3),

- an approach to formulate causality checking, over binary acyclic models, as an optimization problem, based on quantifiable notions within counterfactual computations. We contribute and compare two compact, non-trivial, and sound integer linear programming (ILP) and Maximum Satisfiability (MaxSAT) encodings to check causality, and better they can determine a *minimal* cause from a potentially non-minimal candidate cause (Chapter 4),
  - a multi-objective optimization problem formulation of causality inference in binary models that utilizes the degree of responsibility (Chapter 4), and
  - a generalization of the single-objective optimization problem formulation that eliminates the limitation to binary models to check actual causality in numerical models (Chapter 5).
2. **Causal Modeling and Contextualization.** These are two generic problems that cannot be addressed with general solutions. Hence, in this thesis, we show how to address them in a domain-specific context. To close *Gap 3 and Gap 4*, we propose and evaluate practical, effective, and (semi-)automated methods for causal modeling and contextualization to support causal reasoning in the domain of microservice-based information systems. Specifically, we contribute:
- an approach to construct causal models by extracting knowledge from existing threat models, or automatically generating these models (Chapter 6 and Chapter 7), and
  - a methodology that advocates on the right level of abstraction, granularity, frequency, and specificity of logging for purposes of contextualization (Chapter 8).
3. **A Unifying Framework.** To fill *Gap 1* and generalize the solution of filling *Gap 3 and Gap 4*, we propose a unifying methodology to operationalize actual causality activities and demonstrate its accountability purposes (Chapter 9). The methodology is supported with:
- a general-purpose, interactive platform called the *Actual Causality Canvas* (short: *Canvas*), which supports discovering and updating the system's knowledge of causal relations, automates contextualization, and facilitates reasoning, and
  - four instantiations of the framework in such areas as aircraft accidents, microservices systems, unmanned aerial vehicles, and artificial intelligence (AI) systems for purposes of forensic investigation, fault diagnosis, and explainable AI.

Parts of the contributions of this thesis have previously appeared in the following peer-reviewed publications, co-authored by the author of this thesis (ordered according to their appearance in the following chapters):

1. **Ibrahim, A.,** Rehwald, S., Pretschner, A. (2019). Efficient Checking of Actual Causality with SAT Solving. *Engineering Secure and Dependable Software Systems*, 53, 241.

2. **Ibrahim, A.**, Pretschner, A. (2020) From Checking to Inference: Actual Causality Computations as Optimization Problems. In Automated Technology for Verification and Analysis. ATVA 2020. Lecture Notes in Computer Science, vol 12302. Springer, Cham.
3. **Ibrahim, A.**, Rehwald, S., Scemama, A., Andres, F., Pretschner, A. (2020). Causal Model Extraction from Attack Trees to Attribute Malicious Insider Attacks. In International Workshop on Graphical Models for Security. GramSec 2020. Lecture Notes in Computer Science, vol 12419. Springer, Cham.
4. **Ibrahim, A.**, Kacianka, S., Pretschner, A., Hartsell, C., Karsai, G. (2019, May). Practical Causal Models for Cyber-physical Systems. In NASA Formal Methods Symposium (pp. 211-227). Springer, Cham.
5. **Ibrahim, A.**, Bozhinoski, S., Pretschner, A. (2019, April). Attack Graph Generation for Microservice Architecture. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (pp. 1235-1242).
6. **Ibrahim, A.**, Klesel, T., Zibaei, E., Kacianka, S., Pretschner, A. (2020). Actual Causality Canvas: A General Framework for Explanation-based Socio-Technical Constructs. In Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (pp. 2978 - 2985).

In addition, the author of this thesis has co-authored the following peer-reviewed publications, which tackle relevant problems related to the topic of this thesis but are not part of this thesis:

1. Ahmadvand, M., and **Ibrahim, A.** (2016, September). Requirements reconciliation for scalable and secure microservice (de) composition. In 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW) (pp. 68-73). IEEE.
2. Rehwald, S., **Ibrahim, A.**, Beckers, K., Pretschner, A. (2017). Accbench: A framework for comparing causality algorithms. In Proceedings of Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies CREST 2017, Sweden, (pp. 16–30).
3. Kacianka, S., **Ibrahim, A.**, Pretschner, A., Trende, A., Lüdtkke, A. (2019). Extending Causal Models from Machines into Humans. In 4th Workshop on Formal Reasoning about Causation, Responsibility, and Explanations in Science and Technology.
4. Kacianka, S., **Ibrahim, A.**, Pretschner, A. (2020). Expressing Accountability Patterns using Structural Causal Models. arXiv preprint arXiv:2005.03294. Under review.



## 1.5 Summary of Results

The main results obtained from the research conducted as part of this thesis are:

1. When limiting ourselves to binary causal models, actual causality is checked and inferred efficiently with large models. As we show in Chapter 3 and Chapter 4, different flavors of causality computation that cater to different requirements of causal queries vary in their efficiency.
2. Actual causality queries, especially when presented as optimization problems, can be answered within seconds even for large models, and they can be tailored to search for actual causes intelligently. Also, quantifiable notions related to causality are an essential factor to enable causality inference because they provide a sound foundation to compare causes.
3. When considering non-binary causal models such as numeric causal models, actual causality computations are efficient and scale to an acceptable size (around 2000 variables) of causal models.
4. Causal modeling and contextualization are effectively operationalized utilizing domain-specific artifacts. The domain of micro-service-based systems, for instance, benefits from enabling accountability in the context of insiders. Attack trees, depicting potential attacks by insiders, are a useful template to create causal models semi-automatically. The automation considers domain-specific attributes such as collusion attacks, and preemption. Further, such templates and, in turn, causal models can be automatically generated. This practical operationalization of modeling enhances contextualization as well. In this domain, contextualization is effectively and efficiently achieved through model-driven methodologies that advocate on the right level of logging.
5. Given general and efficient reasoning mechanisms and practical modeling and contextualization approaches, the operationalization of actual causality theories can be integrated into modern systems with minimal effort, using our unifying framework.

## 1.6 Structure

In addition to the introduction given in this chapter, Chapter 2 provides the required details to understand the theory of actual causality.

Then, in the second part of this thesis, we tackle the computational aspects of actual causality. Specifically, in Chapter 3, we present a set of SAT-based algorithms to handle the problem of causality checking. Then, in Chapter 4, we introduce methods to formulate the causality checks as optimization problems, opening the door to support other notions of reasoning. Chapter 5 is the last chapter in the second part of this thesis; it presents an

approach that generalizes the ideas from Chapter 3 and Chapter 4 to support non-binary models.

Causal modeling and contextualization are crucial to enable causality reasoning. Although they are generic problems, they can only be solved domain-specifically. Thus, in the third part of the thesis, we show how our reasoning technology can be applied leveraging approaches to automate modeling and contextualization in the domain of microservices. In Chapter 6, we show how to extract causal models from attack trees to attribute insider attacks. Then, in Chapter 7, we present an idea of how causal models can be automatically generated from network architectures and other sources. In Chapter 8, we describe an approach that leverages the models from previous chapters to guide the activities of monitoring and logging in modern information systems.

In the fourth part of this thesis, we present a generic framework (Chapter 9) that unifies our contributions to operationalize actual causality reasoning. Lastly, in the fifth part of this thesis, we overview the related work in Chapter 10, discuss the limitations and conclusions of the thesis in Chapter 11. We provide additional details related to our experiments in Appendix A.

## 2 Background

*This chapter presents the formal foundations required to understand the concept of actual causality.*

### 2.1 Overview

The Halpern-Pearl definition of (actual) causality (HP) is a well-known approach based on counterfactual reasoning and structural equations. Published for the first time in [79, 80] (*original* HP definition), the HP definition was updated in [81, 83] (*updated* HP definition) and, more recently, again modified in [74] (*modified* HP definition). We use the modified HP definition because it is simpler, solves problems described by various other authors, leads to (more) reasonable results for some causal scenarios, and reduces computational complexity [74].

The counterfactual view on causality goes to Hume (1748) [92]. Informally, *counterfactual reasoning* is thinking of alternative worlds, where (counter to the fact) if the cause is removed, the effect does not happen. A typical example of counterfactual reasoning is observed among sports fans [62]. Typically, fans of losing sports teams tend to describe scenarios where their team loss, would not have occurred had some events changed during the game. For example, had *Roberto Baggio* (not) missed the penalty kick, *Italy would not have lost the world cup in 1994*. Counterfactual reasoning is simply a but-for test, i.e., “but for the existence of  $X$ , would  $Y$  have occurred?” Although the test seems straightforward, answering such queries about a system is metaphysical; we do not possess the ability to create and manipulate alternative worlds. Thus, structural-equation models (details in Section 2.2) are introduced to formalize the knowledge and the counterfactual reasoning.

Counterfactual causality has two notions [75]. The first is general (type) causality, which is concerned with reasoning about *effects of causes*. For example, a statement such as *fatigue causes accidents* is a type causal statement, which aids scientists in predicting and prevent future accidents. Type causality, as seen by Halpern [75], is forward-looking since it is used to predict future effects. The second notion of causality is the *actual causality*. Actual causality “is the retrospective causal attribution of specific past events.”<sup>1</sup> For example, this sentence from the press release around the Hudson incident (also known as Miracle on

---

<sup>1</sup>Source: a talk by Christopher Hitchcock at Judea Pearl Symposium(2010), <https://www.youtube.com/watch?v=FfPYZM6Avag>, last accessed: July 2020

the Hudson incident)<sup>2</sup> is an actual causation statement “... the ingestion of large birds into each engine, ... resulted in an almost total loss of thrust in both engines.” In contrast to type causality, such statements are important when aiming to attribute blame to a specific event.

Accountability requires actual causality. Our definition of accountability entails answering questions regarding unwanted events. Intuitively, this means that we are considering a specific outcome that has occurred and are trying to find its cause. To deal with problematic cases and limitations of but-for tests,<sup>3</sup> Halpern and Pearl introduced their definition of actual causality.

In this chapter, a short overview of the latest, *modified* version of the HP definition of causality is given [74]. This includes the structure of causal models (Section 2.2), the language and the definition of actual causality (Section 2.3), and the extension of the definition to include responsibility (Section 2.4). Finally, a discussion of the advantages and challenges of HP concludes the chapter (Section 2.5). This chapter can be skipped for a reader with previous knowledge of HP.

## 2.2 Causal Models

HP uses variables to describe the world and *structural equations* to define its mechanics [155]. The variables are split into *exogenous* and *endogenous*. The values of the former, called a *context*  $\vec{u}$ , are governed by factors that are not part of the modeled world (they represent the environment). The endogenous variables, in contrast, are determined by equations of exogenous and endogenous variables. In this formulation, we look at causes within a specified universe of discourse represented by the endogenous variables, while exogenous variables are not considered to be part of a cause but rather as given information. An equation represents the semantics of the dependency of the endogenous variable on other variables.

Consider the following example (adapted with some changes from [155]): We observe that the grass in our garden is wet and know that there are only two possible causes for that, namely the sprinkler or rain. We could define the three variables  $GW$  for “grass is wet”,  $S$  for “sprinkler was on” and  $R$  for “it was raining”. We could then say that  $GW$  takes on value 1 if  $S$  or  $R$  are 1 (and not 0) and that the values of  $S$  and  $R$  are defined by two exogenous variables  $S_{exo}$  and  $R_{exo}$ . That is, we would have three structural equations, each of which determines the value of one of the variables. To formally define a *causal model*, we first need to introduce and define the term *signature*.

---

<sup>2</sup>US Airways Flight 1549 which, in the climbout after takeoff from LaGuardia Airport on January 15, 2009, struck a flock of Canada geese just northeast of the George Washington Bridge and consequently lost all engine power. Unable to reach any airport, pilots Chesley Sullenberger and Jeffrey Skiles glided the plane to a ditching in the Hudson River. All 155 people aboard were rescued.

[https://www.ntsb.gov/news/press-releases/Pages/CREW\\_Actions\\_and\\_Safety\\_Equipment\\_Credited\\_with\\_Saving\\_Lives\\_in\\_US\\_Airways\\_1549\\_Hudson\\_River\\_Ditching\\_NTSB\\_Says.aspx](https://www.ntsb.gov/news/press-releases/Pages/CREW_Actions_and_Safety_Equipment_Credited_with_Saving_Lives_in_US_Airways_1549_Hudson_River_Ditching_NTSB_Says.aspx)

<sup>3</sup>A discussion of the limitations of but-for testing can be consulted in [123, 122].

**Definition 2.1. Signature [157]:** A signature  $\mathcal{S}$  is a tuple  $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ , where

- $\mathcal{U}$  is a set of exogenous variables,
- $\mathcal{V}$  is a set of endogenous variables and
- $\mathcal{R}$  associates with every  $Y \in \mathcal{U} \cup \mathcal{V}$  a nonempty set  $\mathcal{R}(Y)$  of possible values for  $Y$ .

Distinguishing between *exogenous* and *endogenous* variables allows focusing on causal relationships between selected variables. Kuntz et al. describe an example, which illustrates the usefulness of this distinction [117]. We want to examine the cause of a concrete accident at a railway crossing. Since we do not want the decision of the train engineer union not to strike on that day to be considered as a potential cause of the accident, we can model this decision as an exogenous variable. It is thus assumed as given for the causal analysis and we can focus on more relevant variables like the fact that the gates did not close in time. This property should, therefore, be modeled as an endogenous variable. We can now define causal models.

**Definition 2.2. Causal Model [157]:** A causal model  $M$  is a tuple  $M = (\mathcal{S}, \mathcal{F})$ , where

- $\mathcal{S}$  is a signature and
- $\mathcal{F}$  associates with each endogenous variable  $X \in \mathcal{V}$  a function
 
$$F_X : (\times_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\times_{Y \in \mathcal{V} - \{X\}} \mathcal{R}(Y)) \rightarrow \mathcal{R}(X)$$

The set of functions  $\mathcal{F}$  in a causal model describes a set of *structural equations*. Definition 2.2 makes precise the fact that  $F_X$  determines the value of  $X$ , given the values of all the other variables. The structural equations formalize inter-dependencies among variables. In their examples, Halpern and Pearl focus their models to *recursive (acyclic)* equations. However, Halpern also provided an extension of the definition that generalizes to cyclic ones.

**Definition 2.3. Acyclic Equations [157]:** If  $F_X(\dots, y, \dots) = F_X(\dots, y', \dots)$  for all  $y, y' \in \mathcal{R}(Y)$ , then we call  $F_X$  independent of  $Y$  and we write  $X \prec Y$ . A causal model  $\mathcal{M} = (\mathcal{S}, \mathcal{F})$  with signature  $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$  is *recursive (acyclic)*, if there is a total ordering  $\prec$  on the variables in  $\mathcal{V}$ .

In this thesis, we limit ourselves to acyclic models. If a causal model is acyclic, there is always a unique solution to the equations given the values of the exogenous variables;<sup>4</sup> we refer to the solution as the *actual evaluation* of the model. The solution is obtained by solving the equations for the endogenous variables in the order defined by  $\prec$ . A causal model is visualized as *causal networks*—a graph with the variables  $\mathcal{U} \cup \mathcal{V}$  as nodes. There is an edge from a node  $X$  to a node  $Y$  in the causal network, if  $F_Y$  depends on  $X$  in the causal model, i.e.,  $X$  appears on the right side of the equation defining  $F_Y$ . For recursive models, the causal network is a *directed, acyclic graph*.

<sup>4</sup>This, however, does not mean that a unique cause necessarily exists. As we shall see in Section 2.3, within a unique solution, one or multiple causes may satisfy the conditions in Definition 2.4.

## 2.3 Reasoning about Causality

In this section, we present the formal language used by Halpern and Pearl ([78], [82]). These notations are necessary for the definition of an actual cause, which follows in Definition 2.4.

A *primitive event* is a formula of the form  $X = x$ , for  $X \in \mathcal{V}$  and  $x$  is its value, e.g.,  $\in \{0, 1\}$  for binary models. A sequence of variables  $X_1, \dots, X_n$  is abbreviated as  $\vec{X}$ . Analogously,  $X_1 = x_1, \dots, X_n = x_n$  is abbreviated  $\vec{X} = \vec{x}$ .  $\varphi$  is a Boolean combination of such events.  $(M, \vec{u}) \models X = x$  if the variable  $X$  has value  $x$  in the unique solution to the equations in  $M$  given context  $\vec{u}$  (values of the exogenous variables). The value of a variable  $Y$  can be overwritten by a value  $y$  (known as an intervention) writing  $Y \leftarrow y$  (analogously  $\vec{Y} \leftarrow \vec{y}$  for vectors). Then, a causal formula is of the form  $[Y_1 \leftarrow y_1, \dots, Y_k \leftarrow y_k]\varphi$ , where  $Y_1, \dots, Y_k$  are distinct variables in  $V$  that make  $\varphi$  hold when they are set to  $\vec{y}$ . We write  $(M, \vec{u}) \models \varphi$  if the causal formula  $\varphi$  is true in  $M$  given context  $\vec{u}$ . Lastly,  $(M, \vec{u}) \models [\vec{Y} \leftarrow \vec{y}]\varphi$  holds if we replace the equations for the variables in  $\vec{Y}$  by equations of the form  $Y = y$  denoted by  $(M_{\vec{Y}=\vec{y}}, \vec{u}) \models \varphi$  [75]. That is the act of intervention means the replacement of the equation in the model with an equation that hard-codes the value to a fixed value.

A causal formula  $\psi$  can be evaluated to true or false in a causal model  $M$  given a context  $\vec{u}$ . We write  $(M, \vec{u}) \models \psi$  if  $\psi$  evaluates to true in the causal model  $M$  given context  $\vec{u}$ . This is, the statement  $(M, \vec{u}) \models [\vec{Y} \leftarrow \vec{y}](X = x)$  implies that solving the equations in the submodel  $M_{\vec{Y}=\vec{y}}$  with context  $\vec{u}$  yields the value  $x$  for variable  $X$ .

All three versions of HP ([78], [82], [74]) have the same structure. They consist of three clauses AC1, AC2, and AC3 (AC stands for *actual cause*). The AC1 and AC3 rules have stayed the same throughout the three versions, only AC2 has changed.

**Definition 2.4. Actual Cause** (latest/modified version [74])

$\vec{X} = \vec{x}$  is an actual cause of  $\varphi$  in  $(M, \vec{u})$  if the following three conditions hold:

**AC1.**  $(M, \vec{u}) \models (\vec{X} = \vec{x})$  and  $(M, \vec{u}) \models \varphi$ .

**AC2.** There is a set  $\vec{W}$  of variables in  $\mathcal{V}$  and a setting  $\vec{x}'$  of the variables in  $\vec{X}$  such that if  $(M, \vec{u}) \models \vec{W} = \vec{w}$ , then  $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}]\neg\varphi$ .

**AC3.**  $\vec{X}$  is minimal, i.e., no subset of  $\vec{X}$  satisfies conditions AC1 and AC2.

With AC1, it is ensured that the events  $\vec{X} = \vec{x}$  are only considered as a cause of  $\varphi$  if both occurred, i.e., the cause is sufficient for the occurrence of the effect. Formally:  $\vec{X} = \vec{x}$  can only be a cause of  $\varphi$ , if  $\vec{X} = \vec{x}$  and  $\varphi$  are true under  $(M, \vec{u})$ . For example, the sprinkler can only be a cause for the grass being wet if the grass is actually wet, and the sprinkler was on. AC2 checks the counterfactual (necessary) relation between the cause and effect. It holds if there exists a setting  $\vec{x}'$  for the cause variables  $\vec{X}$  different from the actual evaluation  $\vec{x}$  (in binary models such a setting is the negation of the actual setting [98]), and another set of variables  $\vec{W}$ , referred to as a contingency set, that we use to *fix* variables at their actual values, such that  $\varphi$  does not occur anymore. The contingency set  $\vec{W}$  is meant to deal with issues such as preemption and redundancy. Preemption is a problematic situation where

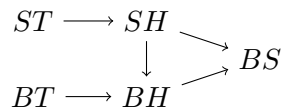
multiple possible causes coincide (illustrated by an example below) [126]; thus a naive counterfactual check cannot determine the cause [123]. AC3 checks that  $\vec{X}$  is minimal in fulfilling the previous conditions (AC1 and AC2). To check a cause, we need to think of two worlds (variable assignments): the *actual* world with all the values known to us, and the counterfactual one in which the *cause* and *effect* take on different values. Two factors further complicate the search for this counterfactual world. First, finding an arbitrary  $\vec{W}$ , such that AC2 holds which is exponential in the worst case. Second, no (non-empty) subset of the cause variables is sufficient for constructing such a counterfactual world. The tuple  $(\vec{W}, \vec{w}, \vec{x}')$  is called a *witness* of  $\vec{X} = \vec{x}$  being a cause of  $\varphi$ . The role of the contingency set  $\vec{W}$  becomes more clear when considering one of Halpern and Pearl’s [79, 81] examples described in the following: (due to Lewis [126]).

Suzy and Billy both throw a rock on a bottle which shatters if one of them hits. Furthermore, we know that Suzy’s rock hits the bottle slightly earlier than Billy’s and both are accurate throwers. In [79, 81, 74], this world is modeled with the following variables (exogenous variables are excluded). The causal model is depicted in Figure 2.1.

- $ST$  for “Suzy throws”, with values 0 (Suzy does not throw) and 1 (she does),
- $BT$  for “Billy throws”, with values 0 (he doesn’t) and 1 (he does),
- $BH$  for “Billy’s rock hits the (intact) bottle”, with values 0 (it doesn’t) and 1 (it does),
- $SH$  for “Suzy’s rock hits the bottle”, again with values 0 and 1,
- $BS$  for “bottle shatters”, with values 0 (it doesn’t shatter) and 1 (it does),

and equations

- $BS$  is 1 iff one of  $SH$  and  $BH$  is 1, i.e.,  $BS = SH \vee BH$ ,
- $SH$  is 1 if  $ST$  is 1, i.e.,  $SH = ST$ ,
- $BH = 1$  if  $BT = 1$  and  $SH = 0$ , i.e.,  $BH = BT \wedge \neg SH$ .



Each node  $A$  represents an endogenous variable; an edge from  $A$  to  $B$  means that  $B$  “depends” on  $A$ .

Figure 2.1: Rock-Throwing Example (Source: [79])

Let us assume that we are given a context  $\vec{u}$  that sets  $ST = 1$  and  $BT = 1$ . As a result, we have  $SH = 1$ ,  $BH = 0$  and  $BS = 1$ . We now want to find out whether  $ST = 1$ ,  $BT = 1$ , or both are a cause for  $BS = 1$ . We begin by checking whether  $ST = 1$  is a cause. Obviously, AC1 is fulfilled as both  $ST = 1$  and  $BS = 1$  actually happened in our example. Since  $ST$  (and all other variables likewise) can only take on two different values, the only possibility for a setting  $\vec{x}'$  for  $ST$  is 0, i.e., Suzy does not throw. A first attempt with  $\vec{W} = \emptyset$  shows that AC2 does *not* hold: If  $ST = 0$ , then  $SH = 0$  such that  $BH$  changes to 1 as we did not change  $BT$  and ultimately  $BS$  is still 1. However, the HP definition allows us to define  $\vec{W} = \{BH\}$ , i.e., we replace the original equation of  $BH$  with  $BH = 0$  which was its value in the original scenario. Now, AC2 holds as  $BS = 0$  and so does AC3 as our cause  $ST = 1$  already consists of a single primitive event only. This example illustrates the use of  $\vec{W}$ . In the original scenario, Billy did not hit the bottle, because Suzy has hit it before. Hence, [74] argues that it is reasonable to keep some variables at their values of the original context even though the causal model and its equations alone would define another value. Taking a look at  $BT = 1$  as cause for  $BS = 1$ , we can immediately see that AC1 is fulfilled as well. Similar to  $ST$ , the only possible setting  $\vec{x}'$  for  $BT$  is 0. However, this does not affect  $BS$  and it is also not possible to find a  $\vec{W}$  such that  $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}] \neg \varphi$  would hold. That is,  $BT = 1$  is *not* a cause. Also,  $ST = 1 \wedge BT = 1$ , i.e., the conjunction of both, is not a cause, since it would not fulfill AC3: There is a subset, namely  $ST = 1$ , which is a cause by itself.

In the previous example, we see that due to the fact that Suzy *did* throw a rock, the latter action is considered as a cause. However, Halpern points out that also the opposite, i.e., that something does *not* happen, can be considered as cause according to the HP definition in some scenarios. The following example shows the concept [75]:

Billy, having stayed out in the cold too long throwing rocks, contracts a serious but nonfatal disease. He is hospitalized and treated on Monday, so he is fine Tuesday morning.

The story is modeled using variables  $MT$  (“Monday treatment”), which is 1/0 if Billy does/does not get his treatment on Monday, and  $BMC$  (“Billy’s medical condition”), which is 0/1 if Billy did/did not recover on Tuesday morning. Even though the author does not explicitly specify the equations, we can see that  $BMC$  must be defined as  $BMC = \neg MT$ , i.e.,  $BMC = 1$  if  $MT = 0$ . In his example, Halpern now assumes a scenario in which Billy is sick on Monday, but the doctor does not treat him. Hence,  $MT = 0$  and  $BMC = 1$ . Asking whether  $MT = 0$  causes  $BMC = 1$ , we can easily see that it fulfills AC1, AC2, and AC3 and is thus a cause according to the HP definition: AC1 and AC3 trivially hold; for AC2, we change  $MT = 0$  to  $MT = 1$ , i.e., the doctor now *does* treat Billy, which leads to  $BMC = 0$ , i.e., Billy recovered on Tuesday. This shows that also not doing something or the non-occurrence of an event can be a cause.<sup>5</sup>

---

<sup>5</sup>These examples are only two out of many others in [74] as well as [79, 80, 81, 83] and [75]. For a thorough understanding of the HP definition it might therefore be helpful to consider the mentioned papers.



## 2.4 Responsibility as an Extension

As opposed to the all-or-nothing treatment of causality, Chockler and Halpern added ([33], modified in [75]) a notion of *responsibility* to a cause. They introduced a metric, degree of responsibility ( $dr$ ), that “measures the minimal number of changes needed to make  $\varphi$  counterfactually depend on  $X$ .” Their idea is often motivated with an example of 11 voters that can vote for Suzy or Billy. If Suzy wins 6-5, we can show that each Suzy voter is a cause of her winning. If Suzy wins 11-0, then each subset of size six of the voters is a cause. The authors argue that in 11-0 scenario, “a voter feels less responsible” compared to 6-5 situation. Definition 2.5 shows  $dr$  [33, 75], which we use for causality inference in our work.

**Definition 2.5. (Degree of Responsibility).** *The degree of responsibility of  $X = x$  for  $\varphi$  in  $(M, \vec{u})$  according to the modified HP definition denoted  $dr((M, \vec{u}), (X = x), \varphi)$ , is 0 if  $X = x$  is not part of a cause of  $\varphi$  in  $(M, \vec{u})$  according to the modified HP definition; it is  $1/k$  if there exists a cause  $\vec{X} = \vec{x}$  of  $\varphi$  and a witness  $(\vec{W}, \vec{w}, \vec{x}')$  to  $\vec{X} = \vec{x}$  being a cause of  $\varphi$  in  $(M, \vec{u})$  such that (a)  $X = x$  is a conjunct of  $\vec{X} = \vec{x}$ , (b)  $|\vec{W}| + |\vec{X}| = k$ , and (c)  $k$  is minimal, in that there is no cause  $\vec{X}_1 = \vec{x}_1$  for  $\varphi$  and a witness  $(\vec{W}', \vec{w}', \vec{x}'_1)$  to  $\vec{X}_1 = \vec{x}_1$  being a cause of  $\varphi$  in  $(M, \vec{u})$  according to the modified HP definition that includes  $X = x$  as a conjunct with  $|\vec{W}'| + |\vec{X}_1| < k$ .*

Since in the 11-0 voting scenario each voter is part of a cause  $\vec{X} = \vec{x}$ ,  $|\vec{X}| = 6$  and we can show that  $\vec{X} = \vec{x}$  is a cause for Suzy winning the vote with  $\vec{W} = \emptyset$ , the responsibility of each voter is  $1/6$ . Taking the rock-throwing example, the responsibility of  $ST = 1$  is  $1/2$ , because we had  $\vec{W} = \{BH\}$ , and the responsibility of  $BT = 1$  is 0 because we showed that Billy’s throw is not a cause according to the HP definition.

## 2.5 Discussion

HP is probably one of the most referred definitions of causality in computer science research. Halpern and Pearl’s main contribution is to give a very general and broadly applicable definition. Gössler and Le Metayer’s trace-based approach [66], is, for example, inherently well suited for real-time systems, but is not trivially applicable to other types of systems. Leitner-Fischer and Leue [123] on the other hand gear their work to applicability in the automotive industry. In contrast to these causality definitions, HP is very open towards possible fields of application.

One reason for HP’s variability is the simple structure the definition is based upon. Every world that can be described using random variables for its properties and combining the variables in structural equations to define the world’s mechanisms, can be modeled with the definition. To reason over causal relationships in the world, (partial) observability of the values of the variables is additionally needed.

The HP definition was decisively affected by the problems other approaches encountered. Therefore, the qualities of the definition manifest particularly in comparison to other

definitions. The following is a list of problems that HP deals with well according to Halpern and Pearl [82].

- Distinguishing between exogenous and endogenous variables, at first sight, does not appear to be revolutionary. However, this distinction enables the choice of what to count as a possible cause (*endogenous*) and what not to (*exogenous*). Hence, it treats cases of *irrelevance*. Consequently, it allows us to limit our attribution based on the goal. If we are looking for legal evidence, then we can include possible human actors in the set of endogenous variables. If we are looking for an explanation of an intrusion, then we can include the running services as endogenous variables. Furthermore, HP correctly classifies the *non-occurrence* of events as causes. For example, an administrator “forgetting” to install the latest update of the firmware on a server can be a cause of an exploit.
- A typical problem of causality definitions, which HP deals with, is *preemption*. It resembles the confusing cases where several potential causes exist and coincide, but one cause preempts the others. The problem for simple counterfactual definitions is that if the earlier cause  $A$  had not been there, cause  $B$  would have triggered the effect anyway (just a bit later). Thus,  $A$  is not classified as a cause. HP deals with this by using  $\vec{W}$  from Definition 2.4 and auxiliary variables. Accounting for preemption in accountable systems is beneficial. Consider, for example, security attacks with different strategies of attacking. For instance, an administrator copying a DB backup file, although this is a policy violation, is not the actual cause of the data breach that happened. The copy act was preempted by a privilege abuse of another employee. Further, differentiating actual causes in cases of preemption is crucial when preventive measures such as an intrusion detection system (IDS) are deployed. For example, an IDS may preempt an attack from succeeding although the basic steps of the attack were carried out.
- Conjunction and Disjunction as causes. HP can consider a combination of events as a cause. There are attacks that are carried out by multiple steps and hence are modeled using an *AND* gate. For example, to read a service’s memory, an attacker accesses the machine, then attaches a debugger to the running process. On the other hand, there are attacks that can be carried out using different techniques or by exploiting different vulnerabilities. For example, to steal the master key from a system, the attacker can either obtain it decrypted from memory *or* encrypted from the database (the attacker then has to decrypt it). A more interesting scenario would be if two agents cooperated in carrying out an attack, i.e., a collusion attack. Such attacks are a major threat class of insiders [109].
- There is a number of further problems such as double prevention, bogus prevention, causation by omission and commission, and trumping preemption. The HP definition is able to deal with those problems, sometimes with extensions to the definition. An

example of such extensions is the set of allowable settings for endogenous variables, presented in [82]. Another important extension is presented by Halpern and Hitchcock [77]; they introduce an addition to the HP definition considering *normality* to deal with the so-called *problem of disagreement*.

A challenge of the HP definition is the question of setting up a causal model. Halpern and Pearl themselves have shown several times the difficulties of coming up with a proper model and the considerable influence of the model on the result of the evaluation of causes [82]. Additionally, the size of the models in practice is significantly larger than the examples given by Halpern and Pearl. This raises the question of how to build big causal models properly and naturally leads to the question of (semi-) automatic model creation. In contrast to that, Gössler and Le Metayer's approach relies on program specifications for causal reasoning, which is simply closer to software design than the HP definition [66].

Lastly, the main HP challenge arises from the perspective of inferring causality. Apart from the formal correctness of the definition of causality, practical applicability is an important matter. In particular, the automated evaluation of causes for a certain event should be possible efficiently. Three papers have examined the question of complexity of the different versions of the HP definition. Eiter and Lukasiewicz [48] analyzed the first version of the definition [78]. Aleksandrowicz et al. [6] conducted the analysis for the updated version of the HP definition [82]. Halpern himself also discussed the complexity of his modified version of the definition when introducing it [74].

Halpern shows that under the *modified* HP definition, determining causality is in general (binary and non-binary models), i.e., given  $\vec{X} = \vec{x}$ ,  $D_1^P$ -complete and  $NP$ -complete given a singleton cause  $X = x$ . The family of complexity classes  $D_k^P$  with  $k = 1, 2, 3, \dots$  was introduced by [6]. Following them,  $D_k^P$  is a generalization of  $D^P (= D_1^P)$  introduced by [152], who show that  $NP \subseteq D^P$ . Specifically, checking  $AC1$  can be done in polynomial time ( $P$ ). However, checking  $AC2$  is  $NP$ -complete, and checking  $AC3$  is  $co-NP$ -complete. For binary models, complexity considerations may suggest a reduction to SAT or Integer Linear Programming [37, 107, 74]. Thus, causality checking using the HP definition is hard. While this might not be problematic for the rather basic examples in [74, 75] whose major purpose seems to be to help understanding the HP definition, in industry-relevant causal models regarding, for instance, aircraft or software systems, manually determining causes might not be a feasible option anymore.

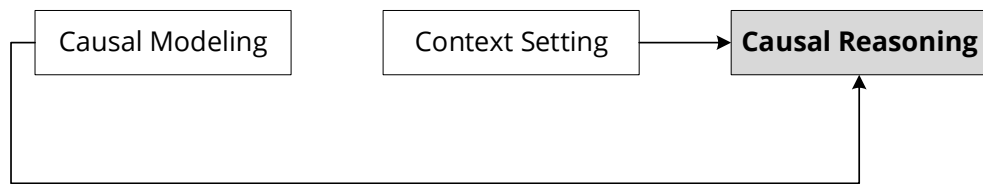
## 2.6 Summary

We presented the preliminaries for understanding this thesis. We pointed out the various aspects and concepts of the theory of actual causality proposed by the *modified* HP definition (Definition 2.4) and illustrated them using examples. We saw that it consists of three conditions  $AC1$ ,  $AC2$ , and  $AC3$ , all of which need to be fulfilled such that we call conjunction of primitive events  $\vec{X} = \vec{x}$  a cause for a combination of primitive events  $\varphi$  under a context  $\vec{u}$  in a causal model  $M$ . We discussed the advantages and challenges of the definition.



## Part II

# Computational Aspects of Actual Causality Reasoning





# 3 Efficiently Checking Actual Causality with SAT Solving

*This chapter presents a novel encoding of binary causality checking queries into SAT. Several other encodings are derived and evaluated as part of the chapter. Parts of this chapter have previously appeared in the publication [98], co-authored by the author of this thesis.<sup>1</sup>*

## 3.1 Introduction

As we have seen in Chapter 2, HP provides a definition of when we can call one or more events a cause of another event in a way that captures human intuition. When automated, this definition can, then, be used to answer causal queries in the postmortem of unwanted behavior; it is a vital ingredient to enable accountability. Causality checking, using any version of HP, is computationally hard. Halpern shows that under the *modified* HP definition, determining causality, i.e., computing AC1, AC2, and AC3, is in general  $D_1^P$ -complete and  $NP$ -complete given a singleton cause [75, 74].<sup>2</sup> In Table 3.1, the complexity of each single condition of the HP definition is detailed. As we can see, AC2 and AC3 are computationally hard because they are in  $NP$  and  $co-NP$ , respectively, while AC1 can be computed in polynomial time.

AC1	AC2	AC3
$P$	$NP$	$co-NP$

Table 3.1: Complexity of AC1, AC2, and AC3

The computational complexity led to a domain-specific (e.g., database queries, counterexamples of model checking), adapted (e.g., use lineage of queries, use Kripke structure

---

<sup>1</sup>Parts of this chapter are reprinted from Engineering Secure and Dependable Software Systems, 53, Ibrahim et al., Efficient Checking of Actual Causality with SAT Solving, 241 - 255, (2019), with permission from IOS Press. The publication is available at IOS Press through <http://dx.doi.org/10.3233/978-1-61499-977-5-241>

<sup>2</sup>Recall that the family of complexity classes  $D_k^P$  with  $k = 1, 2, 3, \dots$  was introduced by [6], who investigated the complexity of the *original* and *updated* HP definition. Following them,  $D_k^P$  is a generalization of  $D^P$  ( $= D_1^P$ ) proposed by [152], who show that  $NP \subseteq D^P$ .

of programs), or restricted (e.g., monotone queries, singleton causes, single-equation models) utilization of HP for binary models (details in Chapter 10). Conversely, brute-force approaches work with small models (less than 30 variables [89]) only. Therefore, to the best of our knowledge, there exists no comprehensive, efficient, and scalable framework for modeling and checking actual causality for binary models (i.e., models with binary variables only). Consequently, no existing algorithm allows applying HP on more complex examples than the simple cases in the literature.

In this chapter, we conceptualize a novel approach towards checking causality in acyclic binary models based on the Boolean satisfiability problem (SAT). We intelligently encode the core of HP as a SAT query that allows us to reuse the optimization power built into SAT solvers. As a consequence of the rapid development of SAT solvers (1000X+ each decade), they offer a promising tactic for any solution in formal methods and program analysis [147]. Leveraging this power in causality establishes a robust framework for efficiently reasoning about actual causality. Moreover, since the transformation of SAT to other logic programming paradigms like answer set programming (ASP) or integer linear programming is almost straightforward, this approach in this chapter establishes the ground to tackle more causality issues (e.g., causality inference) using combinatorial solving approaches.

In this chapter, we present our approach towards checking causality in binary models. We begin by introducing a Brute-Force algorithm. Then, we present a SAT-based approach to check actual causality over binary models. It includes **three** SAT-encodings that are formally proved to reflect HP and two variants for optimization. We also show an empirical evaluation that uses different examples to show the efficiency and scalability of our approach.

### 3.2 Brute-Force Based Causality Checking

Both for the sake of comparison and demonstrating the complexity of checking actual causality, we implemented a brute-force algorithm to check AC2 and AC3 of HP. AC1 verifies if the set of primitive events  $\vec{X} = \vec{x}$  and  $\varphi$  itself occurred in the evaluation of  $M$  under context  $\vec{u}$ . Hence, we only need to compute the values of all variables given the values of the exogenous variables as defined by  $\vec{u}$  and check whether the mentioned conditions hold. Therefore, we do not explicitly propose an algorithm for AC1.

For AC2, we need to determine a set of variables  $\vec{W}$  and a setting  $\vec{x}'$  for  $\vec{X}$  such that  $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}] \neg \varphi$  given  $(M, \vec{u}) \models \vec{W} = \vec{w}$ . For Boolean variables, we define  $\vec{x}'$  as the negation of  $\vec{x}$  (denoted by  $\neg \vec{x}$ ); this is a result of Lemma 3.2 which we present in Section 3.3. Then,  $\vec{W}$  remains the only variable part of AC2 which requires enumerating all possible combinations of variables. Before that, we can check whether AC2 already holds for  $\vec{W} = \emptyset$ . AC3, on the other hand, requires checking that the first two conditions do not hold for any subset of  $\vec{X}$ . Thus, the AC2 enumeration must be performed again, possibly, for every subset of the  $\vec{X}$ .



Algorithm 1 shows the two functions that correspond to the two conditions. The input includes the model  $M$ , the context (the set of exogenous variables and their values)  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ , the effect as a boolean combination of variables  $\varphi$ , the candidate cause  $\langle X_1, \dots, X_\ell \rangle = \langle x_1, \dots, x_\ell \rangle$ , and the evaluation of the endogenous variables  $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$ . Function *FulfillsAC2* starts by checking if  $\neg\varphi$  holds for an empty  $\vec{W}$  (Line 2). Otherwise, the function enumerates, in ascending order by size, the powerset of endogenous variables  $\mathcal{V} \in M$  that are not part of  $\vec{X}$  (Line 4). If we have found a  $\vec{W}_i$  for which AC2 holds, we return this set (Line 6). Otherwise, after iterating through the powerset, we return *false* (Line 10) indicating that AC2 does not hold. If,  $\vec{X} = \vec{x}$  is not a singleton cause and  $\varphi$  occurred, we check AC3 using *FulfillsAC3*. First, we compute the powerset of  $\vec{X} = \vec{x}$  containing all its possible subsets while excluding the empty set and the original cause. Then, we check if any one of them fulfills AC1 and AC2. Since we made sure that  $\varphi$  actually happened, i.e.,  $(M, \vec{u}) \models \varphi$ , it is sufficient for AC1 to analyze if all elements of the current subset of the cause actually happened as well, i.e.,  $(M, \vec{u}) \models (\vec{X}_i = \vec{x}_i)$ . In case we find a subset for which this condition holds, AC3 is violated (Line 16). Otherwise, we return that AC3 is fulfilled (Line 20).

Since the size of the power set  $\mathcal{P}(S)$  of a set  $S$  of size  $n$  is defined by  $|\mathcal{P}(S)| = 2^n$ , the potential number of iterations for checking AC2, within Algorithm 1, and its execution time increase exponentially with the number of endogenous variables in causal model  $M$ . Also, if  $\vec{X}$  consists of more than one element, the complexity of checking AC3 as proposed increases exponentially as well because we need to consider each subset of  $\vec{X}$ , which results in checking its power set. Consequently, we think it is reasonable and necessary to search for a more efficient approach.

### 3.3 SAT Based Causality Checking

We define the Boolean Satisfiability Problem, which is the first *NP*-complete problem [61], as follows.

**Definition 3.1. (Boolean Satisfiability Problem)[61, 180].** *The Boolean Satisfiability Problem (SAT) is defined as the question of whether there exists an assignment  $\alpha$  (mapping from the variables to the values 0 and 1) for a Boolean formula  $F$  such that  $F\alpha = 1$ . If so, we call  $F$  satisfiable, otherwise unsatisfiable.*

There are different variants of SAT, e.g., MaxSAT [128], UniqueSAT [21], or  $k$ -SAT [40]. Yet another variant is the so-called All-SAT problem, in which we want to identify not only *one* but *all* satisfying assignments of a formula. We will use this concept in Section 3.3.2. A possibility of obtaining all satisfying assignments is to iteratively constrain a given formula with the solutions found so far until no more satisfying assignments can be identified.

In this section, we propose our algorithmic approaches towards the HP definition. To answer a causal question efficiently, we need to find an intelligent way to search for

---

**Algorithm 1** Check whether HP holds (Brute-Force)

---

**Input:** causal model  $M$ , context  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ , effect  $\varphi$ , candidate cause  $\langle X_1, \dots, X_\ell \rangle = \langle x_1, \dots, x_\ell \rangle$ , evaluation  $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$

- 1: **function** FULFILLSAC2( $M, \vec{u}, \varphi, \vec{X} = \vec{x}$ )
- 2:   **if**  $(M, \vec{u}) \models [\vec{X} \leftarrow \neg \vec{x}] \neg \varphi$  **then return**  $\emptyset$
- 3:   **else**
- 4:     **for all**  $\vec{W}_i \in \mathcal{P}(\{V \mid V \in \mathcal{V}, V \notin \vec{X}\}) \setminus \emptyset$  **do**
- 5:       **if**  $(M, \vec{u}) \models [\vec{X} \leftarrow \neg \vec{x}, \vec{W}_i \leftarrow \vec{w}_i] \neg \varphi$  given  $(M, \vec{u}) \models \vec{W}_i = \vec{w}_i$  **then**
- 6:         **return**  $\vec{W}_i$
- 7:       **end if**
- 8:     **end for**
- 9:   **end if**
- 10: **return false**
- 11: **end function**
- 12: **function** FULFILLSAC3( $M, \vec{u}, \varphi, \vec{X} = \vec{x}$ )
- 13:   **if**  $|\vec{X}| > 1 \wedge (M, \vec{u}) \models \varphi$  **then**
- 14:     **for all**  $\vec{X}_i = \vec{x}_i \in \mathcal{P}(\vec{X} = \vec{x}) \setminus (\emptyset \cup (\vec{X} = \vec{x}))$  **do**
- 15:       **if** FULFILLSAC2( $M, \vec{u}, \varphi, \vec{X}_i = \vec{x}_i$ )  $\wedge (M, \vec{u}) \models (\vec{X}_i = \vec{x}_i)$  **then**
- 16:         **return false**
- 17:       **end if**
- 18:     **end for**
- 19:   **end if**
- 20: **return true**
- 21: **end function**

---

a  $\vec{W}$  such that AC2 is fulfilled as well as to check whether AC3 holds. Therefore, we propose an approach that uses SAT-solving. We show how to encode AC2 into a formula whose (un)satisfiability and thus the (un)fulfillment of AC2 is determined by a SAT-solver. Similarly, we show how to generate a formula whose satisfying assignments obtained with a solver indicate if AC3 holds.

### 3.3.1 Checking AC2

For AC2, such a formula  $F$  has to incorporate (1) the negation of the effect  $\neg \varphi$ , (2) the context  $\vec{u}$ , (3) a setting,  $\vec{x}'$  for candidate cause,  $\vec{X}$ , and (4) all possible variations of  $\vec{W}$ , while still (5) keeping the semantics of the underlying model  $M$ . In the following, we describe the concept and, then, the algorithm that generates such a formula  $F$ . Since we check actual causality in hindsight, we have a situation where  $\vec{u}$  and  $\vec{v}$  are determined, and we have a candidate cause  $\vec{X} \subseteq \vec{V}$ . Thus, the first two requirements are straightforward. First, the effect  $\varphi$  should not hold anymore, hence,  $\neg \varphi$  holds. Second, the context  $\vec{u}$  should be set to its values in the original assignment (the values  $\vec{u}$  of  $\vec{U}$ ).

Since we are treating binary models only, the setting  $\vec{x}'$  (from AC2) can be tailored down to negating the original value of each cause variable. This is a result of Lemma 3.2, which utilizes the fact that we are considering binary variables to exclude other possible settings and define precisely the setting  $\vec{x}'$ . The proof of the Lemma, along with the proofs of all theorems in this chapter, is given in Section 3.6. Thus, to address the third requirement, according to Lemma 3.2, for  $\neg\varphi$  to hold, all the variables of the candidate cause  $\vec{X}$  are negated.

**Lemma 3.2.** *In a binary model, if  $\vec{X} = \vec{x}$  is a cause of  $\varphi$ , according to Definition 2.4, then every  $\vec{x}'$  in the definition of AC2 always satisfies  $\forall i. x'_i = \neg x_i$ .*

To ensure that the semantics of the model are reflected in  $F$  (requirement 5), we use the logical equivalence operator ( $\leftrightarrow$ ) to express the equations. Particularly, to bind each endogenous variable  $V_i$  to its equation  $F_{V_i}$ , we use this clause  $V_i \leftrightarrow F_{V_i}$ . This way, we create a (sub-)formula that evaluates to true if both sides are *equivalent* in their evaluation. If we do so for all other variables (that are not affected by requirements 1-3), we ensure that  $F$  is only satisfiable for assignments that respect the semantics of the model.

Further, we need to find a possibility to account for  $\vec{W}$  (requirement 4) without having to iterate over the power-set of all variables. In  $F$ , we accomplish this by adding a *disjunction* with the positive or negative literal of each variable  $V_i$  to the previously described equivalence-formula, depending on whether the actual evaluation of  $V_i$  was 1 or 0, respectively. Then, we can interpret  $((V_i \leftrightarrow F_{V_i}) \vee (\neg)V_i)$  as “ $V_i$  either follows the semantics of  $M$  or takes on its original value represented as a positive or negative literal.” By doing so for all endogenous variables, we allow for all possible variations of  $\vec{W}$ . It is worth noting that we exclude those variables that are in  $\vec{X}$  from obtaining their original value, as we are already changed their setting to  $\neg\vec{x}$  and thus keeping a potential cause at its original value is not reasonable. It might not always make sense to add the original value for all variables. We leave this as an optimization that we present in Section 3.4.1.

### AC2 Algorithm

We formalize the above in Algorithm 2. Similar to Algorithm 1, the evaluation, in the input, is a list of all the variables in  $M$  and their values under  $\vec{u}$ . The rest is self-explanatory. We slightly change the definition of  $\varphi$  from a combination of primitive events to a combination of literals. For instance, instead of writing  $\varphi = (X_1 = 1 \wedge X_2 = 0 \vee X_3 = 1)$ , we use  $\varphi = (X_1 \wedge \neg X_2 \vee X_3)$ . In other words, we replace each primitive  $(X = x) \in \varphi$  with  $X$  if  $x = 1$  or  $\neg X$  if  $x = 0$  in the original assignment, such that we use  $\varphi$  in a formula. The same logic is achieved using the function  $f(Y = y)$  in Line 5 of the algorithm.

Before we construct formula  $F$ , we check if  $\vec{X} = \vec{x}$  given  $\vec{W} = \emptyset$  (Line 2) fulfills AC2. Hence, in this case, we do not need to look for a  $\vec{W}$ . Otherwise, we construct  $F$  (Line 4) that is a conjunction of  $\neg\varphi$  and the exogenous variables of  $M$  as literals depending on  $\vec{u}$ . Note that  $\varphi$  does not necessarily consist of a single variable only; it can be any Boolean

---

**Algorithm 2** Check whether AC2 holds using SAT

---

**Input:** causal model  $M$ , context  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ , effect  $\varphi$ , candidate cause  $\langle X_1, \dots, X_\ell \rangle = \langle x_1, \dots, x_\ell \rangle$ , evaluation  $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$

- 1: **function** FULFILLSAC2( $M, \vec{U} = \vec{u}, \varphi, \vec{X} = \vec{x}, \vec{V} = \vec{v}$ )
- 2:   **if**  $(M, \vec{u}) \models [\vec{X} \leftarrow \neg \vec{x}] \neg \varphi$  **then return**  $\emptyset$
- 3:   **else**
- 4:     
$$F := \neg \varphi \wedge \bigwedge_{i=1 \dots n} f(U_i = u_i) \wedge \bigwedge_{i=1 \dots m, \exists j \bullet X_j = V_i} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i))$$

$$\hookrightarrow \wedge \bigwedge_{i=1 \dots \ell} f(X_i = \neg x_i)$$
- 5:     **where**  $f(Y = y) = \begin{cases} Y, & y = 1 \\ \neg Y, & y = 0 \end{cases}$
- 6:     **if**  $\langle U_1 = u_1 \dots U_n = u_n, V_1 = v'_1 \dots V_m = v'_m \rangle \in \text{SAT}(\text{CNF}(F))$  **then**
- 7:        $\vec{W} := \langle W_1, \dots, W_s \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \Rightarrow W_i \neq W_j) \wedge (W_i = V_j \Leftrightarrow v'_j = v_j)$
- 8:       **return**  $\vec{W}$
- 9:     **else return** *not satisfiable*
- 10:    **end if**
- 11: **end if**
- 12: **end function**

---

formula. For example, if  $\varphi = (BS = 1 \wedge BH = 0)$  in the notation as defined by [74], we would represent it in  $F$  as  $(BS \wedge \neg BH)$ . This consideration is handled by Algorithm 2 without further modification. In addition, we represent each endogenous variable,  $V_i \notin \vec{X}$  with a disjunction between its equivalence formula  $V_i \leftrightarrow F_{V_i}$  and its literal representation. To conclude the formula construction, we add the negation of the candidate cause  $\vec{X} = \vec{x}$ , a consequence of Lemma 3.2. If  $F$ , represented in a conjunctive normal form, is satisfiable, we obtain the satisfying assignment (Line 6) and compute  $\vec{W}$  (Line 7) as the set of those variables whose valuations were *not* changed in order to ensure  $\neg \varphi$  that is finally returned. The unsatisfiability of  $F$  entails that AC2 does not hold. The soundness of the encoding is stressed in Theorem 3.3.

**Theorem 3.3.** *Formula  $F$  constructed within Algorithm 2 is satisfiable iff AC2 holds for a given  $M, \vec{u}$ , a candidate cause  $\vec{X}$ , and a combination of events  $\varphi$ .*

### Minimality of $\vec{W}$

In Algorithm 2, we considered  $\vec{W}$  to consist of all the variables whose original evaluation and satisfying assignments are equal. This is an over-approximation of the  $\vec{W}$  set because, possibly, there are variables that are not affected by changing the values of the cause, and are yet not required to be fixed in  $\vec{W}$ . Despite this consideration, a non-minimal  $\vec{W}$  is still valid, according to HP. Since notions such as the degree of responsibility [33] are quantified

depending on the size of  $\vec{W}$ , it is intuitive that a minimal  $\vec{W}$  is required for some situations. Therefore, we briefly discuss two modifications (in Algorithm 2) that yield a minimal  $\vec{W}$ .

We need to modify two parts of Algorithm 2. First, we cannot just consider *one* satisfying assignment for  $F$ . Rather, we need to analyze *all* the assignments. Determining all the assignments is an All-SAT problem. Second, we have to analyze each assignment of  $\vec{W}$  further to check if we can find a subset such that  $F$ , and thus AC2, still holds. Specifically, we check if each element in  $\vec{W}$  is equal to its original value because it was explicitly set so, or because it simply evaluated according to its equation. In the latter case, it is *not* a required part of  $\vec{W}$ . Precisely, in Algorithm 2, everything stays the same until the computation of  $F$ . After that, we check whether  $F$  is satisfiable, but now we compute all the assignments. Subsequently, for each assignment, we compute  $\vec{W}_i$ , as explained. Then, we return the smallest  $\vec{W}_i$  at the cost of iterating over *all* satisfying assignments.

### 3.3.2 Checking AC3

Our approach for checking AC3 using SAT is similar to the one for AC2. We construct another formula,  $G$ . The difference between  $G$  and  $F$  is in how the parts of the cause are represented. In  $G$ , we allow each of them to take on its original value *or* its negation (e.g.,  $A \vee \neg A$ ). Clearly, we could replace that disjunction with *true*. However, we explicitly do not perform this simplification such that a satisfying assignment for  $G$ , as returned by the SAT solver, still contains all variables of  $M$ .

In general, the idea is as follows. If we find a satisfying assignment for  $G$  such that at least one conjunct of the cause  $\vec{X} = \vec{x}$  takes on a value that equals the one computed from its corresponding equation, then, we know that this particular conjunct is not required to be part of the cause and there exists a subset of  $\vec{X}$  that fulfills AC2 as well. The same applies if the conjunct is equal to its original value in the satisfying assignment; this would mean that it is part of a  $\vec{W}$  such that  $\neg\varphi$  holds. When collecting all those conjuncts, we can construct a new cause  $\vec{X}' = \vec{x}'$  by subtracting them from the original cause and then checking whether or not it fulfills AC1. If it does, AC3 is violated because we identified a subset  $\vec{X}'$  of  $\vec{X}$  for which both AC1 and AC2 hold.

#### AC3 Algorithm

We formalize our approach in Algorithm 3. The input and the function  $f(V_i = v_i)$  remain the same as for Algorithm 2; the latter is omitted. In case  $\vec{X} = \vec{x}$  is a singleton cause or  $\varphi$  did not occur, AC3 is then fulfilled automatically (Line 2). Otherwise, Line 3 shows how formula  $G$  is constructed. This construction is only different from the construction of  $F$  in Algorithm 2 in how to treat variables  $\in \vec{X}$ . They are added as a disjunction of their positive and negative literals. Once  $G$  is constructed, we check its satisfiability; if it is not satisfiable, we return *true*, i.e., AC3 is fulfilled. For example, this can be the case if the candidate cause  $\vec{X}$  did not satisfy AC2. Otherwise, we check *all* its satisfying assignments. We need to

do this, as  $G$  might also be satisfiable for the original  $\vec{X} = \vec{x}$  so that we cannot say for sure that any satisfying assignment found, proves that there exists a subset of the cause. Instead, we need to obtain all of them. Obviously, this is problematic and could decrease the performance if  $G$  is satisfiable for a large number of assignments. Therefore, we present, in Section 3.3.4, an optimization of the check that eliminates the dependency on ALL-SAT. However, for now, we compute one assignment and check the *count* of the conjuncts in the cause that have different values in  $\vec{v}'$  than their original, and that their formula does not evaluate to this assignment (Line 5). If the *count* is less than the size of the cause, then AC3 is violated. Otherwise, we check another assignment. Theorem 3.4 proves the soundness of Algorithm 3; the proof is in Section 3.6).

---

**Algorithm 3** Check whether AC3 holds using ALL-SAT

---

**Input:** causal model  $M$ , context  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ , effect  $\varphi$ , candidate cause  $\langle X_1, \dots, X_\ell \rangle = \langle x_1, \dots, x_\ell \rangle$ , evaluation  $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$

- 1: **function** FULFILLSAC3( $M, \vec{U} = \vec{u}, \varphi, \vec{X} = \vec{x}, \vec{V} = \vec{v}$ )
- 2:   **if**  $\ell > 1 \wedge (M, \vec{u}) \models \varphi$  **then**
- 3:      $G := \neg\varphi \wedge \bigwedge_{i=1..n} f(U_i = u_i) \wedge \bigwedge_{i=1..m, \exists j \bullet X_j = V_i} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i))$   
 $\quad \leftrightarrow \bigwedge_{i=1..l} X_i \vee \neg X_i$
- 4:     **for all**  $\langle \vec{U} = \vec{u}, \vec{V} = \vec{v}' \rangle \in \text{SAT}(\text{CNF}(G))$  **do**
- 5:       **if**  $|\{j \in \{1, \dots, \ell\} \mid \exists i \bullet V_i = X_j \wedge v'_i \neq v_i\}|$   
 $\quad \leftrightarrow \wedge v'_i \neq [ \vec{V} \mapsto \vec{v}' ] F_{X_j} \} < \ell$  **then return false**
- 6:       **end if**
- 7:     **end for**
- 8:   **end if**
- 9:   **return true**
- 10: **end function**

---

**Theorem 3.4.** *Algorithm 3 returns false iff  $\vec{X}$  is a non-minimal cause.*

### Combining AC2 and AC3

While developing Algorithm 2 and Algorithm 3, we discovered that combining both is an option for optimizing our approach. In particular, we can exploit the relationship between the satisfying assignment(s) for the formulas  $F$  and  $G$ , i.e.,  $\vec{a}_F \in A_G$ . This holds as we allow the variables  $\vec{X}$  of a cause to be both 1 or 0 in  $G$  so that we can show that the satisfying assignment,  $\vec{a}_F$  for  $F$  in Algorithm 2 is an element of the satisfying assignments  $A_G$ , for  $G$ . Then, instead of computing both  $F$  and  $G$ , we could compute  $G$ , then filter those satisfying assignments that  $F$  would have yielded and use them for checking AC2 while we use all satisfying assignments of  $G$  to check AC3.

### 3.3.3 Example

Recall the *rock-throwing* example from Section 2.3. We want to find out whether Suzy throws,  $ST = 1$ , is a cause of the bottle shattering,  $BS = 1$ ? Assuming the context  $\vec{u}$  sets  $ST = 1$  and  $BT = 1$ , the original evaluation of the model is shown in the first row of Table 3.2. Algorithm 2 generates  $F$  shown in Equation 3.1, that is satisfiable for one assignment (Table 3.2 second row):  $BS = 0, SH = 0, BH = 0, ST = 0, BT = 1$ . All the variables, except  $BH$  and  $BT$ , change their evaluation. Thus, we conclude that  $ST = 1$  fulfills AC2 with  $\vec{W} = \{BH, BT\}$ . Notice that even though this  $\vec{W}$  is not minimal, it is still valid. That said, we still can calculate a minimal  $\vec{W}$  with more processing as described in Section 3.3.1.

Table 3.2: Truth Assignments of Formulae  $F$  and  $G$

	$BS$	$SH$	$BH$	$ST$	$BT$
$M$	1	1	0	1	1
$F$	0	0	0	0	1
$G_{\vec{a}_1}$	0	0	0	0	0
$G_{\vec{a}_2}$	0	0	0	0	1

$$\begin{aligned}
 F := & \overbrace{\neg BS}^{\neg\varphi} \wedge \overbrace{ST_{exo} \wedge BT_{exo}}^{\vec{u}} \wedge \overbrace{((BS \leftrightarrow SH \vee BH) \vee BS)}^{\text{equation of BS}} \wedge \overbrace{BS}^{\text{orig. BS}} \wedge \overbrace{((SH \leftrightarrow ST) \vee SH)}^{\text{equation of SH orig. SH}} \\
 & \wedge \overbrace{((BH \leftrightarrow BT \wedge \neg SH) \vee \neg BH)}^{\text{equation of BH}} \wedge \overbrace{\neg BH}^{\text{orig. BH}} \wedge \overbrace{\neg ST}^{\text{equation of ST}} \wedge \overbrace{((BT \leftrightarrow BT_{exo}) \vee BT)}^{\text{equation of BT}} \wedge \overbrace{BT}^{\text{orig. BT}}
 \end{aligned} \tag{3.1}$$

To illustrate checking AC3, we ask a different question: are  $ST = 1 \wedge BT = 1$  a cause of  $BS = 1$ ? Note that AC2 is fulfilled with  $W = \emptyset$ , for this cause. Obviously, if both do not throw, the bottle does not shatter. Using Algorithm 3, we obtain formula  $G$  shown in Equation 3.2.

$$\begin{aligned}
 G := & \overbrace{\neg BS}^{\neg\varphi} \wedge \overbrace{ST_{exo} \wedge BT_{exo}}^{\vec{u}} \wedge \overbrace{((BS \leftrightarrow SH \vee BH) \vee BS)}^{\text{equation of BS}} \wedge \overbrace{BS}^{\text{orig. BS}} \wedge \overbrace{((SH \leftrightarrow ST) \vee SH)}^{\text{equation of SH orig. SH}} \\
 & \wedge \overbrace{((BH \leftrightarrow BT \wedge \neg SH) \vee \neg BH)}^{\text{equation of BH}} \wedge \overbrace{\neg BH}^{\text{orig. BH}} \wedge \overbrace{ST}^{\text{orig. ST}} \vee \overbrace{\neg ST}^{\text{negated orig. ST}} \wedge \overbrace{BT}^{\text{orig. BT}} \vee \overbrace{\neg BT}^{\text{negated orig. BT}}
 \end{aligned} \tag{3.2}$$

As Table 3.2 shows,  $G$  is satisfiable with *two* assignments  $G_{\vec{a}_1}$  and  $G_{\vec{a}_2}$ . For  $\vec{a}_1$ , we can see that both  $ST$  and  $BT$  have values different from their original evaluation and that both do not evaluate according to their equations. Thus, we cannot show that AC3 is violated, yet. For  $\vec{a}_2$ ,  $BT = 1$ , so it is equal to the evaluation of its equation. Consequently,  $BT$  is not a required part of  $\vec{X}$ , because  $\neg\varphi = \neg BS$  still holds although we did not set  $BT = 0$ . So, AC3 is not fulfilled because AC1 and AC2 hold for a subset of the cause.

### 3.3.4 Optimized AC3 Check with SAT

Informally,  $G$  encodes the set of satisfying assignments for  $\neg\varphi$  with the removal of the equations for  $\vec{X}$ . Thus, it can be used to check AC2 (we have all combinations of  $\vec{X}$ ), and to check AC3 by analyzing *all* of them to find irrelevant causes. To solve ALL-SAT problems, modern solvers typically add clauses (of size equal to the number of variables) called blocking clauses, to block already-found solutions [69] and let the solver find a new solution. This method has benefits, especially for formulas that have a small number of solutions [190]. However, the two downsides are the potential memory saturation and the slow-down of the solver [204, 190, 69]. Therefore, the approach in Section 3.3.2 then fails to scale for larger models, especially when the cardinality of the cause is big. We present an optimization that eliminates this need. The *optimized* approach extends  $G$  with new clauses that encode the notions of *non-minimality* and *non-empty causes*.

We express *non-minimality* because it is easier to explain than minimality in this context. Non-minimality of a cause means that there exists one variable  $X_i = x_i$  in a satisfying assignment of  $G$  (which includes  $\neg\varphi$ , i.e., effect not happening) such that the value of  $X_i$  evaluates according to its equation, *or* equals its actual value (i.e., the value for which  $\varphi$  evaluates to true) represented as a negative or positive literal. In other words, it is not necessary to negate the value of  $X_i$  to have  $\neg\varphi$  occurring. We express this notion of non-minimality by the sub-formula  $H$  shown in Equation 3.3.

$$H := \bigvee ((X_i \leftrightarrow F_{X_i}) \vee f(X_i = x_i)) \quad (3.3)$$

The disjunction in  $H$  holds if at least one cause variable violates the minimality, as explained. On the other hand, a *non-empty* cause means that at least one cause variable  $X_j$  is not determined by its equation or takes on its original value (which led to  $\varphi$  holding), and is negated due to an intervention. We express this notion using the sub-formulas  $K$ , shown in Equation 3.4.

$$K := \neg(\bigwedge f(X_i = x_i)) \wedge \neg(\bigwedge X_i \leftrightarrow F_{X_i}) \quad (3.4)$$

With the first part of  $K$ , we make sure that not all the variables are equal to their original value (which led to  $\varphi$  holding). Similarly, for the second part: Not every variable is allowed to evaluate according to its equation.

The idea now is to extend  $G$  to  $G'$  by adding the notions of *non-minimality* ( $H$ ), and *non-empty* cause ( $K$ ), i.e.,  $G \wedge H \wedge K$ . This way, we ensure that  $G'$  is only satisfiable if there exists a smaller, and non-empty subset of the cause  $\vec{X} = \vec{x}$  that would satisfy AC2. Otherwise, we know that there exists no strict subset of  $\vec{X} = \vec{x}$  for which AC2 holds. Relating to AC3, the new clauses make the *unsatisfiability* of  $G'$  an indication that AC3 holds. This reduces the analysis effort, and eliminate the need to compute all assignments, but introduce additional overhead and clauses when constructing  $G'$ .



## Encoding

For the optimized AC3 check we show an encoding rather than an algorithm because, unlike the previous approaches, no analysis of the satisfying assignment is required. In Equation 3.5 the encoding of  $G'$  is shown formally. It starts with the negation of the effect  $\varphi$ ; then it enumerates the context (values of exogenous variables), i.e.,  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ . Each variable in  $\vec{U}$  is represented with  $U$  if  $u = 1$  or  $\neg U$  if  $u = 0$  in the context. This logic is achieved using the function  $f(Y = y)$ . In the next line, we use the equivalence operator as explained in Section 3.3.1 to express each variable ( $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$ ) not in the candidate cause. The remaining of the encoding represents the variables of the candidate cause  $\langle X_1, \dots, X_\ell \rangle = \langle x_1, \dots, x_\ell \rangle$ . The last two lines are the newly added parts in  $G'$ , which correspond to sub-formulas  $H$ , and  $K$  explained above.

$$\begin{aligned}
 G' := & \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i) \\
 & \wedge \bigwedge_{i=1\dots m, \exists j \bullet (X_j = V_i)} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i)) \\
 & \wedge \neg \bigwedge_{i=1\dots\ell} (\neg(X_i \leftrightarrow F_{X_i}) \wedge \neg(f(X_i = x_i))) \\
 & \wedge \neg \bigwedge_{i=1\dots\ell} (f(X_i = x_i)) \wedge \neg \bigwedge_{i=1\dots\ell} (X_i \leftrightarrow F_{X_i})
 \end{aligned} \tag{3.5}$$

To check if  $\vec{X}$  is a cause of  $\varphi$ , we need to check  $F$  and  $G'$  separately or combine them in one formula ( $F \wedge \neg G'$ ). The first option allows us to examine each condition (AC2, AC3) in isolation, while the second would report the overall result. Depending on the specific situation, the two options can be used. The time for constructing the formulas in the two options is similar; however, the CNF conversion and solving time may differ. In our evaluation in Section 3.5, we used the first option because it is comparable to the other approaches in being able to distinguish the violation of the two conditions. To discuss the soundness of our encoding, we present Theorem 3.5 (the proof is presented in Section 3.6).

**Theorem 3.5.** *Formula  $G'$  constructed with Equation 3.5 is satisfiable iff AC3 is violated.*

## Example

Let us consider our approach to check AC3 with the *rock-throwing* example. Remember that we want to check whether  $ST = 1 \wedge BT = 1$  is a cause for  $BS = 1$  under  $ST_{exo}, BT_{exo} = 1$ . Using our encoding (Equation 3.5), we obtain  $G'$  shown in Equation 3.6 for the current

example (extended parts highlighted).

$$\begin{aligned}
 G' = & \overbrace{\neg BS}^{\neg\varphi} \wedge \overbrace{ST_{exo} \wedge BT_{exo}}^{\bar{u}} \wedge \overbrace{(BS \leftrightarrow SH \vee BH)}^{\text{equation of } BS} \wedge \overbrace{((SH \leftrightarrow ST) \vee SH)}^{\text{equation of } SH \text{ orig. } SH} \wedge \\
 & \overbrace{((BH \leftrightarrow BT \wedge \neg SH) \vee \neg BH)}^{\text{equation of } BH \text{ orig. } BH} \wedge \\
 & \neg(\underbrace{\neg(ST \leftrightarrow ST_{exo})}_{\text{negated equ. of } ST} \wedge \underbrace{\neg ST}_{\text{negated orig. } ST}) \wedge \neg(\underbrace{\neg(BT \leftrightarrow BT_{exo})}_{\text{negated equ. of } BT} \wedge \underbrace{\neg BT}_{\text{negated orig. } BT}) \wedge \\
 & \neg(\underbrace{ST}_{\text{orig. } ST} \wedge \underbrace{BT}_{\text{orig. } BT}) \wedge \neg(\underbrace{(ST \leftrightarrow ST_{exo})}_{\text{equation of } ST} \wedge \underbrace{(BT \leftrightarrow BT_{exo})}_{\text{equation of } BT})
 \end{aligned} \tag{3.6}$$

$G'$  is *satisfiable* for  $ST = 0, BT = 1, SH = 0, BH = 0, BS = 0$ , which is exactly what we wanted to show:  $BS = 0$  even if we only set  $ST = 0$  while  $BT = 1$ . Consequently, AC3 does not hold for cause  $ST = 1 \wedge BT = 1$ . Note that we can use this result only to check minimality, and not to determine a minimal cause subset because one assignment is not sufficient to conclude a minimal subset.

### 3.4 Graph-Based Optimizations

As mentioned in Chapter 2, causal models are visualized in causal networks. When considering a causal model as a graph, we can exploit specific graph features (in addition to our knowledge in HP) to reduce the space of computation. In our SAT approaches, such reduction may result in smaller encodings of formulae  $F$  and  $G$ , which does not necessarily result in faster solving time [46]. We present two optimizations that utilize the reachability of the variables based on our knowledge of HP to eliminate constraints.

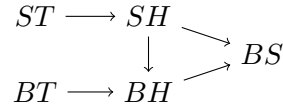
#### 3.4.1 Reduce Number of Potential Variables for $\vec{W}$

Modern SAT solvers implement different pre-processing techniques to reduce the size of the SAT formula [46, 85]. These techniques typically aim to eliminate redundant, subsumed, or tautological clauses that can be determined in polynomial time [85]. According to our experiments with the solver we used (details in Section 3.5), the elimination that we present does not fall under these categories; hence, it is not done by the solver. This is justified by the fact that this optimization stems from our knowledge in HP and its encoding to SAT, which enables us to know that one constraint can be excluded for some variables.

The contingency set  $\vec{W}$  contains the variables that need to be fixed to their original value so that the effect does not occur. For a specific causal query, there may exist some variables that can be excluded when considering set  $\vec{W}$  as part of AC2. Specifically, if an endogenous variable  $V_i$  is not affected by a cause variable, i.e., on the directed acyclic graph variable,  $V_i$  cannot be *reached* from any variable in  $\vec{X}$ . For instance, in Figure 3.1,  $ST$  does not affect the

value  $BT$  in the rock-throwing example. Hence,  $BT$  will not change its original value no matter how we change  $ST$ . However,  $BH$ , for example, is affected by  $ST$  as it depends on  $SH$ , which itself depends on  $ST$ .

The second class of variables that can be excluded when constructing  $\vec{W}$  are those that do not affect  $\varphi$ . That is, there exists no directed path from the variable to any effect variable. Thus, keeping such a variable at its original value does not change  $\varphi$ .



Each node  $A$  represents an endogenous variable; an edge from  $A$  to  $B$  means that  $B$  "depends" on  $A$ .

Figure 3.1: Rock-Throwing Example (Source: [79])

Identifying the variables in the cases, as mentioned earlier, allows us to exclude them when constructing our formulae. We construct set  $T$ , which would include the endogenous variables, excluding the cause variables, that could be in the contingency set ( $\vec{W}$ ). Formally:

$$T = \{V \mid V \in \mathcal{V}, V \notin \vec{X}, V \text{ is reached from } X_i \in \vec{X}, V \text{ affects } \varphi\}$$

Then, we use  $T$  to only add the original value of variables in formulae  $F$  and  $G$ . For instance, the encoding of  $F$  is adapted, as shown in Equation 3.7. We can then use this adaption with all our algorithms and encodings. This will reduce the size of the formula in some situations.

$$\begin{aligned} F := & \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i) \wedge \bigwedge_{i=1\dots m, \bullet V_i \in T} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i)) \\ & \wedge \bigwedge_{i=1\dots m, \exists j \bullet X_j = V_i \wedge T_j = V_i} (V_i \leftrightarrow F_{V_i}) \wedge \bigwedge_{i=1\dots \ell} f(X_i = \neg x_i) \end{aligned} \quad (3.7)$$

### 3.4.2 Remove Irrelevant Sub-formulae

A similar potential optimization is to not only remove the original value of some variables in formulas  $F$  and  $G$  but to remove the complete corresponding sub-formula if it is irrelevant for the current scenario. This is precisely the case with the variables in the second class of the previous optimization. That is, if  $\varphi$  is not affected (reachable) by some endogenous variables, e.g., because we picked an intermediate variable to be the effect. Set  $K$  contains the variables that, besides the variables in the cause, actually affect  $\varphi$ . Formally:

$$K = \{V \mid V \in \mathcal{V}, V \notin \vec{X}, V \text{ affects } \varphi\}$$

The reachability between the effect and a variable  $V_i$  determines if  $V_i$  affects  $\varphi$ , and hence its corresponding sub-formula is added to the  $F$ . As such, formula  $F$  is adapted in Equation 3.8.

$$\begin{aligned}
 F := & \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i) \wedge \bigwedge_{i=1\dots m, \bullet V_i \in K} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i)) \\
 & \wedge \bigwedge_{i=1\dots \ell} f(X_i = \neg x_i)
 \end{aligned} \tag{3.8}$$

We think that this optimization may reduce the size of formulae  $F$  and  $G$  and thus make the corresponding algorithms more efficient in certain situations. As we will see in our experiments shown in Section 3.5, at least the SAT solver we are using does benefit from this optimization in certain situations and therefore does not seem to perform it.

## 3.5 Evaluation

In this section, we provide details on the implementation of our algorithms and evaluate their efficiency in answering causal queries.

### 3.5.1 Technical Implementation

Our implementation is a Java library; it can easily be integrated into other systems. It supports both the creation of binary causal models as well as solving causality problems. For the modeling part and the implementation of our SAT-based approach, we take advantage of the library *LogicNG*.<sup>3</sup> It provides methods for creating and modifying boolean formulas and allows to analyze those using different SAT solvers. We use the implementation of *MiniSAT* solver [47] within *LogicNG*. For the sake of this evaluation, we will compare the *execution time* and *memory allocation* for the following *seven* strategies:

- BRUTE\_FORCE - a standard brute-force implementation of HP that enumerates the power-set of the variables.
- SAT - the basic approach that implements (Algorithm 2 and Algorithm 3).
- SAT<sup>OPT</sup> - the optimized check of AC3 (Algorithm 2 and Equation 3.5).
- SAT<sup>MIN</sup> - the minimal  $\vec{W}$  extension.
- SAT<sup>COM</sup> - optimization of the SAT by combining AC2 and AC3.
- SAT<sup>GR1</sup> - graph-based optimization of the SAT to reduce  $\vec{W}$ .
- SAT<sup>GR2</sup> - second graph-based optimization to remove irrelevant formulae.

---

<sup>3</sup><https://github.com/logic-ng/LogicNG>

All our measurements were performed on Ubuntu 16.04 LTS machine equipped with an Intel® Core™ i7-3740QM CPU and 16 GB RAM. For each benchmark, we specified 10 warmup iterations and 30 measurement iterations. Warm-ups are iterations in which we dry-run our benchmarks before results are collected. This is a recommended practice in Java benchmarking to avoid accounting for factors like Java Virtual Machine (JVM) warm-up threads, compiler optimizations, garbage collection, Just In Time (JIT) compilation, class loading, and initialization, and environmental noise. On the other hand, the measurement iterations are run after the warm-up iterations, and their results are collected.

### 3.5.2 Methodology and Evaluated Models

To the best of our knowledge, no previous work has published their causal models. Thus, we gathered a dataset of 37 models, which included 21 *small* models ( $\leq 400$  endogenous variables)—from domains of causality, security, safety, and accident investigation— and 16 larger security models from an industrial partner, in addition to artificially generated models. The smaller models contained 9 illustrative examples from literature (number of endogenous variables in brackets) such as *Throwing – Rocks*(5), *Railroad*(4) [74], 2 variants of a safety model that describes a leakage in a subsea production system *LSP*(41) and *LSP2*(41) [30], and an aircraft accident model (Ueberlingen, 2002) *Ueb*(95) [187], 7 generated binary trees, and a security model obtained from an industrial partner which depicts how insiders within a company steal a master encryption key *SMK*. Because it can be parameterized by the number of employees in a company, we have 14 variants of *SMK*, 2 small ones *SMK1*(36) and *SMK8*(91), and 12 large models of sizes (550 – 7150). In addition, we artificially generated 4 models: 2 binary trees with different heights, denoted as *BT*(2047 – 4095), and 2 trees combined with non-tree random models, denoted as *ABT*(4103), and *ABT2*(8207). We have evidence that such large models are likely to occur when built automatically from architectures or inferred from other sources [96, 95]. The description, including the semantics, variables, causal network, and the causal queries, of each model, is given in Appendix A.<sup>4</sup>

We formulated a total of 484 *checking* queries that vary in the context, cause, effect, and consequently differ in the result of AC1-AC3, the size of  $\vec{W}$ , and the size of the minimal cause. For the smaller models, we specified the queries manually according to their sources in literature and verified that our results match the sources. For the larger models, we constructed a total of 224 checking queries. We specified some effects (e.g., root of *BT*, or *steal passphrase* in *SMK*) and used different contexts, and randomly selected causes (sizes 1, 2, 3, 4, 10, 15, and 50) from the models.

### 3.5.3 Discussion and Results

We present our results by discussing the general trends that we observed in our experiments. We use Table 3.3 to show the details of representative cases of these trends, and then we

<sup>4</sup>the machine-readable models are available at <https://git.io/Jf8iH>

### 3 Efficiently Checking Actual Causality with SAT Solving

Model	$ \vec{V} $	ID	$ \vec{X} $	Result					Execution Time (s) Memory consumption (GB)								
				AC1	AC2	AC3	$ \vec{W} $	$ \vec{X}_{min} $	Brute_Force	SAT	SAT <sup>OPT</sup>	SAT <sup>MIN</sup>	SAT <sup>COM</sup>	SAT <sup>GR1</sup>	SAT <sup>GR2</sup>		
Forest Fire	3	3	2	Y	Y	Y	0	2									
Assassin	3	9	2	Y	Y	Y	0	2									
Prisoners	4	4	3	Y	Y	N	0	2									
Rail Road	4	3	1	Y	Y	Y	0	1									
Rock_throwing	5	12	3	Y	Y	N	0	1	$6 \times 10^{-5}$ $1.9 \times 10^{-4}$	$7 \times 10^{-5}$ $1 \times 10^{-4}$	$1 \times 10^{-4}$ $2 \times 10^{-4}$	$7.1 \times 10^{-5}$ $1.3 \times 10^{-4}$	$7.2 \times 10^{-5}$ $1.3 \times 10^{-4}$	$8 \times 10^{-5}$ $1.4 \times 10^{-4}$	$7.2 \times 10^{-5}$ $1.35 \times 10^{-4}$		
SMK_3	36	3	3	Y	Y	Y	4	3	N/A N/A	0.00089 0.0011	0.00093 0.0011	0.00108 0.0012	0.00073 0.0009	0.00095 0.0011	0.00092 0.0011		
		24	3	Y	Y	Y	0	3	N/A N/A	0.00071 0.0009	0.00073 0.0009	0.00072 0.0009	0.00079 0.0009	0.0007 0.00086	0.00073 0.0009		
		29	2	Y	Y	N	0	1	0.000136 0.0002268	0.00072 0.0009	0.00069 0.0009	0.00072 0.00093	0.00072 0.0009	0.0007 0.0008	0.00039 0.0005		
LSP	41	3	2	Y	Y	N	0	1	0.00016 $2.524 \times 10^{-10}$	0.0012 0.0014	0.00109 0.0013	0.0023 0.0024	0.0009 0.0011	0.0012 0.0014	0.0012 0.0014		
SMK	91	11	3	Y	Y	Y	0	3	N/A N/A	0.012 0.008	0.00219 0.0027	0.012 0.008	0.011 0.008	0.00215 0.0024	0.00067 0.00069		
Ueberlingen	95	5	4	Y	Y	N	88	3	N/A N/A	0.805 0.0018	0.502 0.0032	0.450 0.0032	0.507 0.0032	0.520 0.0032	0.545 0.0032		
BT_11	4095	35	4	Y	N	Y	N/A	4	N/A N/A	6.949 2.0	6.717 2.0	6.618 2.0	2.94 1.1	7.2 2.0	6.95 2.0		
ABT	4103	1	1	Y	N	N		1	N/A N/A	4.285 1.1	5.46 1.1	4.507 1.055	5.718 1.055	6.24 1.055	6.03 1.06		
		2	2	Y	Y	Y	4086	2	N/A N/A	8.0 2.04	10.5 1.7	23.76 4.76	5.69 1.06	13.2 1.8	11.54 1.8		
		3	5	Y	Y	N	4090	2	N/A N/A	8.331 2.04	12.058 1.80	42.31 7.1982	6.172 1.07	13.712 1.8	11.69 1.8		
		4	10	Y	Y	N	4086	2	N/A N/A	14.37 4.2	12.02 1.80	207.987 13.88	13.312 2.23	19.8 2.97	19.84 2.97		
		5	11	N	Y	N	4086	2	N/A N/A	30.2 5.7	12 1.80	58.14 8.129	30.53 5.03	45.842 5.77	40.734 5.7		
		6	15	Y	Y	N	4086	2	N/A N/A	N/A N/A	11.55 1.8	N/A N/A	N/A N/A	N/A N/A	N/A N/A		
		7	15	N	Y	N	4080	5	N/A N/A	6308 13.88	11.03 1.80	N/A N/A	N/A N/A	N/A N/A	N/A N/A		
		8	15	N	Y	N	4080	5	N/A N/A	6791 13.88	12.04 1.80	N/A N/A	N/A N/A	N/A N/A	N/A N/A		
		9	50	Y	Y	N	4079	5	N/A N/A	N/A N/A	10.35 2.04	N/A N/A	N/A N/A	N/A N/A	N/A N/A		
		10	50	Y	Y	N	4068	11	N/A N/A	N/A N/A	10.37 2.04	N/A N/A	N/A N/A	N/A N/A	N/A N/A		
ABT2	8207	1	11	Y	Y	Y	8161	11	N/A N/A	24.94 4.07	21.04 4.0	25.39 4.077	27.35 4.077	28.26 4.077	25.67 4.077		
		2	22	Y	Y	N	8191	11	N/A N/A	0.93 0.028	0.90 0.028	0.106 0.000003	0.124 0.000001	0.094 0.000003	0.055 0.000003		

Table 3.3: Execution Time and Memory Allocation of the Scenarios

show cactus plots of the overall performance. In Table 3.3, the first three columns show the model *name*, its size, and the *ID* of the scenario that differs in the details of the causal query, i.e.,  $\vec{X}$  and  $\varphi$ . The cardinality of  $\vec{X}$  is shown in the fourth column. Next, the results of AC1-AC3 are displayed. The size of the minimal  $W$  set is displayed next. Finally, the execution time and memory allocation (in the second line of each cell) per strategy are shown. We write N/A if the computation was not completed in two hours or required too much memory. As a general remark, it does not matter which strategy is used if AC2 holds for an empty  $\vec{W}$  and  $\vec{X}$  is a singleton. These cases are explicitly checked in our implementation before proceeding to the solving. Also, all models from the causality literature are solved by any approach in milliseconds.

As expected, the Brute-Force approach (BF) works only for very small models (<five variables), or in situations where only a few iterations are required to check the query. The

former case is evident with the N/A entries in the `Brute-Force` column. Intuitively, this is a result of the potential exponential blow-up when searching for  $\vec{W}$ . For instance, for  $SMK_3$ , the set of all possible  $\vec{W}_i$  has a size of up to  $2^{35}$ . In the worst case, this number of iterations is required for finding out that AC2 does not hold. It is possible that this number of iterations multiplied by the number of subsets of the cause needs to be executed again to check AC3. This causes BF to be extremely slow. The latter case is demonstrated in a few exceptions, such as  $LSP-3$  (query ID 3 using model  $LSP$ ), and  $SMK_3-29$ , where we see BF reported some numbers. Accurately, we see these situations when AC2 holds with a small or empty  $\vec{W}$ , and AC3 does not hold. That is, the number of iterations BF performs is small because the sets,  $\vec{W}_i$  are ordered by size. Such examples did not exhibit the significant problem of BF, i.e., the generation of all possible sets,  $\vec{W}_i$ , whose number increases exponentially.

The SAT, by contrast, always stays below  $1.1ms$  and allocates less than  $1.5MB$  during the execution for all scenarios of the  $SMK_3$ . With some variation, SAT handled many queries on larger models efficiently. For instance,  $BT_{11-35}$ , where the underlying causal model contains 4095 variables, executed in less than  $7s$ . However, the latter scenario is special because AC2 does not hold. In  $ABT 1-5$ , regardless of AC2's result, SAT still can answer a query under  $30s$ . Although SAT requires an ALL-SAT procedure, for many of the models we used, this was not a problem.

That said, our results also showed that computing all satisfying assignments of  $G$  is inefficient when the number of satisfying assignments is large. Although insignificant, this can be seen even in the small models, e.g., scenario  $SMK - 11$  the execution time dropped ( $12ms$  to  $2.2ms$ ) because  $G$  here has 17 satisfying assignments. This case shows the potential slow-down of the SAT approach (even for small models) due to the processing steps of the assignments. More interestingly, the experiments with bigger models and bigger cardinalities of causes confirmed the two potential problems with ALL-SAT. The first is the memory exhaustion [204] of the solver as a result of accumulating the blocking clauses (see Section 3.1). We saw this behavior in scenarios  $ABT : 6, 9, 10$ , where the program ran out of memory ( $14GB$  assigned to the benchmark) before returning an answer (denoted by N/A in the table). These scenarios were checking a non-minimal cause of size 15 and 50 variables.  $SAT^{OPT}$ , on the other hand, finished executing the two scenarios in less than 12 seconds, with a memory utilization of around  $2GB$ .

The other problematic behavior with SAT is the significant potential slow-down of the solver due to the internal unit propagation performance drop as the formula inflates in size [190, 69]. This is clear in almost all the scenarios, especially with bigger causes. For instance, we saw SAT taking two to four times as much as the execution time of  $SAT^{OPT}$  in scenario  $ABT : 5$ . In more extreme cases, we tested non-minimal causes where AC1 does not hold, and SAT took around 2 hours to finish, whereas in these cases  $SAT^{OPT}$  took only 12 seconds.

While obtaining a minimal  $\vec{W}$  using  $SAT^{MIN}$  showed a rather small impact relative to the SAT approach in many scenarios, it showed a significant increase in many other cases. This impact is dependent on the nature and semantics of the underlying model. In all

the cases that  $SAT^{MIN}$  was feasible ( a solution is returned before timeout)  $SAT$  is feasible. However, we also observed cases where  $SAT^{MIN}$  was not feasible although  $SAT$  was feasible, e.g., *ABT-7*. In general, we can only observe a major impact if the number of satisfying assignments or the size of a non-minimal  $\vec{W}$  is large. For instance, a significant increase (up to 15 times) was observed in scenarios *ABT2-5* where the size of the non-minimal  $\vec{W}$  was around 4000. This approach is seen as the slowest approach among the benchmarked methods.

Combining the algorithms for AC2 and AC3 is only beneficial if AC2 and AC3 need to be explicitly analyzed (AC2 does not hold for an empty  $W$ , and the cause is not a singleton). We have many such scenarios in our examples. In evaluating them, we found out that there is a positive impact in using this optimization, but it is rather small on average. Larger differences can be seen, for instance, in *BT11-35* and *ABT:2-5* where the SAT-based approach was considerably slower than  $SAT^{COM}$ . However, both suffered from the All-SAT procedure with large cause cardinalities.

The two graph-based optimizations did not show any significant improvement over the SAT approach. In a small number of examples, they showed an enhancement over SAT. For instance, in *SMK-11*, both finished faster. However, in many other cases, they were both slower than the baseline approach. To analyze these results, we confirmed that the solver does not perform these optimizations itself; we came to this conclusion by comparing the number of clauses in the CNF form of the original SAT formula, which was larger than the number of clauses in the reduced formulae. Usually, these clauses are the result of the pre-processing that SAT solvers perform before solving. As such, the slowdown of our algorithms is partially due to the reduction of the formula that we perform. Further, removing the disjunction (as part of  $SAT^{GR1}$ ) may result in a slightly harder formula because the solver is forced to satisfy the equivalence part. However, that is a good optimization when considering the minimality of  $\vec{W}$ . Lastly, these optimizations are only relevant in specific situations. Namely,  $SAT^{GR1}$  would only be beneficial if the query contains a large candidate cause;  $SAT^{GR2}$ , on the other hand, would be helpful if the effect is not the root of the model.

As part of our analysis, we found that, for a large class of causal queries, actual causality can be computed efficiently with any of our SAT-based approaches. Methods that use brute-force mechanisms do not scale while accounting for AC2 and AC3. Our results showed that the limit of brute-force is around thirty variables. On the other hand, approaches that use SAT are especially efficient in checking AC2. As such, the baseline approach  $SAT$  is a good starting point that deals with models of 4000 variables in less than 7 seconds, using a memory of 2 GB. That said, it also suffers mainly with more complex queries with larger candidate causes. For that,  $SAT^{OPT}$  is very efficient; it handles AC3 without requiring the inefficient procedure of ALL-SAT. As a secondary result, accounting for a minimal  $\vec{W}$ , along with a minimal cause, can be achieved using  $SAT^{MIN}$ , with a significant slowdown in the solving time.

To conclude this evaluation, we show, in Figure 3.2 cactus plots of the results of 224



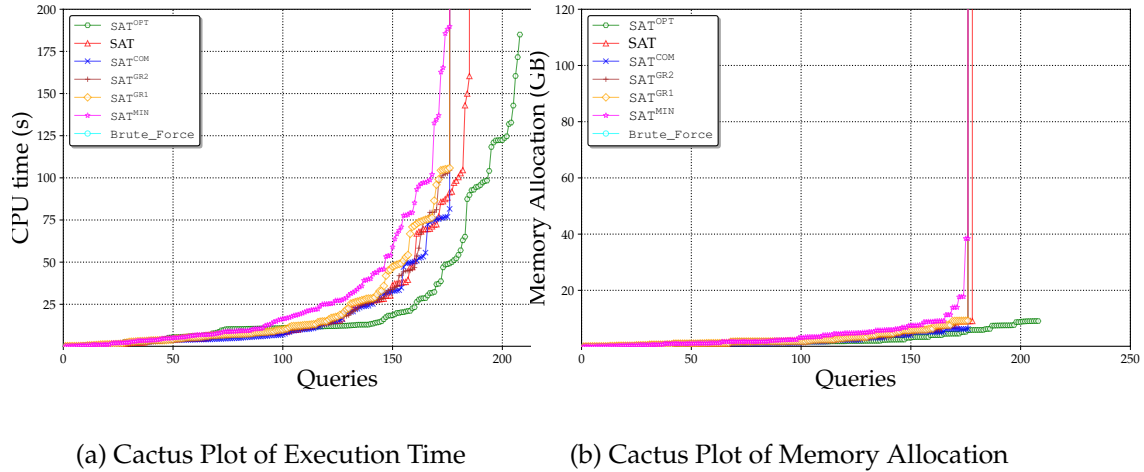


Figure 3.2: Cactus plots of Execution Time and Memory Results on the Larger Models. A point on the x-axis represents a query an approach answered ordered by the execution time, shown on the y-axis; a point  $(x, y)$  on the plot reads as  $x$  queries can be answered in  $y$  or less.

queries on larger models (models with more than 400 variables) to compare the different approaches. The x-axis shows the number of queries an approach answered ordered by the execution time, which is shown on the y-axis; a point  $(x, y)$  on the plot reads as  $x$  queries can be answered in  $y$  or less. Figure 3.2a shows the superiority of  $SAT^{OPT}$ ; in total it was able to answer 209 queries from the set of complex queries, while the other SAT-based approaches only managed to answer around 180 queries. We conclude that  $SAT^{OPT}$  outperforms the other approaches, and is necessary, especially for models larger than 1000 endogenous variables.

## 3.6 Proofs

### 3.6.1 Negation Lemma

**Lemma 3.2.** *In a binary model, if  $\vec{X} = \vec{x}$  is a cause of  $\varphi$ , according to Definition 2.4, then every  $\vec{x}'$  in the definition of AC2 always satisfies  $\forall i. x'_i = \neg x_i$ .*

*Proof.* We use the following notation:  $\vec{X}_{(n)}$  stands for a vector of length  $n$ ,  $X_1, \dots, X_n$ ; and  $\vec{X}_{(n)} = \vec{x}_{(n)}$  stands for  $X_1 = x_1, \dots, X_n = x_n$ . Let  $\vec{X}_{(n)} = \vec{x}_{(n)}$  be a cause for  $\varphi$  in some model  $M$ .

1. AC1 yields

$$(M, \vec{u}) \models (\vec{X}_{(n)} = \vec{x}_{(n)}) \wedge (M, \vec{u}) \models \varphi. \quad (3.9)$$

2. Assume that the lemma does not hold. Then there is some index  $k$  such that  $x'_k = x_k$  and AC2 holds. Because we are free to choose the ordering of the variables, let us set  $k = n$  wlog. We may then rewrite AC2 as follows:

$$\exists \vec{W}, \vec{w}, \vec{x}'_{(n)} \bullet (M, \vec{u}) \models (\vec{W} = \vec{w}) \implies (M, \vec{u}) \models \left[ \vec{X}_{(n-1)} \leftarrow \vec{x}'_{(n-1)}, X_n \leftarrow x_n, \vec{W} \leftarrow \vec{w} \right] \neg \varphi. \quad (3.10)$$

3. We will show that equations 3.9 and 3.10 give rise to a smaller cause, namely  $\vec{X}_{(n-1)} = \vec{x}_{(n-1)}$ , contradicting the minimality requirement AC3. We need to show that the smaller cause  $\vec{X}_{(n-1)} = \vec{x}_{(n-1)}$  satisfy AC1 and AC2, as stated by equations 3.11 and 3.12 below. This violates the minimality requirement of AC3 for  $\vec{X}_{(n)} = \vec{x}_{(n)}$ .

$$(M, \vec{u}) \models (\vec{X}_{(n-1)} = \vec{x}_{(n-1)}) \wedge (M, \vec{u}) \models \varphi \quad (3.11)$$

states AC1 for a candidate “smaller” cause  $\vec{X}_{(n-1)}$ . Similarly,

$$\begin{aligned} \exists \vec{W}^*, \vec{w}^*, \vec{x}'^*_{(n-1)} \bullet (M, \vec{u}) \models (\vec{W}^* = \vec{w}^*) \\ \implies (M, \vec{u}) \models \left[ \vec{X}_{(n-1)} \leftarrow \vec{x}'^*_{(n-1)}, \vec{W}^* \leftarrow \vec{w}^* \right] \neg \varphi \end{aligned} \quad (3.12)$$

formulates AC2 for this candidate smaller cause  $\vec{X}_{(n-1)}$ .

4. Let  $\Psi$  denote the structural equations that define  $M$ . Let  $\Psi'$  be  $\Psi$  without the equations that define the variables  $\vec{X}_{(n)}$  and  $\vec{W}$ ; and let  $\Psi''$  be  $\Psi$  without the equations that define the variables  $\vec{X}_{(n-1)}$  and  $\vec{W}$ . Clearly,  $\Psi'' \implies \Psi'$ .

We can turn equation 3.9 into a propositional formula, namely

$$E_1 := \left( \Psi \wedge \vec{X}_{(n-1)} = \vec{x}_{(n-1)} \wedge X_n = x_n \right) \wedge \varphi. \quad (3.13)$$

Similarly, equation 3.11 is reformulated as

$$E_2 := \left( \Psi \wedge \vec{X}_{(n-1)} = \vec{x}_{(n-1)} \right) \wedge \varphi. \quad (3.14)$$

Because equation 3.10 holds, we fix some  $\vec{W}, \vec{w}, \vec{x}'_{(n)}$  that make it true and rewrite this equation as

$$E_3 := \left( \Psi' \wedge \vec{X}_{(n-1)} = \vec{x}'_{(n-1)} \wedge X_n = x_n \wedge \vec{W} = \vec{w} \right) \implies \neg \varphi. \quad (3.15)$$

Finally, in equation 3.12, we use exactly these values to also fix  $\vec{W}^* = \vec{W}$ ,  $\vec{w}^* = \vec{w}$ , and  $\vec{x}'_{(n-1)^*} = \vec{x}'_{(n-1)}$ , and reformulate this equation as

$$E_4 := (\Psi'' \wedge \vec{X}_{(n-1)} = \vec{x}'_{(n-1)} \wedge \vec{W} = \vec{w}) \implies \neg\varphi. \quad (3.16)$$

It is then a matter of equivalence transformations to show that

$$(\Psi'' \implies \Psi') \implies ((E_1 \wedge E_2) \implies (E_3 \wedge E_4)) \quad (3.17)$$

is a tautology, which proves the lemma. □

### 3.6.2 AC2 Encoding Proof

**Theorem 3.3.** *Formula  $F$  constructed within Algorithm 2 is satisfiable iff AC2 holds for a given  $M$ ,  $\vec{u}$ , a candidate cause  $\vec{X}$ , and a combination of events  $\varphi$ .*

*Proof.* The proof consists of two parts.

**Part 1.**  $SAT(F) \implies AC2$ , AC2 holds if  $F$  is satisfiable

We show this by contradiction. Assume that  $F$  is satisfiable and AC2 does not hold. Based on  $F$ 's truth assignment,  $\vec{v}'$ , we cluster the variables into:

$$F := \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i) \wedge \bigwedge_{i=1\dots\ell} f(X_i = \neg x_i) \wedge \bigwedge_{i=1\dots m, \exists j \bullet X_j = V_i} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i))$$

**1.**  $\vec{X}$ : each variable is fixed exactly to the negation of its original value, i.e.,  $X_i = \neg x_i \forall X_i \in \vec{X}$  (recall  $\vec{X} \subseteq \vec{V}$ ). **2.**  $\vec{W}^*$ : variables in this group, if they exist, have equal truth and original assignments, i.e.,  $\langle W_1^*, \dots, W_s^* \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \implies W_i^* \neq W_j^*) \wedge (W_i^* = V_j \Leftrightarrow v'_j = v_j)$  **3.**  $\vec{Z}$ : variables in this group evaluate differently from their original evaluation, i.e.,  $\langle Z_1, \dots, Z_k \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \implies Z_i \neq Z_j) \wedge (Z_i = V_j \Leftrightarrow v'_j \neq v_j) \wedge (\forall i \nexists j \bullet Z_i = X_j)$ .

Using  $\vec{W}^*$ ,  $\vec{Z}$ , we re-write  $F$  as  $F'$  which is also satisfiable.

$$F' := \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i) \wedge \bigwedge_{i=1\dots\ell} f(X_i = \neg x_i) \wedge \bigwedge_{i=1\dots s} f(W_i^* = w_i^*) \wedge \bigwedge_{i=1\dots k} (Z_i \leftrightarrow F_{Z_i})$$

Recall that  $M$  is acyclic; therefore there is a unique solution to the equations. Let  $\Psi$  be the equations in  $M$  without the equations that define the variables  $\vec{X}$ . Let  $\Psi_k$  be  $\Psi$  without the equations of some variables in a set  $\vec{W}_k$ . Since AC2 does not hold,  $\forall k \bullet \vec{W}_k \subseteq V \setminus X \implies (\vec{X} = \neg\vec{x} \wedge \vec{W}_k = \vec{w}_k \wedge \Psi_k \wedge \neg\varphi)$  evaluates to *false*. In case  $\vec{W}_k = \vec{W}^*$ , the previous unsatisfiable formula is equivalent to the satisfiable  $F'$ , implying a contradiction.

**Part 2.**  $AC2 \implies SAT(F)$ ;  $F$  is satisfiable if  $AC2$  holds

Assume that  $AC2$  holds and  $F$  is unsatisfiable. Then  $\exists \vec{W}, \vec{w}, \vec{x}' \bullet (M, \vec{u}) \models (\vec{W} = \vec{w}) \implies (M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}] \neg \varphi$ . By definition [74],  $(M, \vec{u}) \models [Y_1 \leftarrow y_1 \dots Y_k \leftarrow y_k] \varphi$  is equivalent to  $(M_{Y_1 \leftarrow y_1 \dots Y_k \leftarrow y_k}, \vec{u}) \models \varphi$ , i.e., we replace specific equations in  $M$  to obtain a new model  $M' = M_{Y_1 \leftarrow y_1 \dots Y_k \leftarrow y_k}$ . So, we replace the equations of the variables in  $\vec{X}, \vec{W}$  in  $M$  to obtain a new model,  $M'$ , such that  $(M', \vec{u}) \models \neg \varphi$ . Equations of  $\vec{X}, \vec{W}$  variables are now of the form  $V_i = v_i$ , i.e., each variable is equal to a constant value. Note that  $M'$  is only different from  $M$  in the equations of  $\vec{X}, \vec{W}$ . Hence,  $M'$  is acyclic and has a unique solution for a given  $\vec{U} = \vec{u}$ . We construct a formula,  $F'$  (shown below), that is a conjunction of the variables in sets  $X', W', U$  in  $M'$ . Because of their equations, each variable is represented by a constant, i.e., a positive or a negative literal. Based on the nature of this formula, it is satisfiable with exactly the same truth assignment as the unique solution of  $M'$ .

$$F' := \bigwedge_{i=1 \dots n} f(U_i = u_i) \wedge \bigwedge_{i=1 \dots s} f(W'_i = w'_i) \wedge \bigwedge_{i=1 \dots \ell} f(X'_i = x'_i)$$

Now, we add the remaining variables, i.e.,  $\forall i \bullet V_i \notin (\vec{X} \cup \vec{W})$ , as formulas using the  $\leftrightarrow$  operator. The overall formula  $F''$ , is satisfiable because we have an assignment that makes each equivalence relation true.

$$F'' := F' \wedge \bigwedge_{i=1 \dots m, \exists j \bullet X_j = V_i, W_j = V_i} (V_i \leftrightarrow F_{V_i})$$

We have  $(M', \vec{u}) \models \neg \varphi$ , which says that the model evaluates  $\neg \varphi$  to true with its unique solution (same assignment of  $F''$ ). We add another clause to  $F''$  which evaluates to true and keeps the formula satisfiable. That is,  $F''' := F'' \wedge \neg \varphi$ . Last, we only have to show the relation between ( $F$  from Algorithm 2 and  $F'''$ ). We can rewrite  $F$  (shown at the beginning of the proof) such that we remove all disjuncts of the form  $(V_i \leftrightarrow F_{V_i})$  for the variables in  $\vec{W}$ . Similarly, we remove all disjuncts of the form  $f(V_i = v_i)$  for all the variables that are not in  $\vec{W}$ . According to our assumption,  $F$  is still unsatisfiable, since we removed disjunctions from the clauses. Then, we reach a contradiction since  $F$  is equivalent to  $F'''$  which is satisfiable for the same clauses.  $\square$

### 3.6.3 AC3 Encoding Proof

**Theorem 3.4.** *Algorithm 3 returns false iff  $\vec{X}$  is a non-minimal cause.*

Before presenting the proof, we define the term *non-minimal cause*.

**Definition 3.8.** *Non-minimal Cause is a candidate cause  $\vec{X}_{(n)} = \vec{x}_{(n)}$  that satisfies  $AC2$  yet contains at least one element  $X_n$  that satisfies one of the following conditions*

- **NMC1.** AC2 holds for the smaller cause  $\vec{X}_{(n-1)}$  regardless of whether  $X_n \in \vec{W}$  or not, i.e.,  $(M, \vec{u}) \models [\vec{X}_{(n-1)} \leftarrow \vec{x}'_{(n-1)}, \vec{W} \leftarrow \vec{w}] \neg\varphi$  and  $(M, \vec{u}) \models [\vec{X}_{(n-1)} \leftarrow \vec{x}'_{(n-1)}, X_n \leftarrow x_n, \vec{W} \leftarrow \vec{w}] \neg\varphi$  both hold.
- **NMC2.** AC2 holds for the smaller cause  $\vec{X}_{(n-1)}$  only if  $X_n \notin \vec{W}$ , i.e.,  $(M, \vec{u}) \models [\vec{X}_{(n-1)} \leftarrow \vec{x}'_{(n-1)}, \vec{W} \leftarrow \vec{w}] \neg\varphi$

Informally, **NMC1** deals with irrelevant variables, i.e., those that do not affect the cause relation to the effect. **NMC2** targets the case of relevant variables that are affected by the cause but not necessary for it to be a cause.

**Theorem 3.4.** Algorithm 3 returns false iff  $\vec{X}$  is a non-minimal cause.

*Proof.*

**Part 1.** If the cause is a non-minimal cause, then Algorithm 3 returns false.

For the algorithm to return false,  $G$  must be satisfiable first, and the cardinality check is passed. So we prove this part by showing that if any of the conditions in Definition 3.8 hold then  $G$  is satisfiable and the check is passed and the algorithm returns false.

1. Recall  $G := \neg\varphi \wedge \bigwedge_{i=1\dots\ell} f(U_i = u_i) \wedge \bigwedge_{i=1\dots m, \exists j \bullet X_j = V_i} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i)) \wedge \bigwedge_{i=1\dots\ell} (X_i \vee \neg X_i)$ . Let us rewrite the formula to abstract the first part as,  $G := G_{base} \wedge \bigwedge_{i=1\dots n} (X_i \vee \neg X_i)$ .
2. Note how  $X_{(n)}$  is added to  $G$  as  $(X_1 \vee \neg X_1) \wedge (X_2 \vee \neg X_2) \dots (X_n \vee \neg X_n)$ . Re-write this big conjunction to have its equivalent disjunctive normal form (DNF) i.e.,  $(\neg X_1 \wedge \neg X_2 \dots \wedge \neg X_n) \vee (\neg X_1 \wedge \neg X_2 \dots \wedge X_n) \dots \vee (X_1 \wedge X_2 \dots \wedge X_n)$ . Assume wlog that all the original values of  $X_{(n)}$  (which lead to  $\varphi$  holding true) were true, hence to check them in AC2 we present them as negative literals like  $\neg X_i$ . Looking at the DNF, we have  $2^n$  clauses that list all the possible cases of negating or fixing the elements in  $\vec{X}$ . Then, we partition  $G$  according to the clauses, i.e.,  $G := G_1 \vee G_2 \dots G_{2^n}$ , where  $G_1 := G_{base} \wedge (\neg X_1 \wedge \neg X_2 \dots \wedge \neg X_n)$ . In the case of the clause where all variables  $X_{(n)}$  are negated, the corresponding  $G$ , i.e.,  $G_1$ , is exactly formula  $F$  from Algorithm 2.
3. Each  $G_i$ , other than  $G_1$ , fixes some group of elements to their original evaluation ( $X_i$ ) and negates some, possibly none ( $G_{2^n}$ ), other elements ( $\neg X_i$ ). Clearly,  $G_i$  is an  $F$  formula (from Algorithm 2) for all the negated variables, in a clause, as  $\vec{X}$  but with  $x$  special fixed variables that are added to  $\vec{W}$ . Hence, a  $G_i$  is a check of AC2 for a specific subset of the causes given that the other part (fixed) of the cause is in  $\vec{W}$ . This is the case of **NMC1** in Definition 3.8, i.e., AC2 still holds after transferring some elements  $\vec{X}^*$  from  $\vec{X}$  to  $\vec{W}$ .  $\vec{X}^*$  is guaranteed to be expressed in one of the  $2^n$  clauses, and hence, by a specific  $G_k$ . According to Theorem 3.3, such a  $G_k$  is satisfiable since AC2 holds. Then, for a non-minimal cause based on **NMC1**,  $G$  is satisfiable since  $G_k$

is satisfiable. In this case, we only have to show that it passes the cardinality check in the algorithm. It is clear that  $\forall i \in \vec{X}^* v'_i = v_i$  since they will be in  $\vec{W}$ . This makes the condition in the algorithm evaluates to true and hence, the algorithm returns false.

4. Similarly, for the second case **NMC2**, i.e., the non-minimal part should not be in  $\vec{W}$ . AC2 holds for the non-minimal cause, i.e,  $F$  and  $G_1$  are satisfiable and then  $G$  is also satisfiable. Since the non-minimal parts in this case are not in  $\vec{W}$  or  $\vec{X}$ , then they follow their equations in the model, and hence  $\exists i \bullet v'_i = [\vec{V} \mapsto \vec{v}'] F_{X_i}$ , which results in false returned by the algorithm.

**Part 2.** *If Algorithm 3 returns false, then the cause is a non-minimal cause*

To prove this part, we show that if  $\vec{X}$  is a minimal cause, the algorithm does not return false. The algorithm returns false if  $G$  is satisfiable ( $\vec{X}$  or a subset of it fulfill AC2), and the cause passes the cardinality check. A minimal cause will have a satisfiable  $G$ . For the cardinality check, by Lemma 3.2, a cause should have all its elements negated. Hence the first conjunct in line Line 5 of Algorithm 3 will be *true* for each element. If the second conjunct in the same line ( $v'_i \neq [\vec{V} \mapsto \vec{v}'] F_{X_i}$ ) evaluates to *false* for any element then, this is not a minimal cause. Hence, for a minimal cause the two conjuncts will evaluate to true for all the elements in  $\vec{X}$ , and then a false is never returned for such a case.  $\square$

### 3.6.4 Optimized AC3 Encoding Proof

**Theorem 3.5.** *Formula  $G'$  constructed with Equation 3.5 is satisfiable iff AC3 is violated.*

*Proof.* The proof follows from the fact that to check minimality, it is sufficient to check  $\vec{X}$ 's subsets of cardinality  $k$  where  $k = |\vec{X}| - 1$ , i.e., the subsets of  $\vec{X}$  with one element less. We denote these subsets by  $\mathcal{P}_k(X)$ . A set  $\vec{X}$  of size  $l$  has  $l$  subsets of size  $l - 1$ .

1. Recall that  $G := \neg\varphi \wedge \bigwedge_{i=1\dots\ell} f(U_i = u_i) \wedge \bigwedge_{i=1\dots m, \exists j \bullet X_j = V_i} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i)) \wedge \bigwedge_{i=1\dots\ell} (X_i \vee \neg X_i)$ .
2.  $G' := G \wedge \bigvee_{i=1\dots\ell} ((X_i \leftrightarrow F_{X_i}) \vee (f(X_i = x_i)))$ . Let us take the *base case*:  $l = 2$ , then  $G' := G \wedge ((X_1 \leftrightarrow F_{X_1}) \vee (f(X_1 = x_1))) \vee ((X_2 \leftrightarrow F_{X_2}) \vee (f(X_2 = x_2)))$ . If we distribute the conjunction over disjunction, then  $G' := G \wedge ((X_1 \leftrightarrow F_{X_1}) \vee (f(X_1 = x_1))) \vee G \wedge ((X_2 \leftrightarrow F_{X_2}) \vee (f(X_2 = x_2)))$ . Call  $G \wedge ((X_i \leftrightarrow F_{X_i}) \vee (f(X_i = x_i)))$ ,  $G_i^*$ . Then  $G' = G_1^* \vee G_2^* \dots \vee G_l^*$ . For the base case  $G' = G_1^* \vee G_2^*$
3. A  $G_i^*$  represents the case where one cause variable  $X_i$  is removed from the cause set by adding this clause  $((X_i \leftrightarrow F_{X_i}) \vee (f(X_i = x_i)))$  which then makes  $G_i^*$  only satisfiable if  $X_i$  was not negated, i.e., not part of the cause.  $G_i^*$  then can be seen as an AC2 check of a smaller cause (in relation with formula  $F$  from [98]). Since  $G'$  is a disjunction of all  $G_i^* \in \mathcal{P}_k(X)$ ,  $G'$  is an AC2 check of all subsets of  $\vec{X}$  with size  $k$ .  $G'$  is only

satisfiable if one or more  $G_i^*$  clauses are satisfiable. This satisfiability of a  $G_i^*$  makes the corresponding  $X_i$  an irrelevant cause and hence AC3 is violated.

4. By induction, a cause  $\vec{X}$  of size  $n$  written (by distributing the conjunction over disjunction) as  $G' = G_1^* \vee G_2^* \dots G_n^*$  is only satisfiable if AC2 holds for a subset-cause of size  $n - 1$ , and consequently AC3 is violated.

□

### 3.7 Summary

It is difficult to devise automated assistance for causality reasoning in modern socio-technical systems. Causality checking, according to the formal definitions, is computationally hard. Therefore, efficient approaches that scale to the complexity of such systems are required. In the course of this, we proposed an intelligent way to utilize SAT solvers to check actual causality in binary models on a large scale that we believe to be particularly relevant for accountability purposes, specifically when large models are generated automatically from existing documentation. The baseline of the approach lies in how the encoding of AC2 is done in a way that allows for an efficient conclusion of  $\vec{W}$ . We extended this baseline to check AC3 and optimized our solution for efficiency.

We formally proved the soundness of our approach and empirically showed that it can efficiently compute actual causality in large binary models. Even with only 30 variables, determining causality in a brute force manner is incomputable, whereas our SAT-based approach returned a result for such cases in 1 ms. In addition, causal models consisting of more than 4000 endogenous variables were still handled within seconds using the proposed approach. We have also shown one idea to enhance the quality of the answer, i.e., a minimal  $\vec{W}$ , and one idea to enhance the performance, i.e., combining the two algorithms.





# 4 Actual Causality Computations as Optimization Problems

*This chapter presents a formulation of different notions of actual causality computations over binary models as optimization problems. Parts of this chapter have previously appeared in the publication [97], co-authored by the author of this thesis.<sup>1</sup>*

## 4.1 Introduction

Recall that we distinguish two notions of reasoning: *checking* and *inference*. *Checking* refers to verifying if a candidate cause is an actual cause of an effect, i.e., answering the question “is  $\bar{X}$  a cause of  $\varphi$ ?” *Inference* involves finding a cause without any candidates, i.e., answering the question “why  $\varphi$ ?” In Chapter 3, we presented different encodings for checking causality in acyclic models *with binary variables* based on the Boolean satisfiability problem (SAT). The first encoding requires solving an ALL-SAT problem to verify minimality, i.e., enumerating all the satisfying assignments of a formula. To solve an ALL-SAT problem, modern solvers typically add clauses (of size equal to the number of variables) called blocking clauses, to prevent already-found solutions and let the solver find a new solution [69]. This method has benefits, especially for formulas that have a small number of solutions [190]. However, the two downsides are the potential memory saturation and the slow-down of the solver [204, 190, 69]. In addition to the limitation to checking, the normal SAT approach in Chapter 3 then fails to scale for larger models, especially when the cardinality of the cause is big.

Large models of causal dependencies are likely to occur, especially if generated automatically from other sources [205, 95, 141]. Furthermore, models of real-world accidents are sufficiently large to require efficient approaches. For example, the model of the 2002 mid-air collision in southern Germany consists of 95 factors [187] (discussed in detail in Section 9.4.2), 2006 Amazonas collision consists of 137 factors [184]; such models are expected to grow in size with data-driven causal discovery approaches. Therefore, extending the approach in Chapter 3, we present a novel approach to formulate actual causality computations in binary models as optimization problems [25, 130, 128]. We show how

---

<sup>1</sup>Parts of this chapter are reprinted by permission from Springer Nature: Lecture Notes in Computer Science, vol 12302. From Checking to Inference: Actual Causality Computations as Optimization Problems, Ibrahim A., Pretschner A. (2020).

to construct quantifiable notions within counterfactual computations, and use them for checking and inference.

We encode our checking approach as integer linear programs (ILP), or weighted MaxSAT formulae [128]. Both are well-suited alternatives for Boolean optimization problems. However, MaxSAT has an inherent advantage with binary propositional constraints [128]. On the other hand, ILP has an expressive objective language that allows us to tackle the problem of causality inference as a multi-objective program, and enables the extension to non-binary models. Accordingly, we contribute an approach with *three* encodings. The first two cover causality *checking*, and better they can determine a *minimal* HP cause from a potentially non-minimal candidate cause; we refer to this ability as *semi-inference*. The third encoding tackles causality *inference*. All these encodings benefit from the rapid development in solving complex and large (tens of thousands of variables and constraints) optimization problems [142, 110, 10].

We consider the work in this chapter to be the first to provide an efficient solution to the problem of computing actual causality (checking and inferring), according to HP, for a large class of models (binary models) without any dependency on domain-specific technologies.

## 4.2 Checking and Semi-inference Queries as Optimization Problems

The conclusion of the SAT approaches aid us in determining whether a given  $\vec{X} = \vec{x}$  is a *minimal, counter-factual* cause of  $\varphi$ . If it is not, we cannot use them to *find* a minimal cause from within  $\vec{X}$ , i.e., *semi-inference*. We, also, cannot use them to find a cause without requiring a candidate cause, i.e., *inference*. To efficiently achieve such abilities, we present a novel formulation of causal queries as optimization problems. For that, we conceptualize a technique to check AC2 and AC3 as one problem (AC1 is explicitly checked solely). The result of solving this problem can then be interpreted to conclude AC2,  $\vec{W}$ , AC3, and, better, what is a minimal subset of the cause if AC3 is violated (semi-inference). To compare the efficiency, we formulate the problem as an integer program, and a MaxSAT formula. Both techniques solve the problem based on an *objective* function—a function whose value is minimized or maximized among feasible alternatives.

### 4.2.1 The Objective in Causality Checking

To quantify an *objective* within a causal check, we introduce an *integer* variable that we call the *distance* variable (formalized in Equation 4.1). The *distance* is the count of the cause variables ( $\vec{X}$ ) whose value assigned by the (ILP or MaxSAT) solver ( $x'_i$ ) is different from their value under the given context (actual evaluation  $x_i$  that led to  $\varphi$  hold true). Similar to the Hamming distance [84], the distance is a metric to measure the difference between the cause values when  $\varphi$  holds true and when it holds false. In other words, we quantify the distance between the actual and the counterfactual world. Equation 4.1 shows the distance

as a sum function of the deltas between variable  $X_i$ 's actual ( $x_i$ ) and counterfactual value ( $x'_i$ ). Since an actual cause is non-empty by definition, the *distance* is bounded to have a value greater or equal to 1. Further, the distance is less or equal to the size of  $\vec{X}$  ( $\ell$ ), i.e.,  $1 \leq \text{distance} \leq \ell$ .

$$\text{distance} = \sum_{i=1}^{\ell} d(i) \quad s.t. \quad d(i) = \begin{cases} 1 - x'_i, & x_i = 1 \\ x'_i, & x_i = 0 \end{cases} \quad (4.1)$$

According to AC3, we define the *objective function*. Clearly, our objective function is to *minimize* the distance between the actual values of the candidate cause and the computed causes such that the distance is greater than or equal to 1. So, we encode a causality check as an optimization problem that aims to *minimize* the number of causes variables while satisfying the constraints for AC2 (counterfactuality and  $\vec{W}$ ). Next, we present the specific ingredients for the ILP formulation, and the MaxSAT encoding; then we discuss how to interpret the results to infer a minimal cause from a possibly non-minimal cause.

#### 4.2.2 ILP Formulation

**ILP** is an optimization program with integer variables and linear constraints and objectives. To formulate such a program, three elements have to be specified: *the decision variables*, *the constraints*, and *the objective* [25]. Our *decision variables* are, in addition to the *distance* variable, the set of exogenous and endogenous variables from the model, i.e.,  $\mathcal{U} \cup \mathcal{V}$ . Since we consider binary variables, variables are bound to have values of 0 or 1.

Since ILP and SAT solvers can be used as complementary tools, the translation from SAT to ILP is standard [130, 129]. Therefore, we reuse formula  $G$  from Section 3.3.2 (shown in Equation 4.2) to create the ILP *constraints*. However, since we aim to optimize the SAT answer, we add more *constraints*. The constraints from  $G$  already contain the **a.)** effect not holding true, **b.)** exogenous variables set to their values (the context), **c.)** each endogenous variable either follows the model equation or the actual value, i.e., part of the set  $\vec{W}$ , **d.)** each element in the cause set  $\vec{X} = \vec{x}$  is not constrained, i.e., its equation is removed. Transforming these constraints (on the Conjunctive Normal Form (CNF) level) into linear inequalities is straightforward; we have clauses that can be reduced to ILP directly. For example, we express  $y = x_1 \vee x_2$  as  $1 \geq 2 * y - x_1 - x_2 \geq 0$  [25, 130, 129].

$$G := \neg\varphi \wedge \bigwedge_{i=1 \dots n} f(U_i = u_i) \wedge \bigwedge_{i=1 \dots m, \exists j \bullet X_j = V_i} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i)) \quad (4.2)$$

In addition to the constraints inherited from  $G$ , we add a special constraint for the *distance* variable. Besides setting the bounds of the distance variable, we add a constraint to calculate it by summing the absolute difference between each actual value in  $\vec{x}$  and the corresponding value in the ILP solution  $\vec{x}'$ . Lastly, we define the *objective function*. It is clear by now that our objective function is to *minimize* the distance between the actual values of the candidate

cause and the computed causes such that the distance is greater than or equal to 1. We solve a causality check as an optimization problem that aims to *minimize* the number of causes variables while satisfying the constraints for AC2 (counter-factuality and  $\vec{W}$ ).

### 4.2.3 MaxSAT Encoding

The maximum satisfiability problem (MaxSAT) is an optimization variant of SAT [128]. In contrast to SAT, which aims to find a satisfying assignment of all the clauses in a formula, MaxSAT aims to find an assignment that maximizes the number of satisfied clauses. Thus, MaxSAT allows the potential that some clauses are unsatisfied. In this chapter, we use *partial* MaxSAT solving, which allows specific clauses to be unsatisfied, referred to as *soft clauses*; contrary to the *hard* clauses that must be satisfied [128]. A soft clause can be assigned a *weight* to represent the cost of not satisfying it. In essence, a weighted partial MaxSAT problem is a minimization problem that minimizes the cost over all solutions. Unlike ILP, the objective in MaxSAT is immutable. Thus, we need to construct our formula in a way that mimics the concept of the distance.

As shown in Equation 4.3, the MaxSAT encoding also uses  $G$  (shown in Equation 4.2) as a base.  $G$  embeds all the mandatory parts of any solution. Thus, we use the CNF clauses of  $G$  as *hard* clauses. On the other hand, we need to append the cause variables ( $\vec{X}$ ) as *soft* clauses (underlined in Equation 4.3). Since the solver would minimize the cost of unsatisfying the ( $\vec{X}$ ) clauses, we represent each cause variable as a literal according to its original value (when  $\varphi$  holds). Because this is already in CNF, it is easier to assign weights. We assign 1 as a cost for unsatisfying each cause variable's clause, i.e., when  $X_i$  is negated in the (solved) counterfactual world. Then, the overall cost of unsatisfying the underlined parts of the formula is the count of the negated causes, i.e., the size of the minimal cause. Essentially, this concept maps directly to the *distance*, which the MaxSAT solver will minimize. In contrast to ILP, we cannot specify a lower bound on the MaxSAT objective. Thus, we need to express the non-emptiness of a cause, as *hard* clauses. A *non-empty* cause means that at least one cause variable  $X_j$  does not take its original value, *and* does not follow its equation due to an intervention. The first conjunction (after  $G$ ) in Equation 4.3 ensures the first requirement, while the second corresponds to the second case. The first ensures that not all the variables are equal to their original value. The second ensures that not every variable is allowed to evaluate according to its equation.

$$G_{max} := G \wedge \neg \left( \bigwedge_{i=1 \dots \ell} f(X_i = x_i) \right) \wedge \neg \left( \bigwedge_{i=1 \dots \ell} X_i \leftrightarrow F_{X_i} \right) \wedge \underbrace{\bigwedge_{i=1 \dots \ell} f(X_i = x_i)} \quad (4.3)$$

Clearly,  $G_{max}$  cannot be satisfied completely. We cannot have a non-empty cause, yet represent *all* the cause variables according to their original values (soft clauses). This is precisely the reason for using this formula with MaxSAT. We want the maximum number of (soft clauses) literals to be true.

---

**Algorithm 4** Interpreting the Optimization Problem's Results
 

---

**Input:** context  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ , effect  $\varphi$ , candidate cause  $\langle X_1, \dots, X_\ell \rangle = \langle x_1, \dots, x_\ell \rangle$ , evaluation  $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$ ,  $\vec{C}$ , objective

- 1: **function** CHECKCAUSE( $\vec{U} = \vec{u}, \varphi, \vec{X} = \vec{x}, \vec{V} = \vec{v}, \vec{C}, \text{objective}$ )
- 2:     **if**  $\langle U_1 = u_1 \dots U_n = u_n, V_1 = v'_1 \dots V_m = v'_m \rangle = \text{solve}(\vec{C}, \text{objective})$  **then**
- 3:          $\vec{X}_{min} := \langle X'_1 \dots X'_d \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \Rightarrow X'_i \neq X'_j) \wedge (X'_i = V_j \Leftrightarrow v'_j \neq v_j)$
- 4:          $\vec{W} := \langle W_1 \dots W_s \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \Rightarrow W_i \neq W_j) \wedge (W_i = V_j \Leftrightarrow v'_j = v_j)$
- 5:         **return**  $\vec{X}_{min}, \vec{W}$
- 6:     **else return** *infeasible (unsatisfiable)*
- 7:     **end if**
- 8: **end function**

---

#### 4.2.4 Results Interpretation

With the above, we illustrated the formulation of a causal checking problem. We now discuss how to translate their results to a causal answer once they are solved; Algorithm 4 formalizes this. The evaluation, in the input, is a list of the variables in  $M$  and their values under  $\vec{u}$ . Assuming  $\vec{C}$  is a representation of the optimization problem that encodes the causal model  $M$  (a set of linear constraints without the objective, or hard/soft clauses), then in Line 2, we solve this problem and process the results in Lines 3-4. The feasibility (satisfiability) of the problem implies that either  $\vec{X}$  or a non-empty subset of it is a minimal cause (fulfills AC2 and AC3). If the *distance* (cost returned by the MaxSAT solver) equals the size of  $\vec{X}$ , then the whole candidate cause is minimal (this step is not explicitly performed in Algorithm 4 because it can be concluded from the next step). Otherwise, to find a minimal cause  $\vec{X}_{min}$  (semi-inference), we choose the parts of  $\vec{X}$  that have different values between the actual and the solved values (Line 3). To determine  $\vec{W}$ , in Line 4, we take the variables whose solved values are the same as the actual evaluation (potentially including  $\vec{X}$  variables). Obviously, this is not a minimal  $\vec{W}$ , which is not a requirement for *checking* HP [98]. If the model is *infeasible or unsatisfiable*, then HP for the given  $\vec{X}$  (checking) and its subsets (semi-inference) does not hold. Note that an optimization problem can have more than one optimal solution. This implies that multiple minimal causes are found. We collect them all, but we present the first one. To pick one of the causes, we need to employ additional metrics.

#### 4.2.5 Example

To illustrate our approach, we show the ILP and MaxSAT encodings to answer the query is  $ST = 1, BT = 1$  a cause of  $BS = 1$ ? Figure 4.1 (left part) shows the generated ILP program according to our approach, and its solution (right part). On the other hand, the MaxSAT formula is shown in Equation 4.4.

$$\begin{array}{ll}
 \min & distance \\
 \text{s.t.} & BS = 0 \\
 & ST_{exo} = 1 \\
 & BT_{exo} = 1 \\
 & -SH + BS \geq 0 \\
 & -BH + BS \geq 0 \\
 & -ST + SH \geq 0 \\
 & BT - BH \geq 0 \\
 & -SH - BH \geq -1 \\
 & ST + BT + d = 2 \\
 & V_i \in [0, 1] \quad \forall V_i \in V, \quad distance \in [1, 2]
 \end{array}
 \qquad
 \begin{array}{l}
 distance = 1 \\
 BS = 0 \\
 ST_{exo} = 1 \\
 BT_{exo} = 1 \\
 SH = 0 \\
 BH = 0 \\
 ST = 0 \\
 \mathbf{BT} = 1
 \end{array}$$

Figure 4.1: The Program and the Solution of the Example

$$\begin{aligned}
 G_{max} = & \neg BS \wedge ST_{exo} \wedge BT_{exo} \wedge (BS \leftrightarrow SH \vee BH) \wedge ((SH \leftrightarrow ST) \vee SH) \wedge \\
 & ((BH \leftrightarrow BT \wedge \neg SH) \vee \neg BH) \wedge \neg(ST \wedge BT) \wedge \underline{(ST \wedge BT)}
 \end{aligned} \tag{4.4}$$

Both encodings are solved with a  $distance(d)$  (cost) value of 1, which indicates that  $ST, BT$  is not minimal, and a cause of size 1 is (semi)-inferred, namely  $ST$ . The optimal assignment  $(\neg BS, ST_{exo}, BT_{exo}, \neg SH, \neg BH, \neg ST, BT)$  showed that the constraints can be guaranteed without changing the value of  $BT$ , which violates AC3. Although this example shows the same result of the SAT-based approach, it clearly shows the quality enhancement of the answer, i.e., finding a specific non-empty, minimal cause rather than only checking the violation of AC3.

Theorem 4.1 states the soundness of our approach (for proofs see Section 4.5)

**Theorem 4.1.** *The generated optimization problem (ILP program or  $G_{max}$ ) is feasible iff AC3 holds for  $\vec{X}$  or a non-empty subset of  $\vec{X}$ .*

### 4.3 Causality Inference with ILP

All the previous approaches utilized the candidate cause  $\vec{X}$  to help describe a counterfactual world that proves  $\vec{X}$  is a cause of  $\varphi$ . To check AC2 in Section 3.3.1, we negated the causes in  $\vec{X}$ ; on the other hand, to check AC3 in Section 3.3.2, we removed the equations corresponding to the cause variables. In this section, we present a method,  $ILP_{why}$ , to *infer* causality without requiring  $\vec{X}$ . This approach can be utilized to answer questions of the nature *why  $\varphi$ ?* without hypothesizing anything about the cause. This method leverages a quantifiable metric (degree of responsibility), which can be formulated using ILP's expressive objective language; it is not clear to us if and how can this be encoded into

MaxSAT. Unlike checking, in inference, we cannot aid the solver in a description of the counterfactual world (e.g., negating values of  $\vec{X}$ ). Instead, we describe characteristics of the actual cause that have caused an effect  $\varphi$ .

Describing the characteristics of an actual cause entails translating the conditions of HP (according to Definition 2.4). However, since the conditions depend on the existence of a candidate cause  $\vec{X}$ , ( $ILP_{why}$ ) formulates the problem differently. We leverage the *degree of responsibility* ( $dr$ ) (Definition 2.5), which was introduced in Section 2.4. While the conditions are suitable for determining if  $\vec{X}$  is a cause,  $dr$  judges the “quality” of the cause based on an aggregation of its characteristics. Because we may find multiple causes for which the conditions hold,  $dr$  is reasonable for comparison. In other words, we instruct the solver to *find* a cause with the *maximum* degree of responsibility. We come back to this goal after we construct a formula  $G^*$  that is the base of  $ILP_{why}$ .

Analogous to all the previous formulas, we start our construction with the negated effect formula ( $\neg\varphi$ ), and inclusion of the context  $f(U_i = u_i)$ . To continue with a *why* query, we have two classes of endogenous variables: **1.** variables that belong to the effect formula, and **2.** all the other variables. Because the effect variables are not supposed to be part of the cause, we can represent the first group with the *equivalence* relation, i.e.,  $V_i \leftrightarrow F_{V_i}$ . The complicated part is the representation of the other variables. Since we are looking for the cause set, we need to allow each variable to be in one of the three categories: **a.** a cause variable, **b.** a contingency-set variable, **c.** a normal variable. Recall, in counterfactual computation, a cause variable does not follow its equation, *and* differs from its original value. A contingency-set variable, on the other hand, does not follow its equation while keeping its original value. A normal variable does follow its equation, regardless of whether being equal to the original value or not. Thus, we need to allow variables to be classified in any category in the “best” possible way.

To that end, we represent each (non-effect) variable  $V_i$  with a disjunction between the equivalence holding and not holding, *and* a disjunction between its original value and its negation. This is shown in Equation 4.5.

$$\left( (V_i \leftrightarrow F_{V_i}) \vee \neg (V_i \leftrightarrow F_{V_i}) \right) \wedge \left( V_{i_{orig}} \vee \neg V_{i_{orig}} \right) \quad (4.5)$$

Clearly, each clause in the previous equation is a tautology. However, this redundancy facilitates the classification into the three categories; more importantly, we can incentivize the solver to classify those variables according to specific criteria.

To be able to guide the solver, we add auxiliary boolean variables (indicators) to each clause (left and right parts of a disjunction). They serve two functions. The first is to *indicate* which clauses hold. Since the two parts of the conjunction are not mutually exclusive, i.e., a variable can follow its equation, yet have its original value, we need *two* indicators  $C^1 C^2$ . Secondly, similar to the concept of *distance* from Section 4.2.2, we use the indicators to describe the criteria of the solution. For each variable  $V_i$ ,  $C_i^1$  is appended to the first two clauses:  $\left( (V_i \leftrightarrow F_{V_i}) \wedge C_i^1 \right) \vee \left( \neg (V_i \leftrightarrow F_{V_i}) \wedge \neg C_i^1 \right)$ . Similarly,  $C^2$  is appended to the other clauses. The category of each endogenous variable is determined based on the values of

$C^1$  and  $C^2$ . A cause variable would have a  $C^1C^2 : 00$  (not following the formula nor its original value); a contingency-set variable has a  $C^1C^2 : 01$ ; and a normal variable has a  $C^1C^2 : 10$ , or  $11$ . The overall formula  $G^*$  is shown in Equation 4.6 (equivalence relations of effect variables are omitted for readability).

$$G^* := \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i) \wedge \bigwedge_{i=1\dots m} \left( \left( (V_i \leftrightarrow F_{V_i}) \wedge C_i^1 \right) \vee \left( \neg(V_i \leftrightarrow F_{V_i}) \wedge \neg C_i^1 \right) \right) \wedge \left( \left( V_{orig} \wedge C_i^2 \right) \vee \left( \neg V_{orig} \wedge \neg C_i^2 \right) \right) \quad (4.6)$$

**Theorem 4.2.** *Formula  $G^*$  is satisfiable iff  $\exists \vec{X} = \vec{x}$  such that AC2 holds for  $\vec{X}$*

So far, we have shown the construction of the formula and its parts that correspond to the constraints of the optimization problem. The important step is to define the *objective* of this formulation. As mentioned, we want to find an assignment to the constraints shown in  $G^*$  that corresponds to a cause with a maximum degree of responsibility. Let us recall the degree of responsibility as given in Definition 2.5 is  $\frac{1}{|X| + |W|}$ . Thus, *maximizing* the responsibility entails *minimizing* the sizes of both the cause and the contingency sets, i.e.,  $|X| + |W|$ .

Since the sizes of the three (cause, contingency, and normal) sets of variables form the overall model size (excluding the effect variables), then minimizing  $|X| + |W|$  is equivalent to maximizing the number of normal variables. The sum of values of  $C^1$  variables represents the number of normal variables; thus, *objective<sub>1</sub>* is to *maximize* the sum of  $C^1$  variables.

The above formulation minimizes  $\vec{X}$ , and  $\vec{W}$  as a whole, following *dr*. For our purpose, we argue it is valid to look for causes with higher responsibility first (fewer variables to negate or fix) and favor them over smaller causes. For example, if an effect has two actual causes: one with 2 variables in  $\vec{X}$ , 3 in  $\vec{W}$ , and the second with 1 variable in  $\vec{X}$ , 5 in  $\vec{W}$ , we pick the first.<sup>2</sup> That said, we still want to distinguish between  $\vec{X}$ , and  $\vec{W}$  in causes with the same *dr*. Assume we have two causes: the first with 2 variables in  $\vec{X}$ , 3 in  $\vec{W}$ , and the second with 3 in  $\vec{X}$ , 2 in  $\vec{W}$ . Although both are optimal solutions to *objective<sub>1</sub>*, we would like to pick the one with fewer causes.

Thus, we add *objective<sub>2</sub>* to *minimize* causes, i.e., the number of variables with  $C^1$  and  $C^2$  equal to 0. We use hierarchical objectives in ILP, for which the solver finds the optimal solution(s) based on the first objective, and then use the second objective to optimize the solution(s).

### 4.3.1 Why<sub>ILP</sub> Algorithm

We have discussed the building blocks of the formulation to infer causality based on responsibility. We wrap-up the approach with Algorithm 5. It follows a construction similar

---

<sup>2</sup>We acknowledge that this an arbitrary choice we made in our implementation; however, the same approach can be adapted to work with the other option.



**Algorithm 5** Causality Inference using  $ILP_{why}$ 


---

**Input:** causal model  $M$ , context  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ , effect  $\varphi$ , evaluation  $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$

- 1: **function** FINDCAUSE( $M, \vec{U} = \vec{u}, \varphi, \vec{V} = \vec{v}$ )
- 2:    $\langle Con_1, \dots, Con_n \rangle = \text{convertToILP}(\text{CNF}(G^*))$
- 3:    $obj_1 = \text{Maximize } \sum_{i=1}^m C_i^1 \text{ s.t. } obj_1 \leq |\vec{V}|$
- 4:    $obj_2 = \text{Minimize } \sum_{i=1}^m (1 - C_i^1) * (1 - C_i^2) \text{ s.t. } |\vec{V}| \geq obj_2 \geq 1$
- 5:   **if**  $\langle V_1 = v'_1 \dots V_m = v'_m, C_1^1 = c_1^1 \dots C_m^1 = c_m^1, C_1^2 = c_1^2 \dots C_m^2 = c_m^2 \rangle$   
        $\hookrightarrow = \text{solve}(Con, obj_1, obj_2)$  **then**
- 6:      $\vec{X}' := \langle X'_1 \dots X'_{obj_2} \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \Rightarrow X'_i \neq X'_j) \wedge (X'_i = V_j \Leftrightarrow \neg c_j^1 \wedge \neg c_j^2)$
- 7:      $\vec{W} := \langle W_1 \dots W_s \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \Rightarrow W_i \neq W_j) \wedge (W_i = V_j \Leftrightarrow (\neg c_j^1 \wedge c_j^2))$
- 8:     **return**  $\vec{X}', \vec{W}$
- 9:   **else return** *infeasible*
- 10: **end if**
- 11: **end function**

---

to Algorithm 2; however, without the dependency on a candidate cause set in the input.

The algorithm omits the construction of  $G^*$ . The choice to depend on a propositional formula to express the constraints of the equation seemed natural, given our previous approaches. To start formulating the query as an integer program, we turn the conjunctive normal form of  $G^*$  into a set of linear constraints in Line 2. We then complete constructing the program, in Line 3-Line 4, by adding the objectives. The first objective of the program, i.e., *objective*<sub>1</sub> concerns maximizing the degree of responsibility which, as discussed earlier, corresponds to maximize the sum of all the control variables  $C^1$ . The second objective, *objective*<sub>2</sub>, handles minimizing the size of the cause set.

Lastly, we process the results after solving the program in Line 5-Line 7. To reflect the assignments returned by the solver to answer a *why* query, we check the control variables. Generally, the feasibility of the program means that there is a cause of size at least 1 (exact size is *objective*<sub>2</sub>), which makes  $\varphi$  not hold, under the given context, with the maximum  $dr$  for the effect. For the details, we check the indicators of each variable. The cause is composed of variables that have  $C^1$  and  $C^2$  equal 0; variables in  $\vec{W}$ , have  $C^1 = 0$  and  $C^2 = 1$

## 4.3.2 Example

To illustrate the approach, using the rock-throwing example, Equation 4.7 shows the generated program for a why query. The query is *why did the bottle shatter*  $BS = 1$ ?

$$\begin{array}{ll}
 \max_{C1SUM} & \min_{C3SUM} \\
 \text{s.t.} & BS = 0 \\
 ST_{exo} = 1 & BT_{exo} = 1 \\
 -BS + SH + BH \geq 0 & BS - SH \geq 0 \\
 BS - BH \geq 0 & C1_{SH} - AT - SH \geq -1 \\
 C1_{SH} + SH + AT \geq 1 & -SH + AT - C1_{SH} \geq -1 \\
 SH - AT - C1_{SH} \geq -1 & C2_{SH} - SH \geq 0 \\
 SH - C2_{SH} \geq 0 & C1_{BH} - BT + SH - BH \geq -1 \\
 C1_{BH} + BH + BT \geq 1 & C1_{BH} + BH - SH \geq 0 \\
 -BH + BT - C1_{BH} \geq -1 & -BH - SH - C1_{BH} \geq -2 \\
 BH - BT + SH - C1_{BH} \geq -1 & C2_{BH} + BH \geq 0 \\
 -BH - C2_{BH} \geq -1 & C1_{ST} - ST_{exo} - AT \geq -1 \\
 C1_{ST} + AT + ST_{exo} \geq 1 & -AT + ST_{exo} - C1_{ST} \geq -1 \\
 AT - ST_{exo} - C1_{ST} \geq -1 & C2_{ST} - AT \geq 0 \\
 AT - C2_{ST} \geq 0 & C1_{BT} - BT_{exo} - BT \geq -1 \\
 C1_{BT} + BT + BT_{exo} \geq 1 & -BT + BT_{exo} - C1_{BT} \geq -1 \\
 BT - BT_{exo} - C1_{BT} \geq -1 & C2_{BT} - BT \geq 0 \\
 BT - C2_{BT} \geq 0 & -C1_{BT} - C2_{BT} - 2C3_{BT} \leq -1 \\
 -C1_{BT} - C2_{BT} - 2C3_{BT} \geq -2 & -C1_{BH} - C2_{BH} - 2C3_{BH} \leq -1 \\
 -C1_{BH} - C2_{BH} - 2C3_{BH} \geq -2 & -C1_{SH} - C2_{SH} - 2C3_{SH} \leq -1 \\
 -C1_{SH} - C2_{SH} - 2C3_{SH} \geq -2 & -C1_{ST} - C2_{ST} - 2C3_{ST} \leq -1 \\
 -C1_{ST} - C2_{ST} - 2C3_{ST} \geq -2 & C1_{BT} + C1_{BH} + C1_{SH} + C1_{ST} - C1_{sum} = 0 \\
 C1_{sum} + C1_{com_{sum}} = 4 & C3_{BT} + C3_{BH} + C3_{SH} + C3_{ST} - C3_{sum} = 0
 \end{array} \tag{4.7}$$

The solution of the program is shown in Equation 4.8. From the set of variables not included in the effect formula, we have *two* normal variables based on the sum of  $C^1$  variables ( $C_{sum}^1 = 2$ ). On the other hand, we have two variables that are part of either  $\vec{X}$ , or  $\vec{W}$ , shown by the value of the complement of  $C_{sum}^1$ . To infer the specific cause and contingency set variables, we look at the indicator variables. Based on the truth table,  $SH = 1$  is the actual cause of  $BS = 1$ , given that  $BH = 0$ . This is the result of having  $C_{SH}^1 = 0 \wedge C_{SH}^2 = 0$  as opposed to  $C_{BH}^1 = 0 \wedge C_{BH}^2 = 1$ .

$$\begin{array}{llll}
 C1_{sum} = 2 & C1_{com_{sum}} = 2 & C3_{sum} = 1 & \\
 ST_{exo} = 1 & BT_{exo} = 1 & BS = 0 & \\
 SH = 0 & C1_{SH} = 0 & C2_{SH} = 0 & C3_{SH} = 1 \\
 BH = 0 & C1_{BH} = 0 & C2_{BH} = 1 & C3_{BH} = 0 \\
 ST = 1 & C1_{ST} = 1 & C2_{ST} = 1 & C3_{ST} = 0 \\
 BT = 1 & C1_{BT} = 1 & C2_{BT} = 1 & C3_{BT} = 0
 \end{array} \tag{4.8}$$

The results returned by the approach are correct.  $SH$  is an actual cause of  $BS$ , according to HP. Under the given context, the inferred cause has a maximum degree of responsibility

$\frac{1}{2}$ ). All the instances of this example referred to  $ST$  as the actual cause; however, since the equation of  $SH$  is an identity function ( $SH = ST$ ), this does not compromise our result. Arguably, the depth of the cause, the (geodesic) distance between the nodes in the graph, can be taken into consideration. In this thesis, we do not consider this issue.

## 4.4 Evaluation

To evaluate their efficiency, we implemented our strategies as an open-source library. We used state of the art solvers: *Gurobi* [71] for  $ILP$ , and *Open-WBO* for  $MaxSAT$  [131]. In this section, we evaluate the performance, in terms of *execution time* and *memory allocation*, of the strategies in comparison with previous approaches.

### 4.4.1 Evaluating Checking and Semi-Inference

#### Experiment Setup

We used the same data-set we presented in Chapter 3. Recall that it contains 37 models, which included 21 *small* models ( $\leq 400$  endogenous variables)–from domains of causality, security, safety, and accident investigation– and 16 larger security models from an industrial partner, in addition to artificially generated models. A brief description of the models is given in Section 3.5.2, and the details of the models and the results can be found in Appendix A.

We formulated a total of 484 *checking* queries that vary in the context, cause, effect, and consequently differ in the result of AC1-AC3, the size of  $\vec{W}$ , and the size of the minimal cause. For the smaller models, we specified the queries manually according to their sources in literature and verified that our results match the sources. The approaches, including the previous ALL-SAT approach, answered these queries in under a second. For the larger models, we constructed a total of 224 checking queries. We specified some effects (e.g., root of  $BT$ , or *steal pass phrase* in  $SMK$ ) and used different contexts, and randomly selected causes (sizes 1, 2, 3, 4, 10, 15, and 50) from the models.

For these queries, we collected the results for two checking approaches from Chapter 3:  $SAT$  - the original SAT-based (requires ALL-SAT) approach,  $SAT\_OPT$  - the optimized SAT-based approach, and the two semi-inference approaches in this chapter:  $ILP$ , and  $MaxSAT$ . Although these approaches differ in their abilities, they answer the same queries. Further, as we define it in this thesis, semi-inference is also a checking procedure by definition. Thus, we argue that the comparison of these approaches provides useful insights regarding our encodings, algorithms, and the technologies used to solve them.

We ran each query for 30 warm-ups (dry-runs before collecting results to avoid accounting for factors like JVM warm-up threads), and 30 measurement iterations on an i7 Ubuntu machine with 16 GB RAM. We set the cut-off threshold to 2 hours.

## Discussion

We discuss the overall trends of the results; however, since we are interested in notions of checking, and semi-inference, we also discuss representative scenarios of these trends from our experiments, shown in Table 4.1. Each scenario is identified by the *model name*, its size, and an *ID* of the scenario; those are shown in the first three columns. From the causal query perspective, we only show, in the fourth column, the size of the candidate cause  $|\vec{X}|$ . Then, the results of the three conditions are displayed in columns AC1-AC3. The sizes of  $|\vec{W}|$ , and the minimal cause set  $|\vec{X}_{min}|$  are displayed in the eighth and ninth columns, respectively. Finally, for each approach the execution time, in seconds (s), and the memory allocation, in gigabytes (GB), are shown. The last two columns show the performance of the semi-inference approaches.

Model	$ \vec{V} $	ID	$ \vec{X} $	Result					Execution Time (s)			
				AC1	AC2	AC3	$ \vec{W} $	$ \vec{X}_{min} $	Memory consumption (GB)			
									SAT	SAT_OPT	ILP	MAX_SAT
LSP	41	4	4	Y	Y	N	2	1	0.0017 $1.5 \times 10^{-9}$	0.0015 $1.5 \times 10^{-9}$	0.0027 0.0009	0.001 $1e-9$
SMK	91	11	3	Y	Y	Y	0	3	0.013 0.008	0.0048 0.0027	0.0024 0.0026	0.002 0.002
Ueberlingen	95	5	4	Y	Y	N	88	3	0.805 0.0018	0.502 0.0032	0.488 0.0040	0.005 0.0021
BT_11	4095	35	4	Y	N	Y	N/A	4	7.37 2.03	7.50 2.03	3.24 1.05	0.22 0.07
ABT	4103	4	2	Y	Y	Y	4086	2	8.40 2.04	8.99 2.03	4.54 1.05	1.5 0.082
		5	5	Y	Y	N	4090	2	9.41 2.04	8.14 2.03	3.88 1.05	1.4 0.09
		6	10	Y	Y	N	4086	2	16.77 4.2	8.03 1.80	5.11 1.05	1.32 0.082
		7	11	N	Y	N	4086	2	26.29 4.18	8.07 1.80	5.26 1.05	1.35 0.082
		8	15	Y	Y	N	4086	2	N/A N/A	8.10 1.81	5.108 1.05	1.37 0.082
		9	15	N	Y	N	4080	5	7190 9.5	7.87 1.80	5.05 1.05	1.333 0.09
		10	15	N	Y	N	4080	5	7101 9.5	8.23 1.80	5.17 1.05	1.35 0.09
		11	50	Y	Y	N	4079	5	N/A N/A	8.6 2.04	4.80 1.05	1.44 0.082
		12	50	Y	Y	N	4068	11	N/A N/A	8.63 2.04	5.13 1.05	1.35 0.11
ABT2	8207	1	11	Y	Y	Y	8161	11	N/A N/A	18.96 4.0	22.8 4.0	5.67 0.22
		2	22	Y	Y	N	8191	11	N/A N/A	19.25 4.0	24.8 4.0	6.67 0.16

Table 4.1: Checking and Semi-inference Evaluation Scenarios.

For the smaller models, all the approaches performed efficiently. As shown in the first three rows of Table 4.1, the whole process of encoding, solving, and processing the results

in less than a second, and the memory consumption was under  $10MB$ ; hence, we exclude them from further discussion. The cases we discuss next are results of the bigger models, with which we tried to evaluate the different factors influencing the three approaches. Specifically, we show cases with candidate causes of 50 variables cardinality. In addition to specific scenarios in the table, we use cactus plots to compare the overall performance of the approaches. A point on the x-axis represents a query an approach answered ordered by the execution time, which is shown on the y-axis; a point  $(x, y)$  on the plot reads as  $x$  queries can be answered in  $y$  or less.

As expected, the experiments confirmed the problems seen in Chapter 3 with the SAT encoding—significant solver slow-down and memory exhaustion—[204]. Thus, as shown in Figure 4.2a, SAT only answered 187 of the 224 checking queries; for the remaining either it ran out of memory or took more than 2 hours. For instance, queries on  $SMK(6600)$  checking causes of sizes 2, 3, 4 were not answered because the program ran out of memory. With almost all answered queries, SAT took at least two to four times as much as ILP, and up to twenty times as much as MaxSAT. In extreme cases, SAT took around 113 minutes to finish, whereas others stayed under  $5s$  for the same cases. On the other hand, SAT\_OPT performed better than SAT but worse than the semi-inference approaches. SAT\_OPT answered 209 queries but was always slower than both ILP and MaxSAT.

Memory allocation of the two checking approaches, shown in Figure 4.2b, showed less difference with ILP and sometimes better allocation. Although it is not surprising that an ALL-SAT encoding performs poorly in some situations, the key result is that both ILP and MaxSAT provide more informative answers to a query while performing better than the checking approaches.

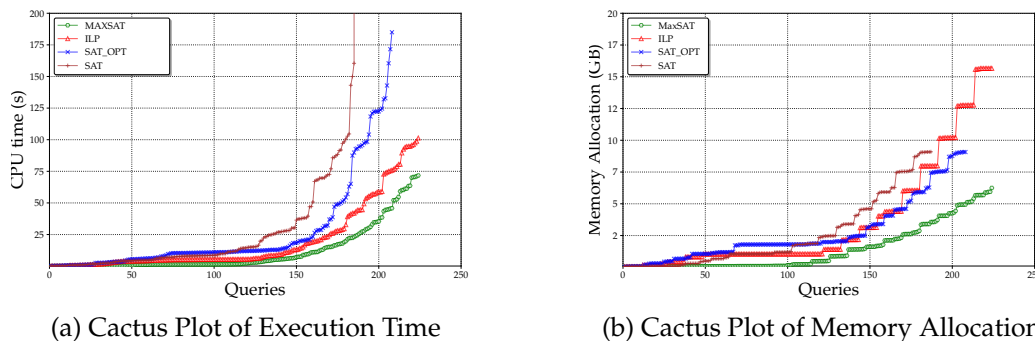


Figure 4.2: Execution Time and Memory Results on the Larger Models. A point on the x-axis represents a query an approach answered ordered by the execution time, shown on the y-axis; a point  $(x, y)$  on the plot reads as  $x$  queries can be answered in  $y$  or less.

According to our dataset, both ILP and MaxSAT, answered *all* (224) queries in less than 70 – 100 seconds. Especially for semi-inference, cases of *non-minimal* causes, and a minimal cause can be found, they are effective. For instance, with queries using  $ABT(4103)$ , we

found causes of size 2, 5, and 11 out of candidate causes of sizes 5, 10, 15, and 50. All these queries were answered in around 5s using ILP, and 2s using MaxSAT. For larger and more complex models e.g.,  $SMK(7150)$ , answering similar queries jumped to 98s with ILP and 71s MaxSAT.

As shown in Figure 4.2a and Figure 4.2b, MaxSAT outperformed ILP in execution time and memory; a scatter plot to compare them is shown in Figure 4.3. Clearly, the propositional nature of the problem gives an advantage to MaxSAT. Especially for easier queries, as shown in Figure 4.3 bottom left, MaxSAT is much faster because no linear transformation step is needed before solving, which explains why the gap between the two decreases among the larger queries. Further, we used Open-WBO solver—a solver that uses cores to initiate a set of (UN)SAT instances [131]—which performs better, especially when the number of hard clauses is high [11]. That said, in addition to the empirical comparison, we used ILP for binary counterfactual computations to incorporate quantifiable notions, to infer causality. This was achieved using multi-objective ILP in  $ILP_{why}$ .

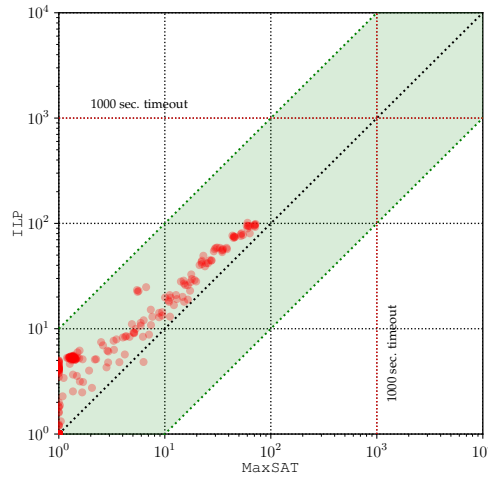


Figure 4.3: Log-log Scatter Plot of ILP vs MaxSAT

#### 4.4.2 Evaluating Inference

Using the same models, we evaluate our inference approach  $ILP_{why}$  presented in Section 4.3. We can reuse the checking queries for inference; the difference, however, is that we omitted to pick cause sets for the causal queries. Instead, we used the approach to answer *why* questions regarding the causal models under different contexts and different effects. As such, we created 180 inference queries including 67 queries of large models. We show some cases in Table 4.2, obtained from our experiments. We indicate the model and the number of its endogenous variables in the first two columns. We show the identifier of the causal query in the third column. In the query part of the table, we only show an identifier

of the context used and the effect formula. Regardless of the exact interpretation of the context and the effect, they emphasize the fact that a model may perform differently based on these factors. Lastly, in the last two columns, we show the performance of each case in terms of execution time in seconds and memory consumption in GB.

			Performance	
Model	$ \vec{V} $	ID	Execution Time (s)	Memory consumption (GB)
LSP	41	3	0.017	0.0041
SMK	91	11	0.062	0.017
BT_9	1023		0.67	0.73
BT_10	2047		2.04	3.00
BT_11	4095		8.5	11.8
ABT	4103	1	8.3	7.8
		2	6.8	7.8
		3	10.05	7.8
ABT2	8207	1	58.2	13.88
		2	63.3	13.88

Table 4.2: The Performance of a Representative Set of Scenarios for Causality Inference

Although we have fewer inference queries (67) and the approaches are different, for comparison, we also plot the checking approaches with  $ILP_{why}$  in Figure 4.4.  $ILP_{why}$  answered **63** out of **67** queries. In comparison, it was, as expected, slower than the checking approaches. Still, it scaled to large and complex queries. For instance, with basic tree models of 4000 variables ( $BT_{11}$ ,  $ABT$ ), it took  $8s$  and scaled to 8000 variable  $ABT2$  within  $63s$ .

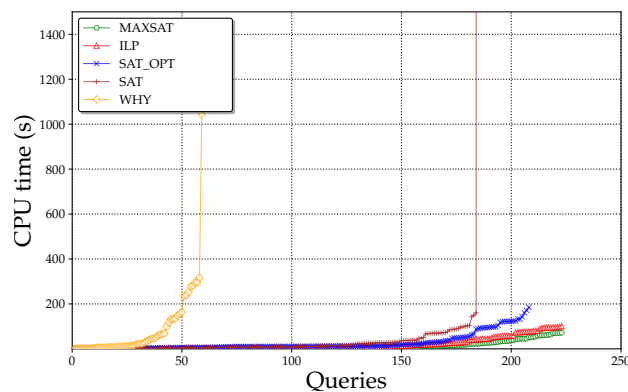


Figure 4.4: Cactus Plot including  $ILP_{why}$  Execution Time and Memory Results on the Larger Models.

However, it slowed down with larger models with complex semantics, i.e.,  $SMK$  different

variants. For instance,  $SMK(5500)$  took 280s, while  $SMK(6600)$  jumped to 1400s. The slowdown is related to the memory allocation because the program, finally, ran out of memory with queries on  $SMK(7150)$ . Given sufficient memory, we think  $ILP_{why}$  computes inference for even larger models.

In summary, we argue that the three approaches efficiently automates actual causality reasoning over binary models. Our  $MaxSAT$  encoding performs well for purposes of causality checking and semi-inference. Although slower,  $ILP_{why}$  is also efficient and scalable for purposes of inference. That said, we, of course, acknowledge that our results are bound to the data-set that we gathered and used.

## 4.5 Proofs

In this section, we present proof sketches of the theorems in this chapter.

**Theorem 4.1.** *The generated optimization problem (ILP program or  $G_{max}$ ) is feasible iff AC3 holds for  $\vec{X}$  or a non-empty subset of  $\vec{X}$ .*

*Proof.* The proof follows from the remark that  $G$ , which both formulations are based on, is a generalization of  $F$  and is satisfiable if the context  $\vec{U}$  makes  $\varphi$  evaluate to its negation, given that the semantics of the model is expressed using the constraints added, and the cause set,  $\vec{X}$ , is not constrained to have other values ( $\vec{x}'$ ). We show this in the following:

1. Recall  $G := \neg\varphi \wedge \bigwedge_{i=1\dots\ell} f(U_i = u_i) \wedge \bigwedge_{i=1\dots m, \exists j \bullet X_j = V_i} (V_i \leftrightarrow F_{V_i} \vee f(V_i = v_i)) \wedge \bigwedge_{i=1\dots\ell} (X_i \vee \neg X_i)$ . Rewrite the formula to abstract the first part as,  $G := G_{base} \wedge \bigwedge_{i=1\dots n} (X_i \vee \neg X_i)$ .
2. Note how  $X_{(n)}^{\vec{x}}$  is added to  $G$  as  $(X_1 \vee \neg X_1) \wedge (X_2 \vee \neg X_2) \dots (X_n \vee \neg X_n)$ . Re-write this big conjunction to its equivalent disjunctive normal form (DNF) i.e.,  $(\neg X_1 \wedge \neg X_2 \dots \wedge \neg X_n) \vee (\neg X_1 \wedge \neg X_2 \dots \wedge X_n) \dots \vee (X_1 \wedge X_2 \dots \wedge X_n)$ . Assume wlog that all the actual values of  $X_{(n)}^{\vec{x}}$  were *true*, hence to check them in AC2 we need to have their values negated, i.e.,  $\neg X_i$ . Looking at the DNF, we have  $2^n$  clauses that list all the possible cases of negating or fixing the elements in  $\vec{X}$ . Then, we partition  $G$  according to the clauses, i.e.,  $G := G_1 \vee G_2 \dots G_{2^n}$ , where  $G_1 := G_{base} \wedge (\neg X_1 \wedge \neg X_2 \dots \wedge \neg X_n)$ .  $G_1$ , is formula  $F$  for  $\vec{X}$ , which according to Theorem 3.3 is satisfiable iff AC2 holds for  $\vec{X}$ .  $G$  holds if any  $G_i$  hold.
3. Generally  $G_i$ , fixes some (possibly none) elements to their original evaluation ( $X_i$ ) and negates some, possibly none ( $G_{2^n}$ ), other elements ( $\neg X_i$ ).  $G_i$  is an  $F$  formula (from Algorithm 2) for the negated variables, in a clause, as  $\vec{X}$  but with some special fixed variables that are added to  $\vec{W}$ . Based on Theorem 3.3  $G_i$  is satisfiable iff AC2 for a the subset of the causes given that the other part (fixed) of the cause is in  $\vec{W}$ , holds. Thus  $G$  is satisfiable if AC2 holds for any subset of it.



4. The transformation from  $G$  to an ILP program  $P$  is proved to be correct [129]. This means that satisfiability of  $G$  entails feasibility of  $P$ .  $P$  is then feasible if AC2 holds for the A.) whole  $\vec{X}$ , B.) parts of  $\vec{X}$ , or C.) an empty set of causes. Adding the distance constraint to  $P$  results in a new program  $P'$ . Recall that the distance will be the count of variables  $\in \vec{X}$  that have a value  $\vec{x}'$  in the solution of  $P'$ . The distance should be greater than 0, i.e., case C is treated. By its nature, the ILP solver will pick the solution set that makes the distance the least. Hence, if  $\vec{X}$  is minimal in fulfilling AC2 it will be picked, i.e., case A. Similarly case B is treated.
5. Similarly, since  $G$  forms the hard clauses of the MaxSAT  $G_{max}$ , then  $G_{max}$  is satisfiable if  $G$  is satisfiable.  $G_{max}$  is then satisfiable if AC2 holds for the A.) whole  $\vec{X}$ , B.) parts of  $\vec{X}$ , or C.) an empty set of causes. We get rid of (C) by adding  $K$  clause as a hard clause. As such,  $G_{max}$  is satisfiable only when for cases A and B.

□

**Theorem 4.2.** *Formula  $G^*$  is satisfiable iff  $\exists \vec{X} = \vec{x}$  such that AC2 holds for  $\vec{X}$*

*Proof.* The proof follows from the correspondence between formula  $G^*$  and  $F$ . The proof consists of two parts.

**Part 1.**  $SAT(G^*) \implies \exists \vec{X}$  such that AC2 holds for  $\vec{X}$

We show this by contradiction. Assume that  $G^*$  is satisfiable and  $\nexists \vec{X}$  such that AC2 holds.

1.  $G^* := \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i) \wedge \bigwedge_{i=1\dots m} \left( \left( (V_i \leftrightarrow F_{V_i}) \wedge C_i^1 \right) \vee \left( \neg(V_i \leftrightarrow F_{V_i}) \wedge \neg C_i^1 \right) \right) \wedge \left( (V_{orig} \wedge C_i^2) \vee (\neg V_{orig} \wedge \neg C_i^2) \right)$
2. For readability let us call  $(V_i \leftrightarrow F_{V_i})$  as  $e_i$ . Since  $G^*$  is satisfiable, every conjunction  $CON_i$ :  $\left( (e_i \wedge C_i^1) \vee (\neg e_i \wedge \neg C_i^1) \right) \wedge \left( (V_{orig} \wedge C_i^2) \vee (\neg V_{orig} \wedge \neg C_i^2) \right)$  holds. It is a matter of natural deduction to show that when  $CON_i$  holds with values  $(C_i^1 \vee C_i^2)$  (01, 10, 11) it implies  $e_i \vee V_{orig}$ , that is proving the following proposition  $\left( (e_i \wedge C_i^1) \vee (\neg e_i \wedge \neg C_i^1) \right) \wedge \left( (V_{orig} \wedge C_i^2) \vee (\neg V_{orig} \wedge \neg C_i^2) \right) \implies e_i \vee V_{orig}$ . The only remaining case of  $(C_i^1 C_i^2)$  is 00. This case, in turn, implies  $\neg e_i \wedge \neg V_{orig}$ . That is the proposition:  $\left( (e_i \wedge C_i^1) \vee (\neg e_i \wedge \neg C_i^1) \right) \wedge \left( (V_{orig} \wedge C_i^2) \vee (\neg V_{orig} \wedge \neg C_i^2) \right) \wedge (\neg C_i^1 \wedge \neg C_i^2) \implies \neg e_i \wedge \neg V_{orig}$  can be proved by deduction. Note that there is no guarantee that the  $(C_i^1 C_i^2) = 00$  case always exists (this case is handled by the algorithm).
3. For each variable in  $M$ , adding the implications from above to a formula  $Y = \neg\varphi \wedge \bigwedge_{i=1\dots n} f(U_i = u_i)$  would result in an  $F$  formula (from Algorithm 2) for some  $\vec{X}$ .  $Y$  is satisfiable which by Theorem 3.3 makes AC2 holds for  $\vec{X}$ . This contradicts with the first assumption.

**Part 2.**  $\exists \vec{X}$  such that AC2 holds for  $\vec{X} \implies \text{SAT}(G^*)$

We show this by contradiction, as well. Assume  $\exists \vec{X}$  such that AC2 holds, and that  $G^*$  is un-satisfiable. Since AC2 holds then there exists a satisfiable  $F$  as constructed in Algorithm 2. Similar to the first part of the proof, since each variable  $V_i$  has a satisfiable conjunction in  $F$ , it implies a conjunction in  $G^*$  (the inverse of the implications in the first part (without  $C_i^1, C_i^2$ )). With that  $G^*$  is satisfiable. This contradiction proves the second part of the theorem.  $\square$

## 4.6 Summary

To conclude this chapter, we briefly summarize the different checking, semi-inference, and inference approaches. In Table 4.3, we show specific properties of each approach, namely: *encoding* a description of the approach, e.g., SAT or UNSAT, *size* an estimation of the size of the encoding in terms of the number of variables and clauses<sup>3</sup> (constraints) which does not necessarily reflect the difficulty of a problem, *complexity* lists the complexity<sup>4</sup> of the encoding but still indicates a comparison aspect, and *type* which refers to the ability of the approach, i.e., *checking*, *semi-inference*, or *inference*.

	Encoding	Size		Complexity	Type
		Variables	Clauses		
SAT	1 SAT, 1 ALL-SAT	$n, n$	$n, n$	NP-complete, NP-complete	checking
$\text{SAT}_{OPT}$	1 SAT, 1 UNSAT (or one combined SAT)	$n, n$	$n, n + 3 \times  \vec{X} $	NP-complete, Co-NP	checking
ILP	1 program	$n$ binary variables + 1 decimal variable	$n -  \vec{X}  + 1$	NP-complete	semi-inference
MaxSAT	1 formula	$n$	$n + 3 \times  \vec{X} $	$\Delta_2^P$	semi-inference
$\text{ILP}_{why}$	1 program	$4n$ binary variables + 4 decimal variable	$4n + 4$	NP-complete	inference

Table 4.3: Summary of the Approaches;  $n$  is the number of all variables in a causal model, and  $|\vec{X}|$  is the size of the cause set.

Using SAT, we encode HP in *two* formulas (for  $\text{SAT}_{OPT}$ , they can be combined into one formula), the second formula either requires solving ALL-SAT or UN-SAT problems. None of which is necessarily harder (the question of NP = co-NP is open) than the other, but as we saw in the evaluation of Chapter 3, ALL-SAT has a severe drawback on the memory consumption in some situations. That said, we have to note that  $\text{SAT}_{OPT}$  has a larger encoding (because of the added clauses), which we estimate to be linear in the size of the cause. Using ILP, we encode the full check of HP into one program that comprises  $n + 1$  variables (one variable more than each SAT formula), where  $n$  is the number of exogenous

<sup>3</sup>The number of clauses shown here does not resemble the CNF form since it is not easy to generalize such number. However, we think showing the number on the level of  $F$  or  $G$  formulas shown in this thesis is sufficient for the comparison.

<sup>4</sup>Obviously, the encoding does not change the complexity of the problem; this column shows the number and complexity of problems an approach requires

and endogenous variables in a model. The number of clauses in the case of ILP benefits from removing variables in  $\vec{X}$ . Hence, fewer constraints are added to the ILP ( $n - |\vec{X}|$ ), but we finally add one constraint to calculate the distance, as shown in Section 4.2.2. The *type* column emphasizes that the ILP result can be exploited for more valuable information, i.e., inferring a minimal  $\vec{X}$ . Similarly, the MaxSAT approach produces only one formula to compute causality. The formula contains the same number of variables in the model. The MaxSAT formula includes a clause for each variable in the model and extra clauses that are linear in the size of the cause set. Solving a weighted MaxSAT problem is  $\Delta_2^P$ , which is the class of problems that can be solved by a linear number of calls to a SAT solver. Lastly, the ILP<sub>why</sub> approach results in one ILP program to present an inference query. Unlike previous methods, ILP<sub>why</sub> expands the program with three control variables for each endogenous variable in the model. Also, the sum of these control variables is optimized; thus, we have four decimal variables in the program. The number of constraints and clauses increases in this approach, as well. Specifically, for each variable, we have four clauses to represent the contrarians on them. Finally, the result obtained by this approach is an actual cause, according to HP, without any hypothesis on  $\vec{X}$ .

According to HP, a set of events ( $\vec{X}$ ) cause an effect ( $\varphi$ ) if (1) both actually happen; (2) changing some values of  $\vec{X}$  while fixing a set  $\vec{W}$  of the remaining variables at their original value leads to  $\varphi$  not happening; and (3)  $\vec{X}$  is minimal. The complexity of the general problem has been established elsewhere. We show that when restricting to binary models, the problem of checking or inferring causality can effectively and efficiently be solved as an optimization problem. The problem is not trivial because intuitively, we need to enumerate all sets  $\vec{W}$  from condition (2) and need to check minimality for condition (3). We show how to formulate both properties as an optimization problem instead which immediately gives rise to using a solver to determine if a cause satisfies all conditions, or find one that does. For that, we define an objective function that encodes the distance between cause values in the actual and counterfactual worlds. If we now manage to optimize the problem with a smaller cause, then we know that it satisfies condition (2) but is not minimal. With an additional objective to quantify responsibility, we also formulate inference as an optimization problem. Using models with 8000 variables, which we deem realistic and necessary for automatically inferred causal models, we show that our approaches answer checking queries in seconds, and inference queries in minutes. In the next chapter, we extend our work to non-binary models.



# 5 Actual Causality Checking Beyond Binary Models

*This chapter presents a generalization of the concepts in the previous chapters, and proposes a method to answer checking causal queries of numeric models.*

## 5.1 Introduction

So far, we have limited ourselves to causal reasoning with binary models. To the best of our knowledge, binary causal models constitute almost all the models used in the actual causality literature. For instance, most of the examples used to illustrate the HP definitions apply binary models. Further, all the applications of HP in the literature are limited to binary causal models, such as causality in databases [134], and model checking [15]. On the one hand, this limitation may be attributed to the complexity of actual causality computation, in which limiting the calculation to binary values gives an advantage. On the other hand, it could be the case that we limit ourselves to binary models, simply because non-binary causal models are not emerging from applications. In other words, the domains that use actual causality so far, seem to require binary models only. Even when non-binary variables, with a finite set of values, are used within examples or applications, it is sufficient to binarize them.

In this chapter, we extend the foundation that we built in Chapter 3 and Chapter 4 to non-binary models. Specifically, we treat numeric causal models; models that contain integer or continuous variables, and arithmetic functions. Given the larger search space, the computation within such models is more complex than the binary models. However, the approach presented in this chapter demonstrates promising results from the efficiency perspective. From an application point of view, we think that numeric causal models will arise in various domains in the near future. One example is explainable artificial intelligence (xAI) [141, 143]. Statistical models learned by machine learning algorithms, or abstractions of such models, are increasingly requiring explanation [141, 143]. These models have various possible representations, such as a set of linear and non-linear equations, decision trees, or tabular listing of features and attributes. Some of these representations can be easily binarized and presented in propositional logic, for instance, decision trees. However, in some cases, we require the numeric nature of the variable when computing a counterfactual world, and hence binarizing the model is not suitable.

In this chapter, we focus on the automated actual causality reasoning with non-binary causal models. To the best of our knowledge, our proposed solution is the first to offer automated reasoning for non-binary actual causality. We show how the concepts introduced and evolved in our previous approaches can be generalized and abstracted away from supporting only binary models. That said, we acknowledge that we only present a computational approach without thorough real-world applications. As motivated above, we believe such applications to arise in the future.

## 5.2 Approach

In this section, we present our non-trivial solution towards checking actual causality (according to HP) in non-binary models. In Section 5.2.1, we elicit the requirements of such a solution and limit the focus to *integer* models with *linear* equations. Later, in Section 5.2.2, we discuss the concepts and the building blocks that are used to realize the requirements. In Section 5.2.3, we put together an algorithm that achieves our aim; and discuss extensions to different types like continuous variables or non-linear relations.

### 5.2.1 Requirements For Causality Checking

Recall that we aim to answer a causal *query* in the form of *Is  $\vec{X}$  a cause of  $\varphi$ ?* Using HP, we achieve this by verifying if a hypothesized *cause* (set of variables) fulfills the three conditions (AC1-AC3) from Definition 2.4, given a causal situation  $(M, \vec{U})$ .<sup>1</sup> Informally speaking, the conditions (namely, AC2 and AC3) encapsulate the counterfactual reasoning. We need to think of two worlds (variable assignments): the *original* world with all the variables and their values already known to us (specific context and acyclic equations), and the counterfactual one in which the *cause* and *effect* take on different (than original) values. Two facts further complicate the search for a counterfactual world. First, an arbitrary set  $\vec{W}$  of endogenous variables keep their values (disregarding their equations). Second, no (non-empty) subset of the cause variables is sufficient for constructing such a counterfactual world. These challenges should be treated while preserving the semantics of the model.

Accordingly, we state the requirements of our algorithm. Computing causality must ensure: **a.)** embedding the context of the query, which entails setting the values of the exogenous variables  $\vec{U}$  to their values **b.)** preserving the semantics of the causal model (represented by the equations) yet allowing some unknown set of variables to keep their original values **c.)** un-conditioning the values of the cause variables by any equations; yet “urge” a cause variable to retain its original value **d.)** forcing the effect variables to fulfill the first part of **(b)**, yet ensuring that their values evaluate differently than the original values.

Similar to our MaxSAT and ILP approaches, we are looking at the problem of computing actual causality as a search problem. Since ILP solves integer optimization problems,

---

<sup>1</sup>The first two paragraphs of Section 5.2.1 give a quick recap of actual causality computation.

we formulate a causal query as an integer program, solve it using efficient solvers [142], and interpret the results from the causal perspective. As a direct consequence of using ILP solvers, we show in Section 5.3 how our approach is both efficient and scalable in the size of the model and the cause. Although we are focusing on integer variables and linear equations in this chapter, the approach points and addresses the challenges of actual causality computation in a general way. As discussed in Section 5.2.3, our approach can be extended to continuous variables and non-linear equations.

An optimization problem formulated as a linear program contains three ingredients: *variables*, *constraints*, and an *objective function*. The *variables* include, in addition to the endogenous and exogenous variables of the model, a set of auxiliary variables that are added to realize the requirements formalized earlier in this section. The *constraints* are our technique to preserve the semantics of the equations and to force the solver to respect the counterfactual reasoning. Lastly, the *objective function* is our way to ensure that a cause is minimal (according to AC3); or what a minimal cause is in case the hypothesized cause is not (semi-inference). We elaborate on each ingredient in correspondence to each requirement in the next section.

## 5.2.2 Building blocks

Let us now consider each group of variables in a causal query, and see how to represent it in our program solely. We start with the *context* variables, i.e., the exogenous variables  $\vec{U}$ . Unlike endogenous variables, exogenous variables are constrained with linear equality to their original value, that is, the context value that led to a certain effect. For example,  $A_{exo} = 20$ .

For the endogenous variables ( $\vec{V}$ ), we have a more complicated treatment. First, let us classify these variables into three groups: 1- cause variables  $\vec{X}$ , 2- effect variables  $\vec{Y}$ , 3- *normal* variables which are neither in (1) or (2). We start with the *normal* variables (group 3). In a counterfactual world, these variables would either evaluate to new values according to their equations or would retain their original values (without following their equation), hence, be a member of the contingency set  $\vec{W}$ . Following our binary approach, we realize this variance of *normal* variables using a *disjunction* of the variable equivalence to its equations, i.e.,  $V_i = \mathcal{F}_i$  (for the defining equation  $\mathcal{F}_i$ ), re-written as  $EQ1 : \mathcal{F}_i - V_i = 0$ , and the variable's equality to its original value i.e.,  $V_i = v_i$ , re-written as  $EQ2 : V_i - v_i = 0$ . Since *all* constraints in a linear program must hold, modeling such *disjunctive constraints on equalities* is not straight forward.<sup>2</sup> However, we can leverage some techniques proposed in the literature to achieve our goal at the expense of adding auxiliary binary and integer variables [19, 196].

Essentially, we transform each equality  $EQ1$  to new  $EQ1'$  by adding a so-called free slack variable  $S_i$ , i.e.,  $EQ1' : \mathcal{F}_i - V_i + S_i = 0$  [17, 71, 19, 196]. Using another new *binary* variable

<sup>2</sup>The formulation in binary models is simpler because the linear constraints were created on the CNF level of the formula

(indicator  $N_i$ ), we constrain  $S_i$  to the following bounds:  $-M(1 - N_i) \leq S_i \leq M(1 - N_i)$ , where  $M$  is a sufficiently large upper bound to the equality.<sup>3</sup> Now, the idea is that  $EQ1'$  will *always* hold; however we are only interested in the case where  $S_i$  equals 0, i.e.,  $N_i$  is 1 then  $EQ1$  holds. Suppose we do the same for  $EQ2$  with a new slack variable  $S_i^*$ , and get  $EQ2' : V_i - v_i + S_i^* = 0$  where  $-M(1 - N_i^*) \leq S_i^* \leq M(1 - N_i^*)$ . We have two constraints that are always holding; however, we need to connect them disjunctively and make sure that at least one of the two indicators  $N_i$  or  $N_i^*$  is equal to 1. Keeping in mind that the indicators are binary, we ensure the disjunction by adding the constraint  $N_i + N_i^* \geq 1$ . With this model, we express each normal variable and establish a way to infer a contingency set. Table 5.1 shows the truth table of the indicators for one variable, and how to interpret each case. We can represent the two indicators with one variable. However, we want to keep the possibility of finding the variables that evaluate according to their equations but still evaluate to the same values, i.e., the last case in Table 5.1.

$N_i$	$N_i^*$	Interpretation
0	0	Program is infeasible
0	1	Variable is in $\vec{W}$
1	0	Normal variable ( $\notin \vec{W}$ )
1	1	Normal variable ( $\notin \vec{W}$ )

Table 5.1: Disjunctive Constraints Cases

The second group of endogenous variables are cause variables  $\vec{X}$ . According to requirement (c) from Section 5.2.1, contrary to normal variables, cause variables are not modeled using their equations. Rather, such variables are allowed to take on any value ( $x'_i$ ) when the program is solved. However, if a cause variable ( $X_i$ ) kept its original value ( $x_i$ ), then it is not a necessary part of the cause set (i.e., minimality is violated). To ensure that our program is solved in a way that filters out the non-necessary cause variables, we utilize the *objective function*. Following our approach in Chapter 4, we count the number of cause variables that have different values in the solved program than their original values. Recall that we call this count a *distance* between the original and the counterfactual world. The objective of the program is, then, to minimize the distance. An optimal solution of the program can be interpreted as the counterfactual assignment with the shortest distance, i.e., the minimal set of altered causes. Formally, Equation 5.1 models the distance for integers.

$$distance = \sum_{i=1}^{\ell} Min(1, Abs(x'_i - x_i)) \quad (5.1)$$

$x_i$  is the value of the cause variable  $X_i$  in the original (actual) world, whereas  $x'_i$  is the value of the same variable in the counterfactual world (returned by the solver). We take

---

<sup>3</sup>Using  $M$  is a known technique in optimization paradigms called the Big-M formulation. It is used in various cases [19, 196], but may impact the performance; hence, it should be chosen as tight as possible



the absolute difference of these values for each cause variable ( $Abs(x'_i - x_i)$ ); obviously, the difference is  $\geq 0$ . Regardless of the difference amount, we count it as 1 by using a minimum function. The minimum function ensures that differences greater than one are still counted as 1. It is worth noting that the real value of the difference could be of use in some applications that consider the normality of the counterfactual world [73, 77], for example. Lastly, Equation 5.1 is integer specific, since it assumes that 1 is the least absolute difference for a necessary variable. It can be extended to account for continuous values by considering a realistic minimum of the difference, e.g., 0.1 and amplify it by multiplication ( $10 \times Abs(x'_i - x_i)$ ). The difference term can be variable-specific in cases of mixed (integer and continuous) models.

The last group of the endogenous variables is the *effect* variables. Recall that, according to Definition 2.4, the effect is a combination of endogenous variables in the form of a mathematical expression, e.g.,  $Z - K = 1$ . Based on requirement (d), we model the effect in two ways. First, we need to make sure that each variable appearing in that expression evaluates according to its equation. For that, we use the first *equality* for the *normal* variables (EQ1). We do not add the slack variable to the equation because it is not needed. Second, we have to ensure that the effect is not happening in the counterfactual world, i.e., the mathematical expression does not hold given the assignments of the solved program. In our previous example, we need  $Z - K$  to not equal 1 re-written as  $Z - K - 1 \neq 0$ . Not-equal comparisons are not supported natively in linear programming; hence, we craft the effect constraint. Essentially, we re-write the linear mathematical expression to be equal to 0 on the right hand, and then we constrain the absolute value of the left-hand term (e.g.,  $Abs(Z - K - 1)$ ) to be greater than 0. This way, we ensure that the effect is not occurring anymore, e.g., the difference between  $Z$  and  $K$  is not 1. The same applies if we are considering more simple effect forms such as  $Z = 5$ . From a practical perspective, we observed that another form of effect expressions often arises, especially in the domain of  $\times$ AI. This format is referred to as contrastive queries, i.e., why not questions [141, 143]. Our approach supports contrastive effect expressions such as *why Z is not 4?* written as  $Z - 4 \neq 0$ . Such effect expressions are negated and appended as a constraint that looks like  $Z - 4 = 0$ .

In this section, we presented a generalization of the binary concepts we contributed in Chapter 3 and Chapter 4. In the course of that, we classified the variables in a causal query into three different classes. For each category, we discussed the realization of its requirements; we identified all the problems that need to be addressed by our program and showed how to model them in a linear program. Concepts like disjunctive constraints, distance function, and effect modeling will all be used as building blocks in the algorithm presented next.

### 5.2.3 Algorithm

In this section, we formalize our approach in Algorithm 6; the soundness of the algorithm is a result of our earlier results in Chapter 3 and Chapter 4. It depicts how the constraints are derived. For readability, we omit the variables declaration inside the linear program.

The input to this algorithm is the set of structural equations represented as  $\langle F \rangle$ , the context represented as a vector of variables  $\vec{U}$  and their values  $\vec{u}$ , similarly a hypothesized set of cause variables  $\vec{X}$  and their values. The effect is presented as a mathematical expression  $\varphi$ , and lastly, the set of endogenous variables, and their original evaluation is given as  $\vec{V}$ .

As described in Section 5.2.2, the exogenous variables are added as equality constraints in the loop that starts at Line 2. More importantly, the endogenous variables, excluding the cause variables, are then added in Line 6 - Line 18. The first part of the loop (Line 8 - Line 13) adds the *disjunctive constraints* for the normal variables. Each constraint is defined and then appended to the list of constraints (*cons*). Note that we add the slack and indicator variables with the same naming scheme we presented earlier. That is,  $S_i$ , and  $S_i^*$  are the slack variables corresponding to a variable  $V_i$ , while  $N_i$  and  $N_i^*$  are the indicator variables corresponding to the same variable. Equations of the effect variables are also added within the for loop but without any auxiliary variables as shown in Line 15. The second part of treating the effect  $\varphi$  is shown in Line 19; for simplicity, we omitted the contrastive format of the effect. To wrap-up the construction of the program, Line 21, and Line 22 show the creation of the distance according to Equation 5.1, and the formalization of the objective.

After constructing the linear program, we solve it using off-the-shelf solvers such as Gurobi [71]. Through Line 23 to Line 25, we interpret the results. Given that AC1 already holds (can be checked in polynomial time), a feasible program with an optimal solution means that the hypothesized cause  $\vec{X}$  or a non-empty subset of it is an actual cause of  $\varphi$ , according to HP. If the objective value returned by the solver equals the size of the cause set, then  $\vec{X}$  is a minimal cause (both AC2 and AC3 hold). Otherwise, a minimal cause of size equal to the objective value can be inferred in Line 24 (AC3 is violated). In all cases, the contingency set  $\vec{W}$  is inferred in Line 25. An infeasible program entails that neither the cause set nor its subsets is an actual cause (AC2 is violated). However, the infeasibility of the program is constrained to the bounds of the variables that we defined. In other words, if there exists a solution to the program beyond the defined limits of the variables, then the program would not be solved. In our implementation, we allow the modeler to configure the bounds of the variables. The clear trade-off here is between the potential impact on performance and the effort to run the algorithm potentially a couple of times.

Multiple optimal solutions can be found for the same problem. In our case, this entails that different subsets of the cause are sufficient to fulfill the HP conditions. Currently, we do not employ any measure to force the solver to compare these causes; however, as we saw in Section 4.3, one way to compare causes would be to use the *responsibility* metric proposed in [33]. For the current version, our implementation stores all the solutions but returns only the first one. Lastly, we conclude with a brief discussion on directions to extend this algorithm. A useful extension of our algorithm would be towards causality *inference*, rather than *checking*. With inference, the hypothesized cause set is not required any more. Using the same building blocks, the inference can be achieved by extending the disjunctive constraints with a possibility to allow arbitrary values, and have the solver to minimize the number of the variables taking on arbitrary values. Another extension we consider

**Algorithm 6** Actual Causality check for Integer Models

---

**Input:** equations  $\langle F \rangle$ , context  $\langle U_1, \dots, U_n \rangle = \langle u_1, \dots, u_n \rangle$ , cause  $\langle X_1, \dots, X_\ell \rangle = \langle x_1, \dots, x_\ell \rangle$ , effect expression  $\varphi$ , evaluation  $\langle V_1, \dots, V_m \rangle = \langle v_1, \dots, v_m \rangle$

- 1: **function** CHECKCAUSE( $\langle F \rangle, \vec{U} = \vec{u}, \vec{X} = \vec{x}, \vec{V} = \vec{v}, \varphi$ )
- 2:   **for all**  $U_i \in \langle \vec{U} \rangle$  **do**
- 3:      $con1 \leftarrow (U_i = u_i)$
- 4:      $cons.add(con1)$
- 5:   **end for**
- 6:   **for all**  $V_i \in \langle V \rangle \wedge \exists j \bullet X_j = V_i$  **do**
- 7:     **if**  $V_i \notin \varphi$  **then**
- 8:        $con1 \leftarrow (F_{V_i} - V_i + S_i = 0)$
- 9:        $con2 \leftarrow -M(1 - N_i) \leq S_i \leq M(1 - N_i)$
- 10:        $con3 \leftarrow (V_i - v_i + S_i^* = 0)$
- 11:        $con4 \leftarrow -M(1 - N_i^*) \leq S_i^* \leq M(1 - N_i^*)$
- 12:        $con5 \leftarrow N_i + N_i^* \geq 1$
- 13:        $cons.addAll(con1, con2, con3, con4, con5)$
- 14:     **else**
- 15:        $con1 \leftarrow (F_{V_i} - V_i = 0)$
- 16:        $cons.addAll(con1)$
- 17:     **end if**
- 18:   **end for**
- 19:    $con_{effect} \leftarrow (Abs(\varphi) \geq 1)$
- 20:    $cons.addAll(con_{effect})$
- 21:    $distance = \sum_{i=1}^{\ell} Min(1, Abs(X_i - x_i))$
- 22:    $objective = Minimize\ distance\ s.t.\ 1 \leq distance \leq \ell$
- 23:   **if**  $\langle V_1 = v'_1 \dots V_m = v'_m \rangle = solve(objective, cons)$  **then**
- 24:      $\vec{X}' := \langle X'_1 \dots X'_d \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \Rightarrow X'_i \neq X'_j) \wedge (X'_i = V_j \Leftrightarrow v'_j \neq v_j)$
- 25:      $\vec{W} := \langle W_1 \dots W_s \rangle$  s.t.  $\forall i \forall j \bullet (i \neq j \Rightarrow W_i \neq W_j) \wedge (W_i = V_j \Leftrightarrow v'_j = v_j)$
- 26:     **return**  $\vec{X}', \vec{W}$
- 27:   **else return** *infeasible*
- 28:   **end if**
- 29: **end function**

---

useful is the support of non-linear equations. There are established approaches to linearize such functions using piecewise functions [19]. Piecewise functions can then be expressed as constraints in the program.

### 5.2.4 Example

In this example, we consider a model trained to predict the price of a house expressed as a median value of owner-occupied homes in USD 1000's. The data used to train the model is known as the Boston Housing Dataset.<sup>4</sup> The data included the following features.

- **crim** - per capita crime rate by town
- **zn** - proportion of residential land zoned for lots over 25,000 sq.ft
- **indus** - the proportion of non-retail business acres per town
- **chas** - Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- **nox** - nitric oxides concentration (parts per 10 million)
- **rm** - the average number of rooms per dwelling
- **age** - the proportion of owner-occupied units built prior to 1940
- **dis** - weighted distances to five Boston employment centers
- **rad** - index of accessibility to radial highways
- **tax** - full-value property-tax rate per USD 10,000
- **prratio**- pupil-teacher ratio by town
- **b**  $1000(B - 0.63)^2$ , where B is the proportion of blacks by town
- **lstat** - the percentage of lower status of the population
- **medv** - median value of owner-occupied homes in USD 1000's

Many researchers and practitioners have used this example to teach statistical modeling approaches.<sup>5</sup> We use one resource that used linear regression. Linear regression aims at using a set of variables to predict the value of another variable. Such approaches depend mainly on calculating correlation coefficients. Needless to say, correlation is not causation; however, recall that, in this example, we aim to explain why a model behaves as such rather than explaining the causal factors around house prices. In other words, we answer causal queries about the statistical model.

According to the model, the target variable, i.e., the *medv*, is mainly influenced by the *crim*, *rm*, *tax*, and *lstat*. Equation 5.2 shows the formula learned by the model.

$$medv = -0.071 * crim + 5.58 * rm - 0.007 * tax - 0.484 * lstat - 3.767 \quad (5.2)$$

---

<sup>4</sup><https://rpubs.com/ezrasote/housepricing>

<sup>5</sup>For instance, <http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html>

We use the above formula as part of a causal model. Each feature value is input to the model through an exogenous variable, e.g.,  $crim_{exo}$  is an exogenous variable that sets the value for crime rate  $crim$ . The set of features are the endogenous variables. Similar to Equation 5.2, we can use models to learn linear equations for each feature. For example, a model to predict the crime rate in towns is also shown in this tutorial.<sup>6</sup> The crime rate is estimated using other factors such as  $percent\_m$ : percentage of males aged 14-24,  $mean\_education$ : mean years of schooling,  $unemploy\_m39$ : unemployment rate of urban males aged 35-39,  $inequality$ : income inequality,  $prob\_prison$ : probability of imprisonment Time, and  $pol\_exp$ : police expenditure.

$$\begin{aligned} crime\_rate = & 8.7238 * (percent\_m) + 16.7212 * (mean\_education) + 7.0635 * (unemploy\_m39) \\ & + 7.0513 * (inequality) - 3863.7869 * (prob\_prison) + 6.7957 * (pol\_exp) \\ & - 4605.8496 \end{aligned} \tag{5.3}$$

This way, the model can be instantiated with equations, endogenous and exogenous variables. The context is set using the feature vector of one data point. Typically, the exciting part with constructing causal queries around data points is the explanation provided by the counterfactual computation [144, 198]. Computing counterfactuals (counterfactual worlds), in this context, is beneficial because it seems intuitive to the data subjects [144, 198]. However, picking one *useful* counterfactual may require additional knowledge that we do not include in our approach. For the sake of illustrating our approach in action, we discuss it in this example. Recall that within a counterfactual computation, we describe a setting (variables and values) that would change the predictor's output. As such, the *explanation* of a prediction made by the model for a new data point will specify the changes (different values of variables) that produces a different outcome of the model. Such an explanation is one type of many other ways to provide explanations [141]. Note that our algorithm still minimizes the number of variables to change the output; however, we also gather the new values of these variables when solving our programs.

For our example, we consider one case with the following values:  $crim = 0.006$ ,  $rm = 6.575$ ,  $tax = 296$ ,  $lstat = 4.98$ . The predicted  $medv$  is 32.20. A house owner may ask *Why is the medv of my house equal to 32.202 and not 33.0?*<sup>7</sup> This translates to a contrastive effect formula:  $medv - 33.0! = 0$ ; asking contrastive queries (why not  $y!$ ) makes the solver indicate those changes in the values that produce  $y!$ . In other words, not any change in the input would cause the effect formula to hold.

All the example variables can change and alter the output; however, they have different effects according to their correlation coefficients. Some of these variables are insufficient to result in the required change (relative to their bounds: 0.0 – 100000); for instance, a checking query of the  $crim$  variable solely, e.g., is  $crim = 0.006$  a cause for  $medv! = 33.0$ ?

<sup>6</sup><https://rpubs.com/ezrasote/crimerate>

<sup>7</sup>can also be formulated as *Which feature values should be changed to increase medv to 33.0?*

is negatively answered.

We constructed different checking queries that included combinations of the four variables (*crim*, *rm*, *tax*, *lstat*) as a cause. We observed several counterfactuals reported by the solver; a change in *rm* was mainly a part of the setting, along with a change in either *tax* or *crim*. For instance, when solving a checking query (the corresponding program is shown in Figure 5.1), that includes all the variables (*crim*, *rm*, *tax*, *lstat*) as a cause, we get back the minimum number of variables to be changed and their values. The result was a specific change in the values of *rm* and *tax*. Specifically, one of the results of this query allowed us to state that "If the number of rooms of your house is 6.721 rather than 6.575 and the full-value property-tax rate is 298.394 rather than 296, you would have a *medv* of 33." The important factor here is *rm* since the increase in *tax* is only to ensure *medv* = 33.0 and not 33.01. Other settings were not beneficial, such as "If the number of rooms of your house is 6.748 rather than 6.575 and the crime rate is 2.364 rather than 0.6, you would have a *medv* of 33." While these settings may seem unrealistic in some cases, e.g., they suggest to increase the crime rate, they produce a difference in the prediction.

Although some factors cannot be adjusted by the owner, such as the town crime rate, other factors can be adapted, such as the number of rooms in a house. In this example, we are not only showing the solutions of some numerical equation, but we are also applying the counterfactual causality definition on a statistical example.

Unlike previous examples in this thesis, we are not interested in the causes of this example, but rather in the method of computing causes. The method, as we illustrated, describes counterfactual explanations of factors. A counterfactual world presents an alternative setting that results in a different output. In our implementation, we look for a setting that alters a minimal number of features; we could rather think of the minimal amount of changed values regardless of how many variables are changed; we could consider a "realistic" change, i.e., one that changes certain features with certain amounts. Because of this generality, we think this problem is a causality optimization problem; it searches for a counterfactual based on an objective. Further, as we have seen in this example, the solver returns multiple solutions representing different counterfactuals. We tackle this by minimizing the number of variables to change (as part of HP) and emphasizing contrastive queries.<sup>8</sup> However, both are insufficient. While multiple counterfactuals are useful to provide several explanations [198, 144], we argue that a refined notion of a counterfactual in this context benefits this application more. The approach in this chapter is a sound base to automate the computation of such a notion. This way, the judgment of a counterfactual will not depend on the size of the cause only, but on the amount of change brought in the variable values, or the normality of such change [77], which we leave to future work.

---

<sup>8</sup>In a counterfactual world, a contrastive query requires the predictor (model output) to evaluate to a specific value, which reduces the number of possible solutions

<i>min distance</i>	<b>s.t.</b>
$1.0 \times 10^9 IND1_{crim} + SLA1_{crim}$	$\leq 1.0 \times 10^9$
$-1.0 \times 10^9 IND1_{crim} + SLA1_{crim}$	$\geq -1.0 \times 10^9$
$1.0 \times 10^9 IND2_{crim} + SLA2_{crim}$	$\leq 1.0 \times 10^9$
$-1.0 \times 10^9 IND2_{crim} + SLA2_{crim}$	$\geq -1.0 \times 10^9$
$IND1_{crim} + IND2_{crim}$	$\geq 1$
$-rm + DEL_{rm}$	$= -6575$
$-tax + DEL_{tax}$	$= -296000$
$1.0 \times 10^9 IND1_{lstat} + SLA1_{lstat}$	$\leq 1.0 \times 10^9$
$-1.0 \times 10^9 IND1_{lstat} + SLA1_{lstat}$	$\geq -1.0 \times 10^9$
$1.0 \times 10^9 IND2_{lstat} + SLA2_{lstat}$	$\leq 1.0 \times 10^9$
$-1.0 \times 10^9 IND2_{lstat} + SLA2_{lstat}$	$\geq -1.0 \times 10^9$
$IND1_{lstat} + IND2_{lstat}$	$\geq 1$
$-MIN_{rm} - MIN_{tax} + res$	$= 0$
$lstat_{exo}$	$= 4980$
$rm_{exo}$	$= 6575$
$crim_{exo}$	$= 6$
$tax_{exo}$	$= 296000$
$crim + SLA2_{crim}$	$= 6$
$crim_{exo} - crim + SLA1_{crim}$	$= 0$
$lstat + SLA2_{lstat}$	$= 4980$
$lstat_{exo} - lstat + SLA1_{lstat}$	$= 0$
$-71crim + 5580rm - 7tax - 484lstat - medv$	$= 3676$
$medv - PHI$	$= 3.3 \times 10^7$
$PHI_{ABS}$	$= 0$
<i>Bounds</i>	
$-1.0 \times 10^8 \leq lstat_{exo}$	$\leq 1.0 \times 10^8$
$-1.0 \times 10^8 \leq lstat$	$\leq 1.0 \times 10^8$
$-1.0 \times 10^8 \leq SLA1_{lstat}$	$\leq 1.0 \times 10^8$
$-1.0 \times 10^8 \leq SLA2_{lstat}$	$\leq 1.0 \times 10^8$
<i>GeneralConstraints</i>	
$ABS_{rm}$	$= ABS(DEL_{rm})$
$MIN_{rm}$	$= MIN(ABS_{rm}, 1)$
$ABS_{tax}$	$= ABS(DEL_{tax})$
$MIN_{tax}$	$= MIN(ABS_{tax}, 1)$
$PHI_{ABS}$	$= ABS(PHI)$

Figure 5.1: A snippet of the program generated for the example. For readability, the bounds section is summarized, the variable declaration is omitted, and values are scaled by a factor of 1000.

## 5.3 Evaluation

As computational complexity is one of the main challenges facing the deployment of causal reasoning in practice, in this section, we evaluate the performance of our solution. We start by estimating the size of the linear programming model in Section 5.3.1. We then benchmark the performance of an implementation of our algorithm in Section 5.3.2.

### 5.3.1 Model Size

Essentially, our solution constructs a linear program  $P$  from a causal model  $M$ . Estimating the size of  $P$  (denoted  $|P|$ ) in relation to  $|M|$  is an indicator of the *expansion* brought by our approach, and a measure to compare different flavors of the approach. Consequently, trade-offs between some of the qualities can be thought of in light of the expansion size; for example, do we need to model indicators of disjunctive constraints using one or two variables (or more in the case of piecewise functions). That said, we acknowledge what some researchers have rightfully pointed out that the size of a program is not, necessarily, an indication of its difficulty [194]. We estimate the number of the program's variables and constraints in the following equations.

In Equation 5.4, we show the total number of variables corresponding to a causal check of the type is  $\vec{X}$  an actual cause of  $\varphi$  given a model  $M$ , and a context  $\vec{U}$ . All the endogenous and exogenous variables in  $M$  are represented in the linear program. For each endogenous variable that is neither in the cause set nor in the effect expression, we add *four* auxiliary variables—two slack variables, and *two* (binary) indicator variables. For each cause variable, we need *three* extra variables—one for the value difference, one for the absolute value calculation, and one for the minimization function. Lastly, three additional variables are required for the negation of the effect expression and the calculation of the distance.

$$\text{Number of Variables} = |M| + 4 * |\vec{V} \setminus (\vec{X} \cup \text{Var}(\varphi))| + 3 * |\vec{X}| + 3 \quad (5.4)$$

Similar to the number of variables, we present the number of constraints in Equation 5.5. We assume here that the model consists of a set of linear equations, i.e., the structural equation related to a variable can be represented in *one* linear inequality. If this was not the case, Equation 5.5 has to be scaled accordingly. To set the context, the program will include one constraint for each exogenous variable. However, the constraints to govern the endogenous variables (excluding the cause and effect variables) are expanded in the magnitude of *seven*. In other words, we need *seven* linear inequalities to govern the behavior of each endogenous variable ( $\notin \vec{X}$  and  $\notin \varphi$ ). *Four* constraints are required to control the boundaries of the slack variables, *one* constraint for the indicator variable, and *two* to represent each part of the disjunctive constraints. Similarly, we require *three* constraints for each cause variable. Namely, *one* to calculate the difference in the variable's value, *one* for the absolute value calculation, and *one* to calculate the minimization of the difference. For the variables in the effect, i.e.,  $\text{vars} \in \varphi$ , we need a linear constraint to represent the



formula of each of them. To negate the effect, we add *three* constraints. Lastly, we add one constraint to calculate the distance measure.

$$\text{Number of Constraints} = |\vec{U}| + 7 * |\vec{V} \setminus (\vec{X} \cup \text{Var}(\varphi))| + 3 * |\vec{X}| + 1 * |\varphi| + 4 \quad (5.5)$$

The approximate upper bound of the number of variables required to represent a non-binary causal query is *five* times the size of the causal model. The number of constraints governing these variables is bounded by an upper limit of *seven* times the number of model variables.<sup>9</sup> Although the expansion in variables and constraints is considerable, the majority of the auxiliary variables are constrained by simple inequalities. As we have seen in Section 5.2, those variables control disjunctive constraints, hold the difference between actual and solved values, and calculate absolute values. Thus, the expansion itself does not pose a significant challenge to the solvers. Other factors, such as the boundaries of these variables, or the choice of the Big M values, impact the solver’s performance. In the next section, we present the results of our experiments regarding the performance.

### 5.3.2 Performance

Within this experiment, we could not use the same set of models we used with the binary approaches. To evaluate our approach, we tested Algorithm 6 with 84 models. Gathering a set of numeric causal models with their equations is challenging. Unlike binary models, we have not found any causal models that use linear arithmetic equations to describe the relationship among causal factors. Thus, we reused 4 simple statistical models from the domain of machine learning. Specifically, we used the Boston housing model *BH* (13 variables)<sup>10</sup>, the crime rate prediction model *CR* (7 variables)<sup>11</sup>, a combined model of both *BH* – *CR* (20 variables), and diabetes risk model *DR*<sup>12</sup> (9 variables). All these models are small; thus, in addition, we randomly generated 80 models consisting of random linear equations.

The generated models contain between 256 and 2048 endogenous variables. Each exogenous variable in these models is randomly set to a value between 1 and 10. Each endogenous variable is either set to a corresponding exogenous variable or *described* using a random linear equation. The linear equation of an endogenous variable is a random weighted combination of its two *child* variables. The equation contains a randomly picked operation of addition or subtraction and two random coefficients for the operands. To control the upper bound value, the coefficients are chosen in the range 1 to 10.

<sup>9</sup>This is an over-approximation. The number is less than *seven* times the endogenous variables only, and the number of exogenous variables added on top.

<sup>10</sup><https://rpubs.com/ezrasote/housepricing>

<sup>11</sup><https://rpubs.com/ezrasote/crimerate>

<sup>12</sup>[https://rstudio-pubs-static.s3.amazonaws.com/346228\\_a62c6c91d5cf40869cd5aef7206826ae.html](https://rstudio-pubs-static.s3.amazonaws.com/346228_a62c6c91d5cf40869cd5aef7206826ae.html)

For each generated model, we picked random cause sets of different cardinalities. We have the following values for the size of the cause 1, 5, 10, 15. We configure two modes of effect expressions ( $\varphi$ ): the first is a normal equality effect, i.e.,  $Y = y$ , and the second is a contrastive expression of the form  $Y \neq y$ . Since the bounds of the variables and the values of BigM play a significant role in the feasibility of the program, we configured our benchmark with a list of settings for the lower, upper bound of each variable, and the value of BigM.

In summary, we automatically generated 20 numerical models of each of the following sizes: 256, 512, 1024, 2048. For bigger sizes, the experiment always exhausted the memory. Each model is evaluated with 4 sets of causes, 5 settings of bounds, two forms of effect expressions, one random context. Resulting in 40 queries for each model (800 queries for each size).

### Results

As we shall see in this section, the time to answer the queries over bigger models varied considerably according to multiple factors. For models with 256 variables, most of the queries finished in less than 1 second. If the cause set does not fulfill AC2, the answer was much faster (less than 0.2 second). However, the performance degraded drastically with some specific queries (4 out of 800). The execution time of these queries varied between 2.22 seconds and 134.23 seconds. The *four* queries were *contrastive* queries. This can be attributed to the fact that in a counterfactual world, a contrastive effect is constrained to a specific value, which is apparently harder for a solver than finding any value (other than  $x$ ). This is only a logical assumption in justifying this observation because it is not clear how modern ILP solvers treat equality and inequality constraints. The second commonality among these *four* queries is the size of their minimal cause. All of them semi-inferred a minimal cause of size 2 out of the hypothesized cause. In comparison to all the other queries which inferred a singleton cause, answering these queries was harder for the solver.

The same patterns were observed when analyzing the results of solving queries with models of size 512 variables. Queries that did not fulfill AC2 (infeasible ILP programs) were mostly answered in less than 0.2 seconds. Most of the remaining positive queries inferred a singleton cause from the hypothesized cause in a period between 0.3 seconds and 1.3 seconds. However, inferring a *two-variables* cause was more expensive in 10 queries. The fastest took 2 seconds, and the slowest took 7138 seconds. These queries were a mix of normal (5) and contrastive queries (5). In extreme cases, we interrupted the experiment after 24 hours of the solver trying to solve the program. These patterns repeated with 1024 and 2048 variables.

In summary, the results discussed in this chapter show the computational challenges facing causal reasoning with non-binary models. Our experiments show a limit to models of size around 2000 numerical variables (no bounds on their values). This limit is only attributed to our machines' memory capacity; thus, such a number can be extended with more memory. However, the execution time of the reasoning varies with numerical models

than with binary models. We have observed that semi-inference queries are efficiently answered in less than 2 seconds if the actual cause is a *singleton* or no minimal cause can be inferred.

On the other hand, actual causes of more than one variable take more time. The exact amount of solving time depends on the model and the variables' bounds. While the results presented in this evaluation may seem inconclusive, reasoning over non-binary causal model is effectively automated, and to some extent efficiently computed using the approach in this chapter.

## 5.4 Summary

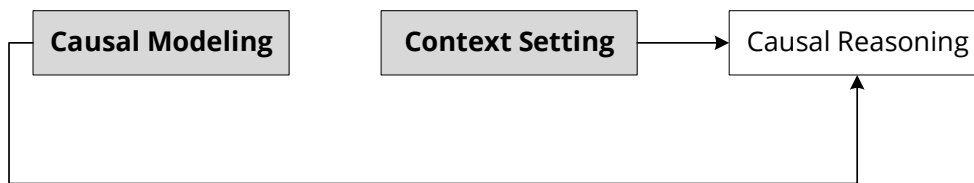
Since ILP can be used in domains other than binary optimization, the extension of the formulations from Chapter 3 and Chapter 4 to numerical models is natural. This chapter presents an approach that deals with the issues arising from such an extension. Specifically, we show a generalization of the *distance* calculation and a representation of the variables using concepts such as *disjunctive constraints*. While the direct applications of this approach are not clear, we focused on examples from the xAI domain. Thus, we treated the idea of *contrastive query* as a key requirement in our implementation and evaluation.

According to the considerable model expansion (as estimated in the evaluation) and the range of variables' bounds, the solver's performance is affected. We have observed a very efficient performance, regardless of model size, with negative queries (conditions are not fulfilled) or queries that can be answered with singleton causes. However, the performance degrades with queries, for which the answers consist of two or more cause variables. The degradation is more severe when the causal models are larger than 2000 variable. The judgment on the usefulness of such results is clearly related to the application. We think such limits may still be useful to explain statistical models, for example.



## Part III

# Domain-specific Causal Modeling and Contextualization





# 6 Causal Model Extraction from Attack Trees to Attribute Insider Attacks

*In addition to causality reasoning (which we tackled in the previous part), building valid causal models and contextualizing them is important. Modeling and contextualization are both generic problems where we cannot expect a general solution. Therefore, in this part of the thesis, we show by examples how these tasks can be operationalized in a specific context, i.e., insiders in microservice-based information systems. We provide two automated methods for model construction and show how our reasoning technology can be used with them. Similarly, contextualization, of course, is application-specific, so we provide one example to show how it can be tackled in practice to support our reasoning approaches. This chapter tackles issues around causal modeling in the context of malicious insiders. Parts of this chapter have previously appeared in publications [99, 95], co-authored by the author of this thesis.<sup>1</sup>*

## 6.1 Introduction

As you may recall, this thesis builds an architecture to tackle the different aspects of actual causality reasoning, as shown in Figure 6.1. In this chapter, we focus on *causal modeling*, which is a necessary step to enable causality reasoning. While building causal models is a generic need across many scientific domains [108, 157], solutions in practice are always context-specific (e.g., [116]). In a specific context, modeling appears possible, even in an automated way (e.g., [205]). This chapter provides an example of automated causal modeling; thus, showing the relevance of our earlier approaches to tackle large models.

Security is crucial in systems that deal with *sensitive customer assets*. Adversaries are constantly trying to compromise the integrity, confidentiality, or availability of such assets. These attempts are carried out by *insiders* or *outsiders* of the system. In this chapter, we are chiefly interested in *insiders*, specifically *malicious insiders* such as a rogue employee. For instance, according to the Cyber Security Intelligence Index by IBM X-Force Research [169], insiders carried out 60% of all attacks in 2015. Insiders can, tamper with records in the

---

<sup>1</sup>Parts of this chapter are reprinted by permission from Springer Nature: Lecture Notes in Computer Science, vol 12419. Causal Model Extraction from Attack Trees to Attribute Malicious Insider Attacks, Ibrahim et al. (2020).

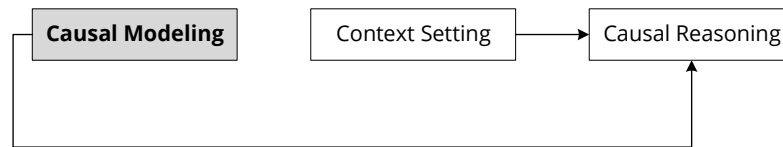


Figure 6.1: An Abstract Architecture of the Solution focusing on Causal Modeling.

database, leak or delete documents, which leads to *reputation damage, legal costs, and reimbursements* [101]. Reports show that *insiders* carry out the most significant, and costly attacks [169, 101]. Such attacks are likely to succeed, and their impact is significant [176]. In this context, preventive measures have a high likelihood of failing because insiders ought to have sufficient privileges for their jobs. They may abuse their privileges. The term “abuse” makes this problem especially hard due to the unpredictable nature of insiders and the necessity of their privileges. That said, insiders are mostly not malicious. Typically, there is a trust base between a company and its employees, not to mention the contracts an employee signs upon starting a job.

We propose addressing the insider threat using a detective approach that helps a company to attribute malicious acts [200]. Detective approaches, such as *accountability*, provide a mechanism to answer questions about security incidents (e.g., “why was the document leaked?”) and attribute responsible parties a posteriori. Attack attribution is the process of identifying the perpetrator of a cyber-attack [148]. This mechanism increases forensic readiness and establishes the basis for taking legal action against an attacker [172, 189, 93]. As such, attribution can be considered in many cases as a deterrent measure [93, 181].

Insider attack attribution does not inherit the challenges facing attribution such as tools prepositioning [201], and the anonymity of the Internet [93]. Still, to the best of our knowledge, there are no robust approaches to attribute insider attacks. Attack attribution surveys show that most of the attribution literature focused on the IP level in network attacks, which is still inconclusive [201, 93, 181]. Instead, we tackle insider’s attacks attribution through an automated reasoning capability. In this chapter, we address the issue of creating *causal models* by relating causal models to domain-specific models such as attack trees [179].

Attack trees (ATs) are appealing to scientists for their formal syntax and semantics [132, 165], to managers for their visual nature, and to engineers for their systematic categorization of threats [112]. They are used for purposes of risk estimation, cost approximation, and defense planning. We aim to add forensics analysts to the list of AT beneficiaries and support causality inference to the list of purposes. However, ATs are not readily sufficient for after-



the-fact forensic analysis because they do not usually include potential attackers (suspects). This is what differentiates ATs from causal models (for actual causality reasoning). Thus, we analyze the implications of adding suspects to ATs. Then, we detail an extraction approach of causal models from attack trees and show their utilization to infer causality (using our results from Chapter 3, and Chapter 4). We focus on insiders' models because while creating such models, we exploit the unique property of insider threats, i.e., the ability to identify *suspects* beforehand. Also, within a company, it is possible to acquire evidence of insiders' acts, which is not trivial in open contexts. To automate the approach, we contribute (*ATCM*): an open-source tool that implements the approach with an evaluation of the efficiency, the validity of the approach, and the effectiveness of the model.

## 6.2 Preliminaries

In this section, we review the formalism of attack trees in Section 6.2.1, and we show an example of insider attacks in Section 6.2.2.

### 6.2.1 Foundations of Attack Trees

ATs model potential security threats within a system and the steps necessary to perform an attack [179]. The root node contains the ultimate goal of an attack tree while the sub-nodes describe activities that are necessary to conduct the respective parent activity/goal. The relationship between a node and its children can be either *OR* or *AND* (represented by a circular line below the node).

Depending on the required purpose, attack trees have been defined using different semantics such as multi-set semantics [132], linear-logic semantics [91], timed automata [115], Markov decision process [9], and propositional logic [165]. In this chapter, we aim to reason about the actual causality relations among binary events, i.e., whether the occurrence or absence of a specific event was the cause of another event. Hence, we use the equation-propositional semantics similar to [165]. Such formalism is simple, expressive, and general. The main difference between our definition and the definition in [165] is that we create a propositional formula for each node in the tree (excluding the leaves), while the whole tree is represented with a minimized formula of the root in [165].

For the formal definition, we follow Mauw and Oostdijk's [132] way of defining an attack tree. However, we adapt it to use propositional logic semantics. Formally, Definition 6.1 expresses attack trees.

**Definition 6.1.** *Attack Tree [132] is a 4-tuple  $AT = (\mathcal{N}, \rightarrow, n_0, [[n]])$  where*

- $\mathcal{N}$  is a finite set of attack nodes, and  $n_0 \in \mathcal{N}$  is the root node,
- $\rightarrow \subseteq \mathcal{N} \times \mathcal{N}$  is a finite set of acyclic relations,
- $[[n]]$  is a function that returns a propositional formula for each  $n \in \mathcal{N}$ , the formula represents the semantical dependency of a node on its children nodes.

## 6.2.2 Malicious Insiders Example

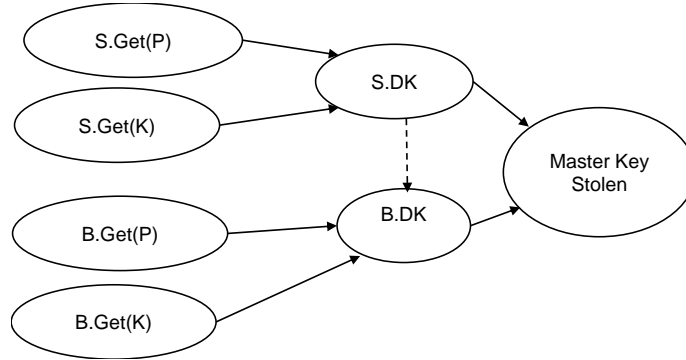


Figure 6.2: An Excerpt of the Steal Master Key Causal Model (corresponding attack tree shown in Figure 6.3).

We introduce a causal model of insider behavior that leads to stealing a master encryption key in a production environment. This is a simplified real-world example, inspired by an industrial partner, and resembles the rock-throwing example. An excerpt from the causal graph is shown in Figure 6.2; as a result of this chapter, we aspire to construct such models using ATs. Basically, the model represents one strategy to steal the key (*MKS*) by obtaining its encrypted version and decrypting it (as opposed to stealing it decrypted, which is omitted for readability). This attack can be executed by one of two administrators (assuming no collusion), Suzy (*S*) or Billy (*B*). They both have sufficient privileges in the system; however, we know that *S* has more expertise in the system and the technology.

The event of *S* or *B* decrypting the key is denoted by the variables *S.DK*, *B.DK* respectively. For that, each of them needs to read the pass-phrase from a script (*Get(P)*) and read the key from the database (*Get(K)*). For now, we assume an arbitrary causal connection between *S.DK* and *B.DK* which is meant to represent a *preemption relation*, i.e., a bias to represent *S*'s stronger abilities; such relations are crucial in causality [96]. In this chapter, we model them using dashed arrows because we think they can be dynamically altered by a modeler to express different concepts, e.g., *S has higher privileges*, *B has a better history in the company*, *S came earlier in the morning when the incident was reported*, or a combination of these factors. We have four exogenous variables (omitted from the model) that set the values for *Get(P)/Get(K)* for both *S* and *B*, i.e.,  $\mathcal{U}$  is  $\{S.Get(P)_{exo}, S.Get(K)_{exo}, B.Get(P)_{exo}, B.Get(K)_{exo}\}$ . The equations of the model follow; the underlined part of the equations shows the *preemption relation*.

- $S.DK = S.Get(P) \wedge S.Get(K)$
- $B.DK = B.Get(P) \wedge B.Get(K) \wedge \underline{\neg S.DK}$
- $MKS = S.DK \vee B.DK$

The model is not sufficient for causality inference, we still need to set the *context* (exogenous variables). This is done through logging and auditing. Assume we have the following context (1, 1, 1, 1) (*S* and *B* both got the pass-phrase and the key) when considering the ordering of the variables as provided. We use the *context* and the *model* to answer causal queries such as: *Q1*: *is Suzy the cause of stealing the key?*, or *Q2*: *what is the actual cause of exposing the key?* Let us answer *Q1* by checking if  $S.Get(K)$  is a cause of  $MKS$  with  $W = \{B.DK\}$  using the conditions from Definition 2.4. Equation 6.1 shows the crucial steps (of checking AC2) to conclude that  $S.Get(K)$  is the cause. Note that with an empty set  $W$ , AC2 does not hold (case of preemption), but with  $W=\{B.DK\}$  (Step 3 Equation 6.1), the effect does not happen (Step 4) and hence  $S.Get(K)$  is a cause.

Step 1	$S.Get(K) = 0$	Intervening on x	
Step 2	$S.DK = 1 \wedge 0 = 0$	Other variables state	
Step 3	$B.DK = 0$	Cannot change this variable	(6.1)
Step 4	$MKS = 0 \vee 0 = 0$	Effect is not happening	

Additionally, we can answer *Q2* by checking if  $B.Get(K)$  is a cause? Following similar steps, the answer is no, because no matter how  $W$  is set,  $MKS$  will still be True. These questions cannot be answered using an attack tree only. Even if we have attributed the attack tree with the potential suspects, we still cannot infer actual causality directly in cases of preemption or missing events. In this chapter, we contribute a method that uses attack trees to construct causal models with suspects and preemption relations; thus, establishing the ability to use causal reasoning to answer queries in the context of insider attacks.

### 6.3 Attack Trees to Causal Models

Causality is model-relative; thus, the creation of a model is a crucial requirement for causality and blame attribution. Although attack trees are widely used to model attacks on a system, they are not readily *sufficient* to attribute blame. Mainly because they normally do not include the attacker; rather, they represent the attack strategies. That said, they are a promising starting point to creating causal models since they express the dependencies among attacker acts, and match the properties of a causal model. First, ATs are already a propositional combination of events with (*OR*, *AND*) relations. The ability to formalize ATs in boolean algebra makes them trivial to be expressed as causal models. Second, HP focuses on acyclic models; ATs are acyclic. This section proposes an automated methodology for constructing causal models based on ATs. Our methodology refers to the following activities that are discussed throughout this section.

1. *Suspect Attribution*. This Refers to representing potential suspects in the model. In Section 6.3.1, we transform the original AT  $\mathcal{T}$  to an *attributed* AT  $\mathcal{T}'$ .
2. *Tree to Model transformation* (Section 6.3.2). It includes a.) *variable selection*: listing the different factors that are considered in the model. They represent the causes, effects,

and the environment. Each factor is expressed as a variable in the model. b.) *variable classification*: classifying what can be considered as a cause (or effect) (endogenous) and what not (exogenous). c.) *semantics expression*: representing how the variables affect each other using propositional logic operators like *and*, *or* and *negation*.

3. *Preemption Relations Addition* (Section 6.3.3). This Refers to incorporating useful knowledge about the variables to create preemption relations.

### 6.3.1 Suspect Attribution

To bring it closer to causal models, we add *suspects* to ATs. As shown in this section, the way suspects are added is crucial in determining the scope of the causal queries that can be answered using the resulting model. To the best of our knowledge, no prior work has tried to explore approaches to restructure ATs to include *roles* in an automated manner. *Instances* of roles (e.g., data-center admin Suzy) are the potential attackers (suspects) that have privileges to perform an attack. We refer to the process of adding suspects (roles or instances of them) to AT as *suspect attribution*. Within our approach, an attribute can be both a *role* or an *instance*. Attribution at the role level addresses potential scalability issues that may arise when dealing with a high number of instances (e.g., a company with 200 employees). On the other hand, on the instance level, attribution produces fine-grained models that can be used to attribute human blame. In the following, although we discuss instance-based attribution, our approach works similarly with role-based attribution.

Suspect attribution is an automated *unfolding* (duplicating) task of parts of the tree followed by *allotting* the new parts to a suspect (or a role). To create a new branch for each suspect, we keep the parent node of the gate, and introduce an intermediate level of attribution nodes that correspond to insiders. The allotment is represented by renaming the nodes to include the suspect identifier, e.g., *Billy.Read\_Pass\_Phrase* or *ADMIN.Read\_Pass\_Phrase*. Regardless of its location, a subtree containing a node and all its descendants is attributed according to Definition 6.2.

**Definition 6.2.** A subtree  $\mathcal{B} = (\mathcal{N}, \rightarrow, n_0, [[n]])$  is attributed with suspects  $\{s_1, s_2, \dots, s_l\}$  by: **1)** Creating a set (size  $l$ ) of  $\mathcal{B}$  duplicates, denoted  $\{\mathcal{B}_1, \mathcal{B}_2 \dots \mathcal{B}_l\}$ . A duplicate  $\mathcal{B}_i$  contains the nodes of  $\mathcal{B}$  with every node renamed with  $i$  suffix. **2)** Constructing a new tree  $\mathcal{AB}$  with root  $n_0$  from  $\mathcal{B}$ , then adding the disconnected  $\{\mathcal{B}_1, \mathcal{B}_2 \dots \mathcal{B}_l\}$ , and connecting their root nodes using an OR function with  $n_0$ .

According to the structure of a tree, unfolding can be done at different levels. However, depending on the internal structure, this may produce trees that model different attack vectors. Consequently, the range of the causal-queries that can be analyzed using the resulting models depends on the unfolding level. For example, in Figure 6.3, we present the complete AT of the example in Section 6.2.2, including stealing the key decrypted. Figure 6.3 is modeled using ADTool [111, 60], which denotes an AND relation by the presence of a horizontal edge touching the input arcs of a node. Let us consider attributing the left

subtree of Figure 6.3 with *two* instances of an *admin* role, i.e., Billy and Suzy. We can do that at *level two* (root level is one). The resulting tree is represented in Figure 6.4. It clearly models the possible ways to steal the master key by *either* Billy or Suzy. The complete attack paths in the tree allow expressing the behavior of *one* suspect performing an attack.

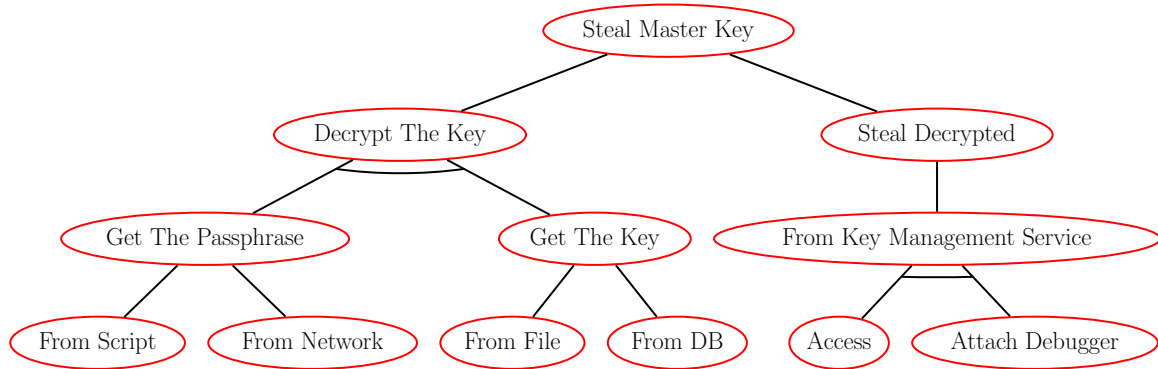


Figure 6.3: Steal Key Attack Tree (drawn using ADTool [111, 60])

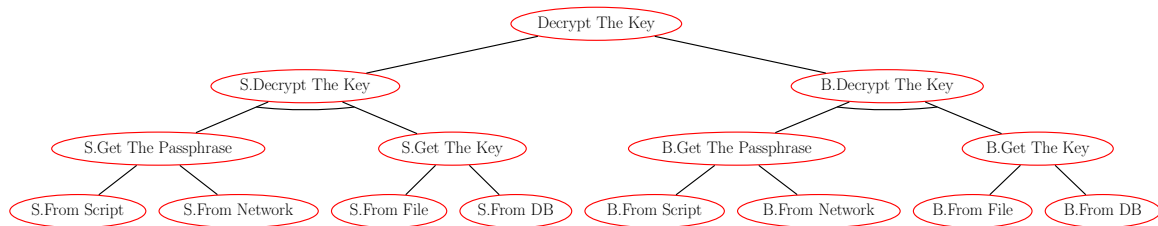


Figure 6.4: L-2 Unfolding (drawn using ADTool [111]). As an alternative, role-based attribution will have Admin, and Secretary instead of Suzy and Billy.

Alternatively, we can attribute the suspects at the *third* level (*L-3*). Interestingly, the resulting tree, as seen in Figure 6.5, models more possibilities than the previous case; now, we can model attack paths with a possibility of *collusion* between insiders [109]. As a result, attacks that involve *both* Suzy and Billy cooperating to steal the master key are now covered in this tree, and hence, causal-queries to blame them are possible on the resulting model.

Since collusion attacks are plausible among insiders [109], we use the second attribution (*L-3*), especially since it also includes the attacks within (*L-2*) attribution. This comparison is an instance of the specialization concept proposed by Horne et al. [91].

Actually, the *attribution level* is not the crucial factor in determining the expressiveness of the attribution. Somewhat, it depends on the structure and the semantics of the branch (first-level subtree). Specifically, if we have an *AND* gate in the branch, the expressiveness of the model will depend on the attribution level. If we want to include the possibility of collusion attacks, *then the unfolding should happen at a level that is greater than the AND gate level in a specific tree.*

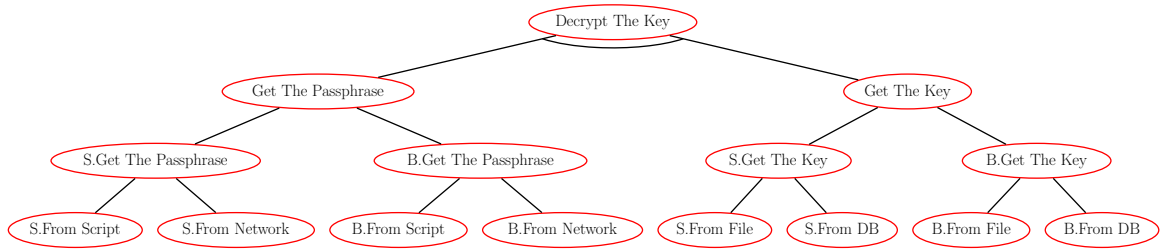


Figure 6.5: L-3 Unfolding (drawn using ADTool [111])

Although, unfolding after the last AND gate allows considering any possibility of colluding attacks, in some cases it may be unnecessary. For example, let us consider the second branch in Figure 6.3. If we attribute suspects after the fourth level, then we assume that suspects collude by having one accessing a container *and* the other attaching a debugger; this is unlikely to happen. Still, it produces a model that can be used for single-agent queries. We propose to generate causal models from *attributed ATs* based on different *attribution levels*. The branch structure automatically determines the level (based on the above), or the modeler can explicitly specify the attribution level.

### Semantics of Attribution

Let us start with **AND Gates**. An AND gate is visualized in the left column of Table 6.1. The semantics of the node is given by the formula associated with it, i.e.,  $a = b \wedge c$ . We discussed how to unfold such a gate, at the first level which does not account for collusion attacks (middle column), or at the second level (right column).

The semantics of unfolding the  $(L_1)$  with two suspects (denoted by ' and ") is shown in the second row (steps 1 – 3) of Table 6.1. The last step shows a disjunctive normal form (DNF) of the formula. Similarly, the right column shows the formulas and simplification of unfolding at  $(L_2)$ . Comparing the forms shows that the possible attack scenarios of  $(L_1)$  unfolding are included in the  $(L_2)$  unfolding (this can be seen as a specialization [91] of attack trees). In other words, the formula  $(L_1)$  *implies*  $(L_2)$ , i.e.,  $L_1 \implies L_2$  is a *tautology*. Thus, causal queries of the single blame can also be answered when unfolding on the second level.

Unfolding allows us to attribute possible suspects of an attack to the best of the modeler's knowledge. Simplifying the unfolded gates into their DNF proves the *preservation* of the original gate semantics, i.e.,  $a = b \wedge c$ . Essentially the occurrence of the two concrete actions  $(b, c)$  combined causes an event  $(a)$ . This is expressed in each clause of the DNFs. Informally, a clause is one instance of the original formula. We have to keep in mind, that this transformation is built on the assumption that the list of suspects is the universe of all the possible agents that can perform this attack. This assumption allows us to say that the semantics of the transformed tree (or branch) is now refined to enumerate all the possible scenarios, each presented as a clause that combines single or multi-suspects. Lastly, the case

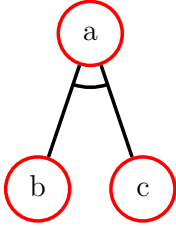
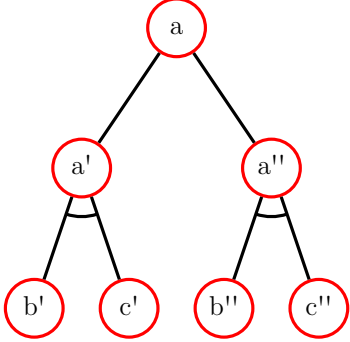
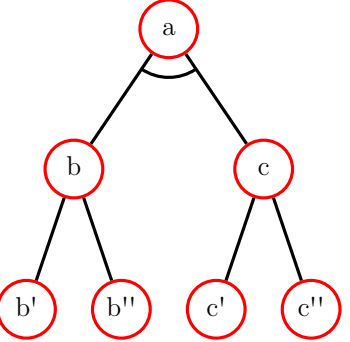
Gate	$L - 1$	$L - 2$
		
Semantics	<ol style="list-style-type: none"> <li>1. <math>a = a' \vee a''</math></li> <li>2. <math>a' = b' \wedge c'</math></li> <li>3. <math>a'' = b'' \wedge c''</math></li> <li>4. <math>a = (b' \wedge c') \vee (b'' \wedge c'')</math></li> </ol>	<ol style="list-style-type: none"> <li>1. <math>a = b \wedge c</math></li> <li>2. <math>b' = b' \vee b''</math></li> <li>3. <math>c = c' \vee c''</math></li> <li>4. <math>a = (b' \vee b'') \wedge (c' \vee c'')</math></li> <li>5. <math>a = (b' \wedge c') \vee (b' \wedge c'') \vee (b'' \wedge c') \vee (b'' \wedge c'')</math></li> </ol>

Table 6.1: Unfolding AND

of unfolding OR gates is similar and simpler because the complication of the unfolding level is eliminated. Regardless of the level, an original formula like  $a = b \vee c$ , will be unfolded to  $a = b' \vee c' \vee b'' \vee c''$ .

### 6.3.2 Attributed Attack Tree Transformation

Since we are reusing the existing knowledge in the attributed attack trees, the three activities *variable selection*, *semantics expression*, and *variable classification* are trivial. Basically, we consider each node as an *endogenous* variable that defines whether or not an attack step has been conducted. Since the nodes are connected with different operators, we use them to construct the equations and therefore express the semantic relationships between the variables. Before we do the transformation, we need to extend the tree, i.e., duplicate its leaves.

In attack trees, a leaf node represents an atomic step that is not further refined [179]. When transferring leaves into *endogenous* variables of a causal model, they lack corresponding formulas. Alternatively, we can consider them as *exogenous* variables that represent the environment (context), but then they cannot be regarded as potential causes in our reasoning. Thus, we extend the tree with a duplicate set of leaves. In other words, each leaf on the tree gets an inbound edge from a new node that has the same name with an *\_exo* suffix. Tree extension aids us in *classifying the variables*, and it also maintains the possibility that any node in the original tree can be considered as a cause. Definition 6.3 is a tree extension function, where  $E(T)$  copies the set of leaves of a tree  $T$ .

**Definition 6.3. Extension Rule** The relation  $T(\mathcal{N}, \rightarrow, n_0) \rightleftharpoons T''(\mathcal{N}'', \rightarrow'', n_0)$  is defined by the following rule.

- $\mathcal{N}'' : \mathcal{N} \cup \text{rename}(E(T), \_exo)$ ; where  $\text{rename}(A, \text{suffix})$  is a function that renames nodes in set  $A$  to with a given suffix.
- $\rightarrow'' : \rightarrow \cup \{m \rightarrow m\_exo \mid m \in E(T)\}$

We should note that the same node can occur multiple times in AT. However, in our causal model, exactly one instance of a variable exists. For the scope of this thesis, we only allow node re-occurrence among leaves. So far, we discussed the first *two* automated steps in our extraction process which are related to the AT. Now, we are ready to create the model from the extended and attributed AT. We will illustrate that by a formal mapping that depends on the definitions Definition 6.1 and Definition 2.2.

**Definition 6.4. Attack Tree To Causal Model**

- $AT = (\mathcal{N}, \rightarrow, n_0, [[n]])$  is mapped to a  $M = (\mathcal{U}, \mathcal{V}, \mathcal{R}, \mathcal{F})$  i.e.,  $AT \mapsto M$  as follows
- $\mathcal{U} = E(AT)$ , where  $E(AT)$  returns the leaf nodes of a tree  $AT$ ,
  - $\mathcal{V} = \mathcal{N} \setminus E(AT)$ , where  $\setminus$  is the difference between two sets,
  - $\mathcal{R} = \{0, 1\}$ ,
  - $\mathcal{F}$  associates with each  $X \in \mathcal{V}$  a propositional formula  $F_X = [[X]]$ , which corresponds to the semantical formula from the AT.

### 6.3.3 Adding Preemption Relations

So far, we discussed how to map the structure (variable and dependencies), the semantics (formulas), and their causal importance (endogenous or exogenous). Now, we augment the model with suspect-related information that is useful to create *preemption relations*. HP introduces a treatment of preemption cases by relating the involved variables “somehow.” As we mentioned in Section 6.2.2, *preemption relations* represent auxiliary connections among variables that express the same event conducted by a different suspect (e.g., *Billy.Get Key/Suzy.Get Key*). They are decisive in models that have potential identical causal relations, especially, with the *symmetrical* nature of our models brought by our attribution approach [75, 96]. They are crucial when multiple coinciding events occurred, leading to the success of an attack.

Since preemption relations can stem from different facts, it can be hard to model them in a general way. For example, they represent the level of Suzy’s privileges in a system, Billy’s criminal record, Suzy’s earlier (than Billy’s) login to the system, or a combination of such factors. These factors can be *static*, such as risk estimate or privileges<sup>2</sup> (e.g., if a manager and an employee tried to read the file at the same time then we *blame* the manager), or *dynamic* such as the temporal order of events (if two employees tried to read the file at the same time then we *blame* the earlier one).

<sup>2</sup>A privilege can, of course, be changed; however, by static, we mean that a factor is a general attribute regardless of the incident details.



To automate their modeling in the context of insiders, we propose to base the creation of preemption relations on metrics of insiders' risk assessment. Specifically, we introduce the *suspiciousness metric* (SM), which provides an order relation over the set of suspects conducting a particular type of attack. In other words, it is a value given to each suspect that aggregates their ability to perform an event or willingness to commit an attack. The precise way of calculating SM depends on the context of an incident; hence, we do not provide one; it can be a simple reflection of privileges in the system; it can be a sum of weighted factors (privileges and record); it can be a reflection of the temporal order of events. Since SM values reflect disparity among suspects, they can be *global* (a value of the attacker ability for all possible attacks) or *local* (a value of attacker ability for a specific attack). This flexibility in deciding how to calculate SM gives the modeler a method to determine whether to model preemption and how to model it, in a case by case manner. Yet, the whole concept can be automated.

We introduce preemption relations among attribution variables *one level* after the attribution level. At that level, the tree contains variables representing the same event allotted for different suspects. We connect every two variables with an edge *from the more suspicious suspect* (higher SM) *to the less suspicious suspect* (in case of equal values the edge is not added). Assume we have three suspects:  $X_1, X_2, X_3$ , each performing event  $Z$ , and the order of their ability is  $X_1 > X_2 > X_3$ . Then, the following acyclic preemption relations are added  $X_1.Z \dashrightarrow X_2.Z, X_1.Z \dashrightarrow X_3.Z, X_2.Z \dashrightarrow X_3.Z$  to the graph. The semantics of this arrow is represented by a *negation clause* added to the *less suspicious suspect* about the *more suspicious one*, i.e.,  $X_3.Z = \dots \boxed{\wedge \neg X_1.Z \wedge \neg X_2.Z}$ .<sup>3</sup>

**Definition 6.5.** Given a model resulting from Definition 6.4, a preemption relation ( $\dashrightarrow$ ) is added between two attribution variables ( $S_1.e, S_2.e$ ) of the same event ( $e$ ) for different suspects, denoted  $S_1.e \dashrightarrow S_2.e$ , if  $SM(S_1.e) > SM(S_2.e)$ .

### 6.3.4 Tool Support

We present our tool *ATCM* (Attack Tree to Causal Model). *ATCM* is a command-line tool that implements our approach.<sup>4</sup> As the name suggests, *ATCM* takes an attack tree and suspects specification as an input and generates a causal model. Attack trees are usually created using a broad variety of tools. In order to get access to the information stored in such an AT, the latter needs to be exportable to a format that can be easily accessed and used by us. Examples of tools fulfilling this requirement are *ADTool* [111, 60] which provides an XML-representation of the models created with them. Consequently, we are able to use those as input for *ATCM*.

In general, *ATCM* incorporates a three-step approach: parsing, transformation, and extraction. First of all, we need to create a machine-readable object, i.e., binary, representation out

<sup>3</sup>While this expression of preemption relations seems to exclude suspects, it is *only* crucial in certain situations, in which the modeler interferes according to blameworthiness factors.

<sup>4</sup>*ATCM* is available at: <https://github.com/amjadKhalifah/ATCM>

of a given XML-File that defines an attack tree (Parsing). For this purpose, we have developed our own parsing components. However, since this object representation is specifically tailored to each of the supported file formats, we want to transform the latter into a uniform tree representation, which comprises both attack and other similar models like fault trees, while ensuring that no semantic information is lost (Transformation). For this representation, we are using the *Model Exchange Format (MEF)* (<https://open-psa.github.io/mef/>) in a slightly simplified version.

The advantage of abstracting the specific format like ADT format is that the most essential functionality of this tool, i.e., the extraction of the causal model, needs to be developed only once. This reduces its error-proneness and increases maintainability. Once an attack has been transferred into this uniform representation, the described generation of the causal model can begin (Extraction). Lastly, we export the results in a human-readable report and generate a causal graph in the *DOT* format, which is commonly used for describing graphs in a textual format and can be rendered into visualization by multiple tools.

## 6.4 Evaluation

In our evaluation, we analyze the following qualities: *the efficiency* of the model extraction procedure, *the validity*, and *the effectiveness* of the resulting models. For the first, we discuss (in Section 6.4.1) the performance cost and the size expansion of the tree in relation to different factors. In Section 6.4.2, we focus on the quality of the model. Clearly, we do not aim to discuss the expressiveness of AT since their refinement and granularity are decided by the modeler. However, we discuss the validity of our models in relation to the input AT. Lastly, we discuss how to use the causal model in a technical setting to infer causality. We

Class	Use Case	Nodes	# Potential Attackers
HP	HP <sub>1</sub>	3	2
	HP <sub>2</sub>	2	2
Insider (Industry)	Steal Master Key	12	2 or 8
Insider (Literature)	BecomeRootUser <sub>1</sub>	8	2 or 8
	BecomeRootUser <sub>2</sub>	11	2 or 8
Artificially Generated	Artificial <sub>1</sub>	255	2 or 8
	Artificial <sub>2</sub>	1017	2 or 8
	Artificial <sub>3</sub>	3057	2 or 8

Table 6.2: Use Cases of the Evaluation

use *four* classes of use-cases in our experiments. Table 6.2 shows the particular attack trees of each class, along with the number of nodes in the tree. Each class contains one or more trees that cover different sources as follows: **1)** HP examples: We use two famous examples from the causality domain, namely: *Arsonists and Rock-Throwing* [74]. This class is mainly used for the discussion of the validity. **2)** Insiders from industry: This class includes a real-world

attack tree that comes from an industrial partner. It represents insider’s strategies to *steal a master key* from a deployment of an enterprise solution. **3) Insiders from Literature:** This class includes two attack trees borrowed from [193]. They represent privilege escalation. The first uses windows command line and scheduler, and the other uses Metasploit and Internet Explorer. **4) Artificially generated trees:** This class contains three trees that we generated automatically. They do not hold any semantic value. The aim of using them is to analyze the efficiency of extraction. In our experiments, we will vary the number of suspects and test our model extraction for 2, or 8 suspects.

### 6.4.1 The Efficiency of the Extraction

Depending on the size, the structure of the AT, the *attribution level*  $l$  of each branch, and the number of suspects  $s$ , the size of the resulting model will vary. Since we are attributing branches at different levels, the size of the resulting model is the sum of attributed branch-sizes plus one. This is expressed as  $(\sum_{i=1}^n |b_{li}(s)| + 1)$ , where  $n$  is the number of branches, and  $|b_{li}(s)|$  is the size (number of nodes) of branch  $i$  attributed at level  $l$  with  $s$  suspects. We express the attributed branch size  $|b_{li}(s)|$  as a function of suspects and its original size.

**Definition 6.6. Attributed Branch Size**  $|b_{li}(s)|$

$$|b_{li}(s)| = (s \cdot (|b_i| - |b_i|_{l>L>1} + |b_i|_{Leafs}) + |b_i|_{l \geq L > 1})$$

- $|b_i|, |b_i|_{Leafs}$  are the sizes of the original branch  $b_i$  and the number of its leaves,
- $|b_i|_{l>L>1}, |b_i|_{l \geq L > 1}$  are the size of the exclusive and inclusive subtree between the branch root and attribution level. Inclusion refers to counting the root and the leaves or excluding them.

We see that our approach increases the tree size. Especially with very large trees, forensic analysts are not supposed to inspect their models manually. Rather, they can use the algorithms from (Chapter 3 and Chapter 4) to analyze binary causal models. Thus, analysts focus on managing their ATs and formulating their causal queries. In fact, this is an example of an automated generation of causal models, which makes it necessary to have analysis engines like those from Chapter 3 and Chapter 4. Next, we evaluate the efficiency of the extraction process.

				2 Suspects						8 Suspects					
				Top		Middle		Leafs		Top		Middle		Leafs	
AT	n	l	b	n	exec(s)	n	exec(s)	n	exec(s)	n	exec(s)	n	exec(s)	n	exec(s)
SMK	12	5	2	37	0.0002	36	0.0002	36	0.0003	139	0.0004	126	0.0004	108	0.0004
Be.Root1	8	4	1	24	0.0002	25	0.0002	23	0.0002	90	0.0004	91	0.0004	71	0.0004
Be.Root2	11	4	1	32	0.0002	35	0.0002	32	0.0003	122	0.0006	125	0.0006	98	0.0006
T <sub>1</sub>	255	8	2	767	0.0069	767	0.0117	767	0.0512	3059	0.0283	2879	0.0460	2303	0.1925
T <sub>2</sub>	1017	8	8	3065	0.0354	3065	0.1133	3065	0.7473	12233	0.1380	11513	0.4610	9209	2.99
T <sub>3</sub>	3057	8	16	6129	0.0939	6129	0.4084	6129	2.94	24465	0.3700	23025	1.65	18417	11.97

Table 6.3: Efficiency Evaluation of the Model Creation.

Table 6.3 shows the execution time  $exec(s)$  in seconds and the model size  $n$  of six ATs. Their properties are shown as  $n$ : number of nodes,  $l$ : depth of the tree, and  $b$ : number of branches. We have attributed the trees with 2 and 8 suspects. We attributed each tree at *root-level*, *middle-level*, and *leaf-level*. We created benchmarks, based on *Java Microbenchmark Harness* to measure the execution time. The benchmarks measure the time from parsing an AT until the creation of the corresponding causal model. The values shown in Table 6.3 have been obtained by running 10 warm-up and 20 measurement iterations on a Windows 10 machine equipped with 8GB of RAM and a quad-core Intel® i7 processor.

For the small use cases (SMK, Be.Root1, and Be.Root2) the execution time is small (below 0.7ms). The interesting part is with the artificial trees, where we see a clear proportional increase of execution time with the *deeper* attribution levels. This is due to our recursive algorithm. Model size, on the other hand, is of less importance in that context, we can see that a 23025 node model took 1.7 sec to be extracted (L-4), while a 9209 node model took 2.9 secs (L-8). Nevertheless, these values do not exhibit a bottleneck. Hence, based on this empirical evaluation, our approach should be efficient enough for any reasonable-sized AT.

## 6.4.2 The Validity of the Approach

There are no properties that discuss the validity of a causal model. Rather, scientists have dealt with the modeling activity by example. We use a similar approach. We apply our approach to problematic examples in the literature [75] and compare the results. Our goal is to check if we were able to automate the method of creating causal models by splitting the general knowledge from the agents (suspects). Although those examples are not really security attacks, we are modeling them as such.<sup>5</sup> To that end, we followed the following process. *First*, represent the abstract causal knowledge as a tree (Table 6.4 middle column). *Second*, configure the actors in the scenarios, e.g., Billy and Suzy. *Third*, generate the model (Table 6.4 right column) and compare the generated model with the model presented in the papers.

For readability, we only present two examples (**Arsonists**, **Rock Throwing**); both are explained in Appendix A. We can see that the two versions of the models are very similar, but as Table 6.4 shows they vary a bit. The variation is an auxiliary variable that is added, in our model, at the suspect attribution level, i.e.,  $ML$ , and  $H$ . The variation is negligible because it does not affect the semantics from a causal perspective, because those extra variables are identity ( $FB = ML$ ,  $BS = H$ ). We can add an optimization step to our approach to removing one of the identical nodes ( $FB$  and  $ML$ ) or ( $BS$  and  $H$ ), we end up with an identical HP model. Furthermore, our models can be proved easily to be a conservative extension ([75]) of the model from HP.

---

<sup>5</sup>Arsonists and Rock-Throwing are typical examples in the causality literature. We may consider setting a forest on fire as an attack on the forest, with lighting matches being a possible step of an attack. We may also consider shattering a bottle an attack on the bottle, with throwing a stone being a possible step of an attack. The point here is to show that our mechanism produces valid results also for well-known examples.

Example	HP Model	Attack Tree	Our Model
Arsonists			
Rock-Throwing			

Table 6.4: Models From HP Examples

### 6.4.3 The Effectiveness of the Model

To evaluate the effectiveness of our models, we show how they are used in a production environment. We experimented with a technical setting of the example from Section 6.2.2. First, we created the corresponding *model* using ATCM. Second, we set the *context* for two concrete scenarios: *Sce-1*, in which *Suzy* stole the key with the existence of preemption, and *Sce-2*, in which *Billy* did. Third, using our reasoning engine from Chapter 3 and Chapter 4, we reason about causality. We tested the models in an environment that contains a set of micro-services and third-party software that are deployed as docker containers.

To set the *context*, we utilized monitoring tools namely, auditD to monitor file accesses, and Couchbase audit to monitor queries. These tools generate logs that we used to set the exogenous variables. For our initial prototype, the context was set manually. For example, a sentence from auditD like ... "MESSAGE" : "PATH name=../ script.txt

"..*avid= 1001 uid= 1001..* is translated into  $S.From\_Script\_exo= 1$  (Suzy's id=1001).  $U = \{S\_Script\_exo, S\_DB\_exo, S\_File\_exo, S\_NW\_exo, ..Billy's\ variables\}$ . Accordingly, we have two contexts, namely *Sce-1*  $\{1, 1, 1, 1, 1, 1, 1, 1\}$  and *Sce-2*  $\{0, 0, 0, 0, 1, 1, 1, 1\}$  when we consider the ordering of the variables. Regarding the preemption relationships,  $U1$  preempts  $U2 \dots U8$ ,  $U2$  preempts  $U3$  and so on.

Using the approaches from Chapter 3 and Chapter 4, we analyzed the two contexts using the *steal master key* 8-suspects model with 91 endogenous and 48 exogenous variables. We used the two queries from Section 6.2.2: *Q1: is Suzy the cause of stealing the key? Q2: Is Billy's decryption of the key or Suzy's the actual cause of stealing it?*

*Sce-1* represented the situation of having multiple tentative suspects. The results matched our ground truth, i.e., Suzy was concluded to be responsible for the incident. Although this may seem intuitive, it was only enabled by the fact that we made our knowledge explicit using a causal model. The analysis of *Q1* took 3.07ms and consumed 3.2MB of memory. For *Sce-2*, it was easier to conclude that Billy is the reason for stealing the key since the context was clearer (Suzy and other suspects did not log into the system). The analysis of the model for *Q2* took 2.78ms and consumed 3.2MB of memory.

## 6.5 Summary

Building causal models is an essential step towards enabling causality reasoning in modern systems. Automating these models' construction in certain domains makes it even easier to operationalize causality reasoning. In this chapter, we showed one example of this automation in the context of insiders.

Preventive security measures have a high likelihood of failing with insiders because they ought to have sufficient privileges to perform their jobs. Instead, we propose to treat the insider threat by holding them accountable in case of violations. For that, we need to create causal models that support reasoning about the causality of a violation. Current security models (e.g., attack trees) do not allow that. However, they are a useful source for creating causal models. To that end, we presented a methodology that extracts HP causal models from attack trees. However, we identified suspect attribution as a crucial step in the conversion. Thus, we introduced a method to add suspects to AT considering the possibility of them colluding. Also, we focused on creating models that include preemption relations. This work leverages our earlier results in Chapter 3 and Chapter 4 to enable forensic analysis of insider accidents. Although it is hard to evaluate models reasonably, we showed that our approach is *efficient* in extracting *valid and useful* models.

# 7 Automated Generation of Attack Graphs and Causal Models for Microservices

*In this chapter, we keep our focus on the general problem of causal modeling in the domain of microservices. We provide another domain-specific solution of causal modeling through the automated generation of attack graphs. Attack graphs are another type of threat models that can be used for causal reasoning. They represent technical low-level attack steps bound to hosts in computer networks. They can be automatically generated from network configuration and the publicly available information about vulnerabilities. This chapter presents an approach to generating graphs for microservices automatically and then discuss their transformation to causal models. Microservices, which are technologically heterogeneous, dominate service systems. However, the automation of their infrastructure, depending on third-party software, increases the security risks they face. Attack graphs and causal models help practitioners analyze, attribute, and prevent plausible attack paths in their microservice-based networks. Parts of this chapter have previously appeared in the publication [94], co-authored by the author of this thesis.<sup>1</sup>*

## 7.1 Introduction

According to our solution (an abstract version of which is shown in Figure 7.1), constructing a causal model remains a necessary step to enable causality analysis. As previously illustrated (in Chapter 6), we are only able to tackle this general requirement in specific domains. This chapter presents an example of generating causal models in the domain of microservices.

Microservices, a recent approach to managing the complexity of modern applications, have been increasingly adopted in real-world systems. This is indicated by the number of companies that use microservice-based architecture [1]. Microservice-based systems are found in various domains, such as video streaming, social networks, logistics, the Internet of Things [28], smart cities [114], and security-critical systems [54]. Such architectures follow

---

<sup>1</sup>Parts of this chapter are reprinted by permission from Association for Computing Machinery: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. Attack Graph Generation for Microservice Architecture, Ibrahim et al. (20019).

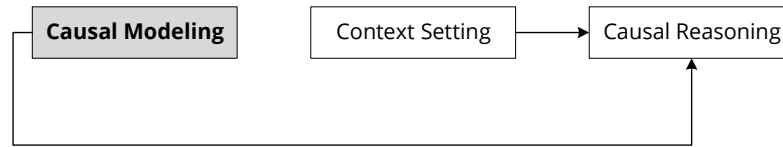


Figure 7.1: An Abstract Architecture of the Solution focusing on Causal Modeling.

the fundamental principle of Unix, i.e., systems are decomposed into small programs, each performing a single cohesive task [202]. These programs work together via universal interfaces, where each program is a microservice that is designed, developed, tested, deployed, and scaled independently [58]. Smaller decoupled services have a positive impact on some system qualities, such as scalability, fault isolation, and technology heterogeneity [146]; however, other qualities, such as security, can be affected negatively [4].

The utilization of microservices has popularized the concepts of *container-based deployment*, where new small services are shipped and deployed in containers [102]. As a result, such systems are deployed as networks of communicating microservices. Due to their lightweight and operating-system level virtualization [23], containerization frameworks, such as *Docker* [29], have become a high-performance alternative to hypervisors [113]. That said, significant concerns have been raised about containers security [4]. These concerns are motivated by the increasing number of communication endpoints among microservices, the potentially increasing number of vulnerabilities emerging from open-source tools and third-party frameworks distributed by Docker hub [183, 70], and weaker isolation (compared to hypervisor-based virtualization) between hosts and containers because all containers share the same kernel [23, 27]. In this chapter, we tackle the problem of forensically analyzing plausible attacks on container networks using attack graphs [112].

In computer networks, attack graphs are the dominant model used to inspect the security aspects of a network. An attack graph depicts the actions an attacker may use to reach their goal, e.g., gaining the privileges of a specific host [182, 150]. Typically, experts (e.g., red teams) construct attack graphs manually; however, this manual process is time-consuming, error-prone, and does not address the complexity of modern infrastructures. Previous studies have dealt with attack graph generation for computer networks [100, 182, 150].

Following an idea similar to what we did with attack trees, this chapter aims to show that sources of causal models can be automatically generated and transformed into causal models. Unlike the static nature of our approach in Chapter 6, in this chapter, we can continuously create causal models. As part of the continuous process of building, shipping,



and deploying systems, we trigger the construction of causal models. Since causal models, like any other model, are incomplete by nature, the ability to accommodate for their change is extremely important. Also, attack graphs can be used with insiders as well as outsiders. In this chapter, we propose attack graphs as a source for causal models in continuous delivery systems. We present an approach based on methods from computer networks to automatically generate them for container-based microservice architectures. The proposed approach is implemented in a tool.<sup>2</sup> We provide an empirical evaluation of the efficiency of the proposed approach relative to generating attack graphs for real-world systems.

## 7.2 Preliminaries

As real-world software increases in size, there is an increasing need to decompose it into an organized structure to promote scalability, reusability, and readability. A software application with modules that cannot be executed independently is referred to as a monolith. Monolithic systems are characterized by tight coupling, vertical scaling and strong dependence [58]. The Service Oriented Architecture (SOA) addresses these issues by restructuring its elements into components that provide services to be used by other entities [154]. In a typical SOA, services are monolithic, which gives rise to the concept of microservices as a more fine-grained decoupling methodology [4]. The term *microservices* was first introduced in 2011 as a common term to describe the work of multiple researchers [58]. In the microservices paradigm, services are split into task-oriented units. According to Dragoni et al., a microservice is a cohesive, independent process interacting via messages. Microservices promise to have cheaper scaling, resilience, organizational alignment, and composability [146]. On the other hand, they add additional complexity and have a wider attack surface as the need for many services to communicate with each other, and third-party software increases. *Container* technology has emerged in cloud computing to provide a lightweight virtualization mechanism. Container technology enables microservices to be packaged and orchestrated through the Cloud [151]. Docker is one of the most popular containerization frameworks. In Docker, a distinction is made between the terms *image*, *container*, and *service*. An *image* is an executable package that includes everything required to run an application, a *container* is a runtime instance of an image, and a *service* represents a container in production. A service only runs a single image, however, it codifies the way that image runs, what ports it should use, and how many replicas of the container should run so the service has the capacity it requires [139]. We construct attack graphs by statically analyzing the topology of the containers; therefore, we treat these terms equally.

A *vulnerability* is a system weakness that can be exploited by a malicious actor with the help of an appropriate suite of tools. Many vulnerabilities are publicly known, tracked in the so-called Common Vulnerabilities and Exposures (CVE) list, and stored in databases, such as the National Vulnerability Database (NVD). CVE<sup>3</sup> is a list of publicly known cybersecurity

---

<sup>2</sup>Accessible at <https://github.com/tum-i22/attack-graph-generator>

<sup>3</sup><https://cve.mitre.org/>

vulnerabilities where each entry contains an identification number, a description, and at least one public reference. This enables the automation of vulnerability management, security measurement, and compliance [22]. *Vulnerability scanners* attempt to detect weaknesses in a software by scanning a single host for the existence of known CVEs [51]. However, more sophisticated approaches are required because many attacks are network-based and performed in multiple steps throughout a network.

*Attack graphs* [182] are a popular way to examine network security weaknesses. They facilitate analysing a given system and detecting its vulnerable components. The definition of an attack graph may vary, however, it is essentially a directed graph comprising nodes and edges with various representations. For instance, Ingols et al. made a distinction between full, predictive, and multiple-prerequisite (MP) attack graphs [100]. A full graph is a directed acyclic graph comprising nodes that represent hosts and edges that represent vulnerability instances. Predictive attack graphs use the same representation, with the only difference lying in the constraint of when the edges are added to the attack graph. Note that predictive graphs are generally smaller than full graphs. An MP is an attack graph with contentless edges, state nodes, vulnerability instance nodes, and prerequisite nodes [100]. For the scope of this thesis, we define an attack graph as a directed acyclic graph with a set of nodes and edges similar to the full graph representation proposed by Ingols et al. [100]. A node represents the state of a host with its current privilege, and an edge represents a successful transition between two such hosts. We consider an edge as a successful vulnerability exploitation initiated from a host with a required privilege to another or the same host with a newly gained privilege.

## 7.3 Approach

In Section 7.3.1, we present the general concept behind attack graph generation, and the technical process to achieve it in microservice-based systems. Then, in Section 7.3.2, we discuss how to turn these graphs into causal models.

### 7.3.1 Attack Graph Generation for Docker

Privileges play a central role in the generation of attack graphs. Normally, privileges are modeled as a hierarchy that varies in the access level (*User*, *Admin*), and access scope (*virtual machine VOS*, *host machine OS*). The privileges used in this chapter are *None*, *VOS(User)*, *VOS(Admin)*, *OS(User)*, and *OS(Admin)*. *VOS* means that the privilege is exclusive to a virtual machine while not affecting the host machine. However in our case, unlike hosts in a network, these privileges refer to images and not virtual machines. The *OS* keyword means that a user who has this privilege can control the host machine. Since *VOSs* are isolated from host machines and their exploitation does not imply the exploitation of the host machine, they are at the lower level of the hierarchy [5]. *None* means that no privilege is obtained, *User* means that only a subset of user level privileges is granted, and *Admin*

grants control over the whole system.

As mentioned, *nodes* and *edges* are the basic building blocks of an attack graph. A *node* represents a combination of a compromised Docker image and a certain privilege gained by the attacker after exploiting a vulnerability. A directed *edge* between two nodes represents an attack step from one node to another (adjacent exploitable image with the gained privileges). Each edge is typed with the (CVE) that could be exploited in the end node.

For attackers to exploit a given vulnerability, they must have certain *preconditions*, i.e., the minimum privileges required to exploit [5]. Once an attacker meets these preconditions and exploits the vulnerability, s/he gains the privilege of the end node as a *postcondition*, and a directed edge is added between the two nodes. Both the preconditions and postconditions in this study are transformed from precondition and postcondition rules manually selected and evaluated by experts [5]. The precondition and postcondition rules use the fields defined by the NVD, as well as an occurrence of specific keywords from CVE descriptions [22].

### Example

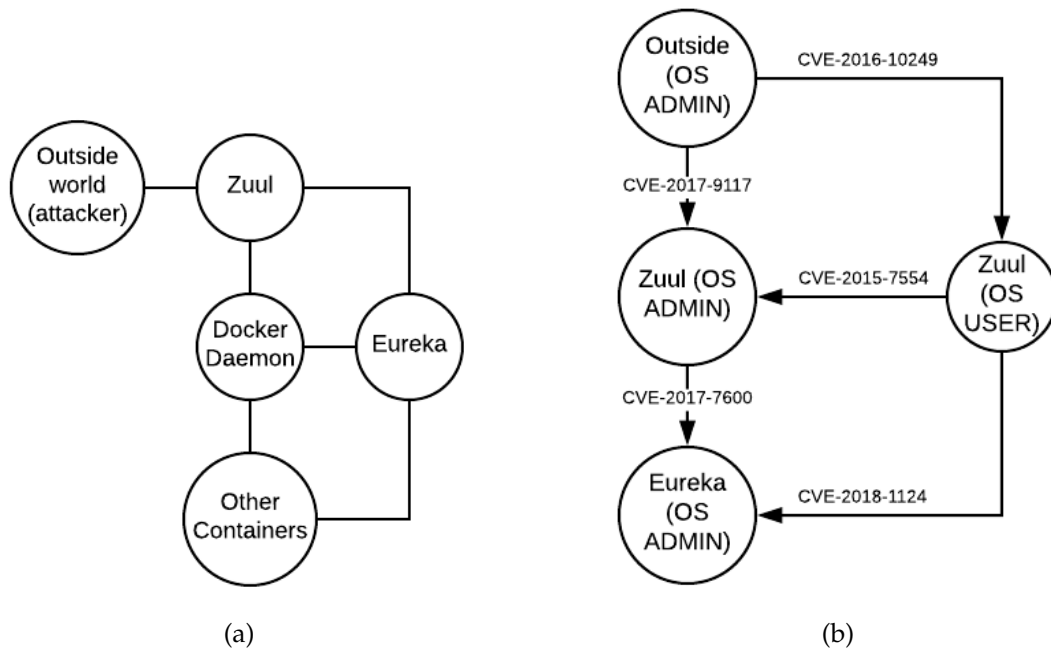


Figure 7.2: Reduced Netflix OSS example: (a) Topology Graph (b) Resulting Attack Graph

Here, we present a small example to demonstrate how attack graph generation works in practice. The example is taken from the Netflix OSS GitHub repository. The Netflix OSS example is a Spring Cloud-based microservice architecture that uses the following microservices: Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing

(Zuul), and Client Side Load Balancing (Ribbon). Figure 7.2a shows a subset of the example topology, where each node denotes a container and each edge is a connection between two containers if one calls the other. The topology comprises an "Outside" node and a "Docker daemon" node, as well as Zuul, Eureka, and other nodes. According to Netflix, Zuul is an edge service that provides dynamic routing, monitoring, resilience, and security functionalities. Eureka is a Representational State Transfer (REST) based service primarily used in the cloud for locating services for load balancing and fail-over of middle-tier servers. Figure 7.2b shows a part of the corresponding attack graph, where a node is a pair of the image and its privilege, while an edge represents an atomic attack. Parts of both graphs have been omitted intentionally for simplicity. An example path an attacker would take could be to first attack the Zuul container by exploiting the CVE-2016-10249 vulnerability by crafting an image file, which triggers a heap-based buffer overflow<sup>4</sup> and gains the USER privilege. With this USER privilege, an attacker can exploit the CVE-2015-7554 vulnerability on the same container via crafted field data in an extension tag in a TIFF image<sup>5</sup> to gain the ADMIN privilege. Once the ADMIN privilege has been obtained on the Zuul container, the attacker can attack the Eureka container by exploiting CVE-2017-7600 via another crafted image<sup>6</sup> and gain the ADMIN privilege. Note that this is not the only path the attacker can take to obtain ADMIN privileges on the Eureka container. Another path would be to exploit the CVE-2018-1124 vulnerability by creating entries in the file system (procfs) by starting processes, which could result in crashes or arbitrary code execution.<sup>7</sup> This vulnerability can be exploited by having only the USER privilege on Zuul to gain the ADMIN privileges of the Eureka container directly. Our attack graph generator shows both paths because it is of interest to identify all possible routes through which a container can be compromised.

### The Attack Graph Generator System

Figure 7.3 shows an overview of our attack graph generator, where the rectangles denote the main system components, the arrows indicate the flow of the system, and the files are intermediate products. The proposed attack graph generator comprises three primary components, i.e., the *Topology Parser*, the *Vulnerability Parser*, and the *Attack Graph Generator*. The Topology Parser reads the static underlying topology of the system and converts it to a format required by the Attack Graph Generator. The Vulnerability Parser scans the vulnerabilities for each image, and the Attack Graph Generator generates the attack graph from the topology and vulnerabilities files. In the following, we first examine the system requirements and then describe each component in greater detail.

The proposed generator is developed and tested for Docker 17.12.1-ce and Docker Compose 1.19.0 [139]. Docker Compose<sup>8</sup> is a tool for defining the orchestration of multi-container

---

<sup>4</sup><https://nvd.nist.gov/vuln/detail/CVE-2016-10249>

<sup>5</sup><https://nvd.nist.gov/vuln/detail/CVE-2015-7554>

<sup>6</sup><https://nvd.nist.gov/vuln/detail/CVE-2017-7600>

<sup>7</sup><https://nvd.nist.gov/vuln/detail/CVE-2018-1124>

<sup>8</sup><https://docs.docker.com/compose/>

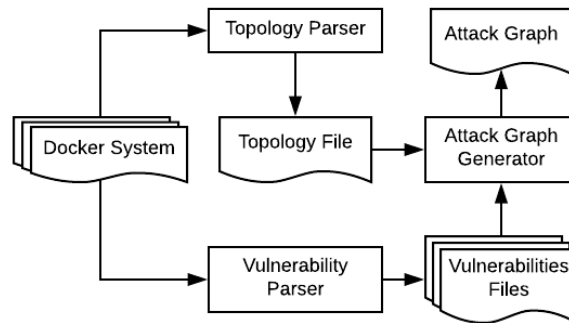


Figure 7.3: Attack Graph Generator

applications. Docker Compose provides a static configuration file that specifies the system containers, networks, and ports. Note that Clair and ClairCtl<sup>9</sup> were used for vulnerability scanning. The generator was written in Python 3.6. Although we used specific versions of these tools, the pipe and filter structure of the generator can be easily extended to other versions of Docker-Compose, vulnerability scanners, and microservice architectures.

**Topology Parser.** To generate an attack graph for a given system, we must arrange its components and connections as a system topology. We used Docker Compose to extract the static topology. The static topology refers to the structure of the images (not the run-time containers or instances).<sup>10</sup> Docker Compose provides a file (`docker-compose.yml`) that is used to describe the orchestration of the services. For an application to be useful in most cases, it communicates with the outside world, i.e., it has endpoints that can be used by an outer network. In Docker, this is typically accomplished by publishing ports. This is the case for both computer networks and microservice architectures.

Another consideration is *privileged access*.<sup>11</sup> In order to function properly, some containers obtain certain privileges that grant them control over the Docker daemon. For example, a user may want to run hardware (e.g., a webcam) or applications that demand higher privilege levels from Docker. In Docker, this is typically achieved either by mounting the Docker socket or specifying the "privileged" keyword in the `docker-compose.yml` file. Here, an attacker with access to these containers also has access to the Docker daemon. Once the attacker has access to the Docker daemon, he has potential access to the entire microservice system because each container is controlled and hosted by the daemon.

<sup>9</sup><https://github.com/coreos/clair>

<sup>10</sup>In principle, this work can be extended to support dynamic topology. We argue that from a forensic attribution perspective, we are interested in which image is involved in compromising an asset, regardless of the run-time instance. That said, there may be examples where attributing run-time instances is required.

<sup>11</sup><http://obrown.io/2016/02/15/privileged-containers.html>

**Vulnerability Parser.** In the preprocessing step, we use Clair to generate the vulnerabilities for a given image. Clair is a vulnerability scanner that inspects a Docker image and generates its vulnerabilities by providing a *CVE-ID*, a description and an attack vector for each vulnerability. An attack vector is an entity that describes which conditions and effects are connected to the given vulnerability. We collect the fields in the attack vector as defined by the NVD [22]:

- Access Vector (Local, Adjacent Network, Network)
- Access Complexity (Low, Medium, High)
- Authentication (None, Single, Multiple)
- Confidentiality Impact (None, Partial, Complete)
- Integrity Impact (None, Partial, Complete)
- Availability Impact (None, Partial, Complete)

Since Clair does not provide a command-line interface to analyze a Docker image, we use Clairctl to analyze a complete Docker image.

**Attack Graph Generator.** After the topology is extracted and the vulnerabilities for each container are generated, we proceed to attack graph generation. Here, we first preprocess the vulnerabilities and convert them to sets of preconditions and postconditions. To achieve this, we match the previously acquired attack vectors from the vulnerability database and keywords of the descriptions of each vulnerability to generate attack rules. When a subset of attack vector fields and description keywords matches a given rule, we use the precondition or postcondition of that rule. An example precondition attack rule would be for a vulnerability to have "gain root," "gain unrestricted, root shell access" or "obtain root" in its description and the impacts from the NVD attack vector [22] to be "COMPLETE" to obtain the OS(ADMIN) precondition [5]. If more than one rule matches, we take the rule with the highest privilege level for preconditions and the lowest privilege level for postconditions. If no rule matches, we take None as the precondition and ADMIN(OS) as the postcondition. This results in a list of container vulnerabilities with their preconditions and postconditions.

**Breadth-first Search (BFS).** After preprocessing, the vulnerabilities are parsed and their preconditions and postconditions are extracted. Together with the topology, they are fed into a BFS algorithm. BFS, a popular search algorithm, is utilized to traverse the topology graph by looking at the neighbors of a given node before diving deeper into the graph. The pseudocode for our modified BFS algorithm is shown in [94]; it requires a topology, a dictionary of the exploitable vulnerabilities, and a list of nodes with privileged access as input. The output comprises nodes and edges that form the attack graph. Informally, the procedure to generate a graph as follows: First, the algorithm initializes the nodes, edges,

queue, and passed nodes. Then, it generates the attacker node as the node where the attack begins. The attacker node is a combination of the image name ("outside") and the privilege level (ADMIN). Then, in a loop, the algorithm iterates through each node, checks the given node's neighbors, and adds the edges if the conditions are satisfied. If a neighbor was not passed, then it is added to the queue. The algorithm terminates when the queue is empty.

The used BFS algorithm is characterized by the following properties. **1.) Completeness:** BFS is complete, i.e., if there is a solution, BFS will find it regardless of the graph type. **2.) Termination:** This follows from the monotonicity property. Monotonicity is ensured if it is assumed that an attacker will never need to relinquish a state [100, 150, 8]. In this implementation, each edge is traversed only once, which ensures that monotonicity is preserved. **3.) Complexity:** The algorithm's complexity is  $O(|N| + |E|)$ , where  $|N|$  is the number of nodes and  $|E|$  is the number of edges in the attack graph.

### 7.3.2 Extracting Causal Models from Attack Graphs

The ultimate goal of this work is to automatically generate causal models. For this purpose, we transform the resulting attack graphs to causal models. Attack graphs are already directed acyclic graphs like causal networks. However, since the edges are typed with a CVE, we need to consider this when creating the set of equations. We present a semi-formal mapping that we implemented within our tool. Each node  $n_{h,p}$  and each edge  $e$  in the input attack graph is turned into an endogenous variable in the causal model; recall that  $n$  is a combination of a host ( $h$ ) and the privilege of the attacker ( $p$ ). The graph leaves, i.e., nodes without any inbound edges are represented by an exogenous variable in addition to their endogenous variable. Similar to the graph leaves, each edge  $e$  is represented with an endogenous variable that is set by an exogenous variable; it represents exploiting a specific CVE. Multiple instances of the same CVE are represented with the same variable. Now the equation for each leaf node is a simple identity function, i.e.,  $n = n_{exo}$ ; the same applies for each edge. For non-leaf nodes, we need first to consider each inbound edge. The semantics of each edge  $e$  connecting a source node  $n_{h,p}^s$  to destination node  $n_{h,p}^d$  is the propositional clause  $(e \wedge n_{h,p}^s)$ . That is, to get to the destination node, an attacker has to exploit a CVE (represented by  $e$ ) after she already gained access to the source node  $n_{h,p}^s$ . Generally, the equation of a non-leaf node endogenous variable is the disjunction of all the edge clauses, i.e.,  $n_{h,p} = \bigvee_{i=1}^k (e_i \wedge n_{h,p}^{s_i})$ , where node  $n$  has  $k$  inbound edges from adjacent nodes. Note that a more expressive semantics is possible. For instance, the relations among privileges on the same host (e.g.,  $n_{host2,admin}$  and  $n_{host2,user}$ ) can be expressed with entails relation. However, the purpose of this chapter is to show that causal models can be automatically generated from network topology; thus, we do not discuss the models' expressiveness in the forthcoming sections.

## 7.4 Evaluation

In contrast to previous experiments, where only the models were required, we need real-world technical setups to evaluate our approach in this chapter. We demonstrate the use-cases used in our evaluation in Section 7.4.1. Our experiments focus on assessing the scalability and practicality of the generation process. In Section 7.4.2, we discuss the scalability of the proposed system with different numbers of containers and varying degrees of connectivity. Finally, in Section 7.4.3, we discuss the effectiveness of the models for causality reasoning.

### 7.4.1 Experiment Setup

Name	Description	Technology Stack	No. Containers	No. Vulnerabilities	Source
Netflix OSS	Combination of containers provided by Netflix.	Spring Cloud, Netflix Ribbon, Spring Cloud Netflix, Netflix's Eureka	10	4111	<a href="https://github.com/Oreste-Luci/netflix-oss-example">https://github.com/Oreste-Luci/netflix-oss-example</a>
Atsea Sample Shop App	An example online store application.	Spring Boot, React, NGINX, PostgreSQL	4	120	<a href="https://github.com/dockersamples/atsea-sample-shop-app">https://github.com/dockersamples/atsea-sample-shop-app</a>
JavaEE demo	An application for browsing movies along with other related functions.	Java EE application, React, Tomcat EE	2	149	<a href="https://github.com/dockersamples/javaee-demo">https://github.com/dockersamples/javaee-demo</a>
PHPMailer and Samba	An artificial example created from two separate containers. We use an augmented version for the scalability tests.	PHPMailer(email creation and transfer class for PHP), Samba(SMB/CIFS networking protocol)	2	548	<a href="https://github.com/opsxcq/exploit-CVE-2016-10033">https://github.com/opsxcq/exploit-CVE-2016-10033</a> <a href="https://github.com/opsxcq/exploit-CVE-2017-7494">https://github.com/opsxcq/exploit-CVE-2017-7494</a>

Table 7.1: Microservice Architecture Use-cases

Microservice architectures use different technologies, different numbers of containers, various degrees of connectivity, and may contain different vulnerabilities. Therefore, it is critically important to demonstrate that an attack graph generator works efficiently in such scalable scenarios. Here, we tested the proposed system on real and own GitHub examples as shown in Table 7.1. We collected publicly available examples to facilitate potential future comparison characterized by different system properties (e.g., topologies, technologies and vulnerabilities) and different usage domains. We also observed the fact that an overwhelming majority of publicly-available examples are meant to teach the technology, hence, are composed of a small number of containers. The resulting list of examples contains *NetflixOSS*, *Atsea Sample Shop App*, *JavaEE demo*, and our own application *PHPMailer and Samba*.

We ran the attack graph generator and manually verified the resulting attack graphs for the small examples based on domain knowledge under the assumption that the output from Clair, the NVD attack vectors [22], and the preconditions and postconditions from Aksu et



al. [5] are *correct*.<sup>12</sup> After running the proposed attack graph generator, the attack graphs for the Atsea Sample Shop App and JavaEE demo were small as expected, containing only a few nodes and edges. The structure of the NetflixOSS<sup>13</sup> attack graph demonstrated a nearly linear structure in which each node was connected to a small number of other nodes to form a chain of attacks. This linearity is due to the fact that each container is connected to only a few other containers to reduce unnecessary communication and increase encapsulation. Therefore, based on this degree of connectivity, an attacker needs to perform multiple intermediate steps to reach the target container. Note that all examples terminated, there were no directed edges from containers with higher privileges to lower privileges, and no duplication of nodes. In addition, the run time of the proposed system with each example was short (less than one second).

### 7.4.2 Scalability evaluation

Extensive studies of attack graph generators scalability are rare in the literature. Many parameters contribute to the complexity of comprehensive analyses. Parameters that typically vary in this sort of evaluation include the number of nodes, their connectivity and the number of vulnerabilities per container. Even though the definitions of an attack graph differ, we hope to achieve a comprehensive comparison with current methods. We compared the proposed system to existing work in computer networks by assuming each container as a host machine and any physical connection between two machines as a connection between two containers. In the following, we examine three existing methods and their scalability evaluation results, then we present our results.

Sheyner et al. [182] tested their system (NuSMV) using both small and extended examples. In their approach, they use model checkers with a goal (property) of not compromising a specific asset. Model checkers use computational logic to determine if a model is correct in preserving the property; otherwise, it returns a counterexample. A collection of all the counterexamples forms their attack graph. The attack graph in their larger example has 5948 nodes and 68364 edges. The time required for NuSMV to execute this configuration was two hours. Ingols et al. [100] tested their system on a network of 250 hosts. They continued the study on a simulated network with 50000 hosts in under four minutes. Although their method yields better performance than NuSMV, their evaluation was based on a MP graph, which differs from our target graph. Ou et al. [150] provided a study, wherein they tested their system (MulVAL) using more examples. They state that the asymptotic CPU time was between  $O(n^2)$  and  $O(n^3)$ , where  $n$  is the number of nodes (hosts). With 1000 fully-connected nodes, their system required more than 1000 seconds to execute.

We used Samba [3] and Phpmailer [2] containers in our scalability experiments. We extended this example and artificially created fully-connected topologies of 20, 50, 100, 500,

<sup>12</sup>The resulting graphs can be inspected at [https://github.com/tum-i4/attack-graph-generator/blob/master/System/examples/output\\_samples/](https://github.com/tum-i4/attack-graph-generator/blob/master/System/examples/output_samples/); we omitted to add them for readability reasons.

<sup>13</sup>[https://github.com/tum-i4/attack-graph-generator/blob/master/System/examples/output\\_samples/netflix-oss-example/attack\\_graph.dot.pdf](https://github.com/tum-i4/attack-graph-generator/blob/master/System/examples/output_samples/netflix-oss-example/attack_graph.dot.pdf)

and 1000 Samba containers to test the scalability of the proposed system. As reported by Clair, the Phpmailer container has 181 vulnerabilities and the Samba container has 367 vulnerabilities. In our tests, we measured the total execution time and partial times for topology parsing, vulnerability preprocessing, and graph generation. The topology parsing time is the time required to generate the graph topology, the vulnerability preprocessing time is the time required to convert vulnerabilities into sets of preconditions and postconditions, and the graph generation time is the time required for the algorithm to traverse the topology and generate the attack graph after the previous steps are complete. The total time does not include the Clair vulnerability analysis, because this evaluation is beyond the scope of this analysis. All experiments were performed on an Intel(R) Core(TM) i5-7200U CPU (2.50GHz) with 8 GB of RAM running Ubuntu 16.04.3 LTS, and were executed five times for each example and their final time was averaged.

Table 7.2 shows the experimental results of the time needed for generating the graphs. In each experiment, the number of Phpmailer containers was constant. In contrast, the number of Samba containers increased in a fully-connected manner, where a node of each container was connected to all other containers. In addition, there were also two additional artificial containers, i.e., "outside," which represents the environment from where the attacker can attack, and the "docker host," i.e., the Docker daemon where containers are hosted. Thus, the total number of nodes in the topology graph is the sum of "outside," "docker host," the number of Phpmailer containers, and the number of Samba containers. The number of edges in the topology graph is a combination of one edge ("outside"- "Phpmailer"),  $n$  edges ("docker host" to all containers) and  $n*(n+1)/2$  edges between the Phpmailer and Samba containers. For example\_20 has 23 containers, and 253 edges in this topology graph.

Statistics	example_20	example_50	example_100	example_500	example_1000
No. of Phpmailer containers	1	1	1	1	1
No. of Samba containers	20	50	100	500	1000
No. of nodes in topology	23	53	103	503	1003
No. of edges in topology	253	1378	5253	126253	502503
No. nodes in attack graph	43	103	203	1003	2003
No. edges in attack graph	863	5153	20303	501503	2003003
Topology parsing time	0.03	0.06	0.1	0.7	2.4
Vulnerability preprocessing time	0.5	0.9	1.7	6.9	15.0
BFS time	0.3	1.6	6.6	165.4	767.5
Graph Generation time (total)	0.8	2.6	8.3	173.07	784.9

Table 7.2: Scalability Results with Graph Characteristics and Generation Time (s)

For smaller configurations, the longest step was the preprocessing step. However, this time increased linearly because the container files are analyzed only once by Clair. Thus, the impact of the preprocessing decreases as the size of the example increases because of the increase in the Breadth First Search (BFS) time (graph generation). For example\_500, we note a sharp increase in execution time (165 seconds) compared to the previous example (i.e., example\_100), where the attack graph was generated in 6.5 seconds. In comparison, the total time of the attack graph generation procedure for 1000 fully-connected hosts

(784 seconds) is better than the results of Ou et al. [150], i.e., 1000 seconds. In Sheyners’s extended example (four hosts, eight atomic attacks and multiple vulnerabilities), the attack graph took two hours to create. In contrast, even for a greater number of hosts (1000), our proposed attack graph procedure demonstrates faster attack graph generation time. However, the proposed system performs worse than the generator proposed by Ingols et al., but that is attributed to the usage of the MP attack graph, which differs from our graph.

In summary, we found that the proposed algorithm generates attack graphs efficiently and handles a system with 1000 containers in 13 minutes. Considering the strongly-connected system employed in the experiment and the high number of vulnerabilities in this system, we consider that the results demonstrate that the proposed system is a practical solution that can be used as part of the continuous delivery processes of real-world systems.

### 7.4.3 Effectiveness of the Graphs

In this section, similar to the previous chapter, we evaluate the usefulness of the generated models for causality analysis of attacks. By their nature, the generated models in this chapter are technical (low-level) models useful to *explain* attacks on microservices. The *explanation* aids investigators in understanding how the attack was carried out, potentially, with additional knowledge attribute it to an insider or an outsider. We present *two* experiments, one where we actually implemented the attack, and the causal model is small. The second example is one of the huge generated graphs from Section 7.4.2.

To utilize one of the generated models, we set-up an environment to experiment with the model from our earlier example (Section 7.3.1). The example corresponds to an open-source repository (Netflix OSS GitHub repository); part of the resulting attack graph is shown in Figure 7.2b. According to our conversion concept in Section 7.3.2, we generated a causal model that contained 8 endogenous variables  $\{zuul\_admin, zuul\_user, eureka\_admin, cve\_249, cve\_117, cve\_600, cve\_124, cve\_554\}$ , and 6 exogenous variables  $\{out_{exo}, cve\_249_{exo}, cve\_117_{exo}, cve\_600_{exo}, cve\_124_{exo}, cve\_554_{exo}\}$ . The equations (without the identity functions) follow.

- $zuul\_admin = (out_{exo} \wedge cve\_117) \vee (zuul\_user \wedge cve\_554)$
- $zuul\_user = (out_{exo} \wedge cve\_249)$
- $eureka\_admin = (zuul\_admin \wedge cve\_600) \vee (zuul\_user \wedge cve\_124)$

Assuming the system’s top-level asset is the *eureka\_admin* node, we ran the example with the specific versions of the services (contain the corresponding vulnerabilities). We could only exploit *two* vulnerabilities based on the information available online, namely *CVE-2017-9117* and *CVE-2017-7600*. We enabled logging in the corresponding libraries to ensure that we can see traces of these exploits. We performed the steps to exploit the vulnerabilities as specified in their NVD entries.<sup>14</sup>

<sup>14</sup>e.g., <https://www.cvedetails.com/cve/CVE-2017-9117/>

Setting this experiment's context was hard because the properties (exploits) are not easy to detect. However, for the two mentioned vulnerabilities, we could find corresponding logging entries that probably indicate their exploit. For *CVE-2017-9117*, seeing this sentence was our indication `15870==ERROR: AddressSanitizer(bmp2tiff): overflow on address...`; similarly, for *CVE-2017-7600*, observing `(file2stvec) outside the range of representable values of type unsigned char` in the log was our indication. For the other vulnerabilities, we also set similar rules for context setting.

We performed the steps that aimed to get to an admin privilege on the eureka docker container. After considerable effort with trials, we were able to exploit the vulnerabilities.<sup>15</sup> We collected the following context  $\{out_{exo} = true, cve_{249}_{exo} = false, cve_{117}_{exo} = true, cve_{600}_{exo} = true, cve_{124}_{exo} = false, cve_{554}_{exo} = false\}$ .

The goal of this experiment is to explain how an attacker acquired admin rights on a critical machine. In other words, from the perspective of our model, we want to know *which vulnerabilities were the cause of eureka\_admin = true?*<sup>16</sup> We used the semi-inference approach with this query with  $\vec{X} = \{cve_{117}, cve_{600}\}$ . The answer was a minimal actual cause  $\vec{X} = \{cve_{117}\}$ , and a contingency set  $\vec{W} = \{cve_{554} = false, cve_{124} = false, cve_{600} = true\}$ .

The benefits brought by this analysis are 1- the ability to detect the attack. We were not logging information on the *eureka* docker image to determine that an attacker acquired rights there. This differs from, e.g., bottle shattering in the rock-throwing example. In principle, this may seem like a contradiction with AC1; it is not; under the context above, our model evaluates to *eureka\_admin = true*. 2- We were able to attribute the attack to an exploit, which may be attributed to a human (responsible developer) or not. 3: Using a similar approach, a system admin can always assume fictional contexts and assess their systems' security. This is extremely useful because those models are generated as part of continuous integration systems, leveraging new knowledge experts are gathering online.

Additionally, we used larger instances of the attack graphs to perform causality analysis. Due to the difficulty in setting up the respective infrastructure and actually perform the attack, we simulated attack scenarios. We used the example from the scalability experiments (Section 7.4.2). We created attack graphs for the two largest architectures from above. The smaller one contained around 1000 nodes and 500000 edges in the attack graph; the larger example had 2000 nodes and about *two* million edges. According to our design (see Section 7.4.2), the attack graph is strongly connected. Hence, the resulting equations of the endogenous variables comprise many clauses (edges in the graph). As such, the larger model was problematic when constructing the formulae within our algorithms. This is due to the memory consumption when converting the formulae into their conjunctive normal form (CNF). We fixed that during the creation of the model by converting each variable's function into its CNF, which makes it easier when solving the SAT formula.<sup>17</sup>

---

<sup>15</sup>We anticipate that for practitioners this step should not be that difficult.

<sup>16</sup>We were interested in exploits as causes because they entail malicious human acts. That said, it is possible that in other contexts, investigators are interested in which machines caused the attack or a combination of machines and vulnerabilities.

<sup>17</sup>The test cases in the referenced class reproduce the experiment: <https://git.io/JTStq>

Since we are interested in finding out which vulnerabilities cause the attack, we used semi-inference algorithms. We constructed several queries of fictional scenarios presented by different contexts. For instance, is  $\vec{X} = \{cve\_086, cve\_462, cve\_166, cve\_522\}$  a cause of *samba.A400*? (Admin rights on samba container number 400) Given a context where all of them were exploited. Similar to the previous query, we changed the cause, the effect, and the context to analyze several scenarios. The queries were all answered for both models. The answers always found a minimal cause (minimal set of exploited vulnerabilities) that caused the compromise, i.e., acquiring admin rights on a specific container. The contingency set returned for all the queries contained many vulnerabilities that were not exploited and should be fixed to prove the cause.

Each query on the smaller model took around 3 minutes to answer. Simpler versions (less connectivity) of the model required less time; the CNF conversion within the algorithm took most of the answering time. On the other hand, queries on the larger model (with  $2M$  edges) required up to 37 minutes to answer (most of the time for the CNF conversion with memory swaps). We consider the models in this experiment to be examples of real-world approaches to create huge and complex causal models. Although the experiments created models with millions of edges, our reasoning technology provided answers to those within minutes.

## 7.5 Summary

Causal models are an essential element in the process of enabling causal reasoning in modern systems. Thus, constructing causal models remain a general requirement that can only be tackled in specific domains. In this chapter, we presented an example of how to fulfill this requirement. Specifically, we proposed an automated attack graph generation approach for microservice-based architectures, which serves as a causal knowledge source for accountability. Since their manual construction is an error-prone, resource-consuming activity, automating their generation provides a basis for a continuous construction of effective causal models. The construction of these models can then utilize the reasoning engines contributed earlier in this thesis.



## 8 Model-driven Contextualization for Microservices

*In addition to causal modeling and reasoning, contextualization is the third general concept in this thesis. It refers to the task of monitoring and logging the events related to an accountability incident. Contextualization, however, is a domain- or even application-specific concept. In this chapter, we keep our focus on microservice-based systems, and we provide one example to show how contextualization can be tackled in practice to support our reasoning approaches. Specifically, we address monitoring and logging granularity by leveraging threat models that we discussed in the previous chapters. We provide a dynamic and centralized automated solution that reduces the log sizes to the necessary information for causality reasoning.*

### 8.1 Introduction

In the introduction to this thesis, we stated that accountable systems require evidence to answer causal questions. The evidence is used to set the context for a causal query. In Chapter 3 - Chapter 5, we always assumed a given context; in Chapter 6 - Chapter 7, we showed specific logs statements to indicate the context. In this chapter, we focus on context-setting or contextualization (as shown in Figure 8.1), which, in systems, refers to monitoring and logging activities during the run-time of a system. While this is a generic requirement for our purpose, automated technical solutions are only relevant in specific contexts like the one we show in this chapter.

Monitoring security-related events is an integral part of the management of security incidents. The term *forensic readiness* is of importance in this field. Forensic readiness is a pre-incident approach aiming to maximize the environment's ability to collect credible digital evidence while minimizing the cost of forensics in an incident response [189]. Because microservices architecture, and its automation, produces very dynamic and complex environments, their readiness is crucial. As we saw in Chapter 6 and Chapter 7, the security of microservices is mainly complicated by third-party tools used in operations, and the need for sufficient privileges by insiders [50] [44].

Third-party tools are mainly used to automate the development and deployment cycle of modern systems. Many of them are vulnerable; researchers showed that in the Docker

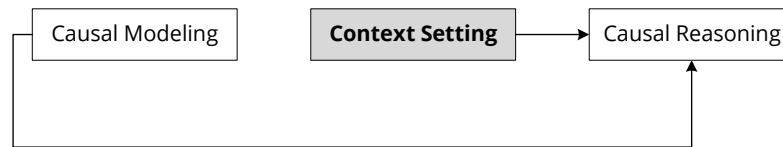


Figure 8.1: An Abstract Architecture of the Solution focusing on Contextualization.

Hub, the standard repository registry for docker images, 40% of the images have severe vulnerabilities, and more than 30% of the official repositories contain vulnerable images [70]. Further, as part of DevOps practices, developers are also responsible for deploying and maintaining their code [68]. Thus, systems are more vulnerable to malicious insiders who can take advantage of their advanced rights (e.g., access to a privileged container), or even without them, to exploit the system vulnerabilities.

Although there are many monitoring tools available, they are quite different. They are built for various purposes and configured in different ways. For example, the Linux Audit monitoring tool,<sup>1</sup> which is used to monitor file changes and system calls, is configured using configuration files. In contrast, the MariaDB Audit monitoring tool<sup>2</sup> is configured using structured Query Language (SQL) statements. Further, the granularity and requirements for logging are, in most cases, not clear; with such dynamic environments, logging and monitoring may become expensive. Without a systematic way of defining the logging granularity, log files can become very large, then more resources are required to analyze the log entries for monitoring. On the other hand, reducing the logging level may result in losing our ability to set the context due to incomplete logs, for instance.

Given models of insiders (as proposed in Chapter 6), or vulnerabilities (as proposed in Chapter 7), in this chapter, we derive monitoring rules in a specific domain, i.e., microservice-based systems, from the models. The benefits of this approach are three-fold. First, it increases the level of forensic readiness, which decreases the response time and the cost of forensics in case of an incident. Second, it proposes a way to address the question of logging granularity in a specific domain. Third, following the modern development practices, it automates the process of identifying, configuring, and deploying monitored properties. To the best of our knowledge, we are the first to suggest a methodology to determine the monitoring and logging granularity in microservices based on threat models. We evaluate the idea using a known open-source ecosystem.

---

<sup>1</sup><https://linux.die.net/man/8/auditd>

<sup>2</sup><https://mariadb.com/kb/en/library/mariadb-audit-plugin/>



## 8.2 The Approach

The goal of our approach is to reduce the logging activities to the minimum that can ensure forensic readiness in case a security incident occurs. For that, we propose to annotate the nodes in our gathered models with monitoring configurations (Section 8.2.1) and automatically deploy these configurations (Section 8.2.2, and Section 8.2.3). Different threat models are introduced in the literature varying in levels of formalism, semantics, syntax, components, and purposes [112]. In addition to attack trees and attack graphs, we surveyed the most known models in the literature to choose one that facilitates adding configuration data to the nodes.

The model that turned out to be relevant for our purpose is the *attack-countermeasure tree*. In 2010, Roy et al. proposed *Attack Countermeasure Trees (ACT)*, to address several limitations raised by previous approaches towards applying defense mechanisms on classical *Attack trees (AT)* [173, 174].<sup>3</sup> In the ACT, a node represents an attack event, which can be countered by one or more detection and/or mitigation events (nodes). A pair of detection and mitigation (or only detection) against an attack constitutes a countermeasure. The formal foundation of ACTs facilitates our approach of leveraging and transforming different sources of causal models to enable accountability. Within this formalism, countermeasures distinguish detection and mitigation events instead of defense actions. We will restrict two aspects of ACTs. According to our contextualization strategy, we consider the leaves of the models to be the exogenous variables, i.e., the most essential actions to monitor and set. As such, the countermeasure nodes are only added to leaf nodes. Second, a countermeasure, in our notion, is a detection mechanism expressed as a monitoring configuration only. Thus, we do not use the mitigation nodes in our trees. We refer to this restricted version of ACTs as *Attack-monitoring Trees AMT*.

### 8.2.1 Monitoring Configuration

A plethora of tools are used in modern systems, each with its own ability to monitor and log events. The tools use different ways to configure their monitoring activities such as properties files, SQL queries, command lines, scripts, or API calls. The configuration typically refers to a set of rules. Each rule specifies what exactly to be monitored, e.g., *read operation of customers.pdf*, the location of storing the corresponding log statement, and other properties such as the format of the statement. To handle this diversity, we designed a *configuration engine*, which is responsible for generating valid monitoring rules as well as providing instructions on how to publish these configurations on a target (docker) container. To accomplish these goals, the engine keeps track of the available monitoring tools in a system, and how to form valid monitoring rules for each. For example, in any system, the engine would be aware of the Linux Audit monitoring tool parameters (key-value pairs)

---

<sup>3</sup>According to Roy et al. [173], previous approaches such as *defense-trees (DT)* [20] had limitations to the locations of defense nodes, and *Attack-response Trees (ART)* [207] suffered from state-space explosion issues

and the respective validation rules, which are necessary to form a valid file watch or system call monitoring rule.

Since we aim to centralize the generation of valid monitoring and logging rules, each tool has to provide its language specification to the engine. Thus, the *configuration engine* uses a plugin architecture. All of the plugins follow a clearly defined interface and implement the complete language specification and configuration logic for a specific monitoring tool. By default, the engine supports the Linux Audit monitoring tool to enable the generation of monitoring configuration for file changes, and system calls, as well as, a database monitoring tool (MariaDB Audit). The engine, then, acts like a plugin manager, which, upon request, delegates the responsibility to the respective monitoring tool plugin.

### 8.2.2 Monitoring Orchestrator

As mentioned in Chapter 7, microservice-based systems ship, deploy and scale their different components as (docker) containers. The containers are orchestrated using container-management software such as the Docker Swarm platform<sup>4</sup>, or Kubernetes. A swarm consists of a cluster of hosts with their docker engines setup in swarm mode. To deploy an image on a swarm, we create a docker service that defines the number of containers (i.e., replicas), which shall be spawned in the cluster for the given image. To securely apply configurations on docker containers, swarm uses docker secrets, blobs of sensitive data, which are securely stored in the Docker swarm database. To comply with the standard tools, we build a monitoring orchestrator on top of swarm. The *monitoring orchestrator* is responsible for orchestrating the monitoring and logging configuration on a cluster of docker services. In other words, the orchestrator deploys valid monitoring rules on the running dockers securely.

To achieve this goal, the orchestrator keeps track of the list of docker containers and their monitoring abilities. When instructed to deploy valid rules for a specific tool in a particular container, the orchestrator verifies service-to-tool compatibility and securely applies the monitoring configuration.

### 8.2.3 The overall Architecture

Having introduced the conceptual aspects related to AMT, and the technical aspects of monitoring configuration and orchestration, we present the architecture of our solution. Let us iterate the main functions of the solution. The *first* function is to manage the creation, update, and storage of attack monitoring trees. The *second* function is to enable the process of generating and validating the configuration of monitoring rules for different tools. The *third* function is to orchestrate the deployment of monitoring configuration into running services. The *fourth* function is to manage the whole life-cycle of collecting, securing, and storing the generated log statements to use in causality analysis operations (Chapter 3-Chapter 5).

---

<sup>4</sup><https://docs.docker.com/engine/swarm/>

To that end, we built a model-driven system aiming to orchestrate the monitoring configuration of containerized microservices running on a cluster of hosts. The system, itself, is composed of containerized microservices communicating over the network. Figure 8.2 shows a high-level overview of the architecture, which contains *model manager*, the *configuration engine*, the *monitoring orchestrator*, and the *log manager*. These components realize the functions that we enumerated earlier in this section. In the following, we explain the steps of orchestrating monitoring rules.

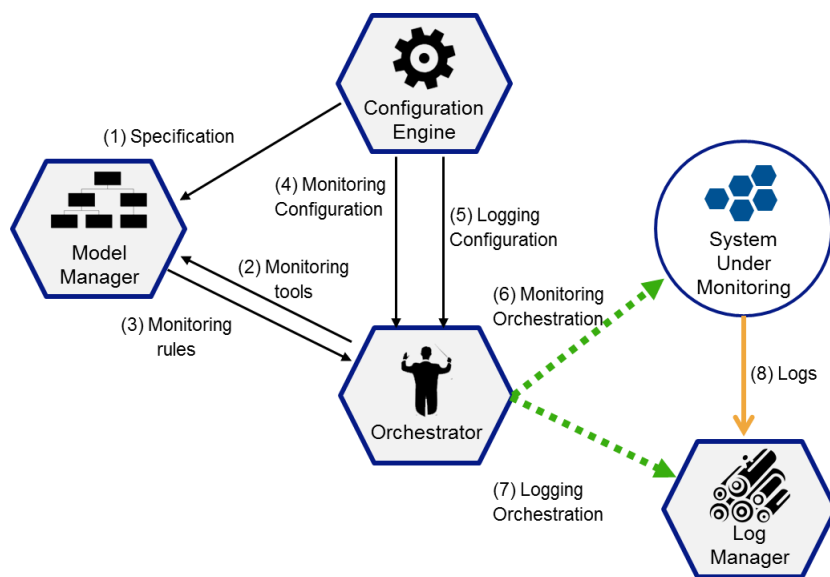


Figure 8.2: System Architecture; graphic source [118]

The *model manager* provides a workbench for the user to create AMTs, and form valid monitoring rules for each. This is achieved by (1) fetching the monitoring rules' specification (i.e., the language definition) from the *configuration engine* and (2) the available monitoring tools from the *monitoring orchestrator*. When the user defines a monitoring rule, the *model manager* (3) forwards them to the *monitoring orchestrator*. The *orchestrator* then (4) requests the monitoring and (5) logging configuration from the *configuration engine*, and (6) applies them on the System Under monitoring and (7) the *log manager* respectively. The applied monitoring configuration on the System Under Test (8) generates log events based on the user-defined monitoring rules, and the log events are collected by the *log manager*.

## 8.3 Evaluation

We evaluated the solution using a known open-source microservices ecosystem: the Pet-Clinic Project,<sup>5</sup> which is used in microservices' research [133, 192, 191]. It is a reference implementation of microservices architecture, aiming to demonstrate the common abilities of Spring. The system is composed of *eight* microservices and a database implementing a clinic for pets with the following functionality.

1. Register, view, and update the information of veterinarians, pets, and pet owners
2. View, and add information of a pet's visitation history

We conducted a threat analysis step to identify the valuable assets and the ways a malicious insider could compromise them. The assets are the credit card information of our PetClinic customers, the medical history of the pets, and the clinic's employee data. Accordingly, we built the following *four* respective *AMT* models in the *model manager* 1) Steal Customers Credit Card 2) Modify Pets Medical Record 3) Delete Veterinarians data 4) Compromise System's Availability. Finally, we executed the attack steps which were necessary to compromise the respective asset according to the AMTs. In the following, we only focus on the first attack scenario; Figure 8.3 shows an excerpt of it. Similar to our example in Chapter 6, a malicious insider can steal the customers' credit card either by reading the data from the database or by accessing the application logs of the customer microservice. To read the data from the database, the attacker can either execute a remote database query to select the credit card information residing on the owners table or via the container of the database. In the latter case, the attacker searches for the container id, use it to enter the container, and finally read the necessary information by connecting to the database and executing the respective 'select' statement. This evaluation focuses on the contextualization part, the causal analysis of this case-study is similar to Section 6.4.3.

For such attacks, we are interested in evaluating the effectiveness and efficiency of model-driven monitoring for forensics. Effectiveness, essentially means that the monitoring system can log all monitored steps based. That is, if the system contains a model of an attack scenario, and this attack is performed according to the model's steps, then our monitoring system collected logs for it. A system would be 100% effective if at least one evidence is generated for each attack action. Of course, we are assuming a secure setup of the system. The efficiency, on the other hand, entails that the system is logging the required steps only, i.e., what is necessary for setting the context.

### 8.3.1 Effectiveness

For our example attack, we configured the leaf nodes with monitoring rules. The first rule concerned the attacker's ability to list the running containers. We used the model manager with the Linux Audit plugin to define a rule that monitors the actions of listing docker

---

<sup>5</sup><https://projects.spring.io/spring-petclinic/>

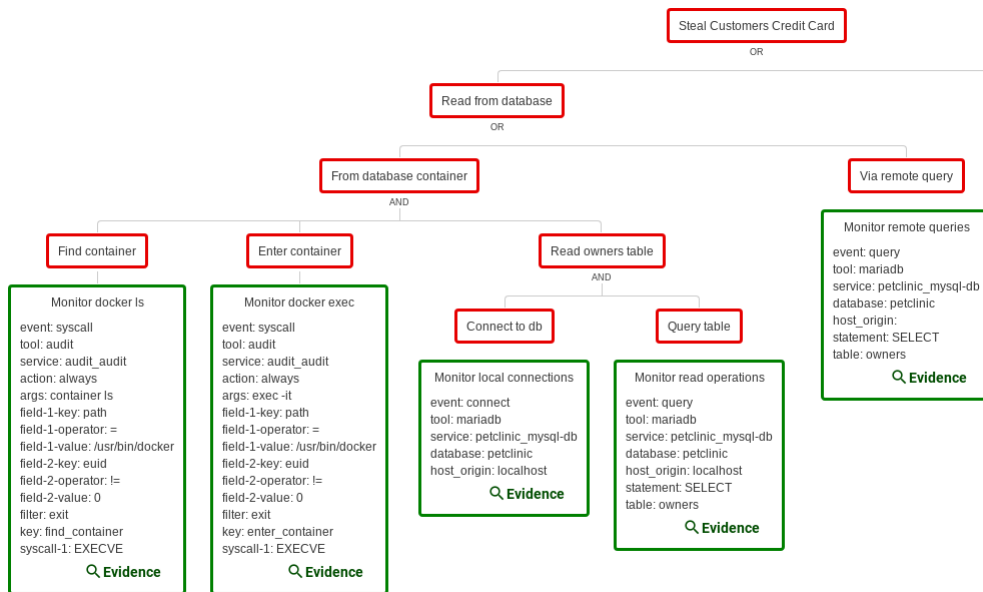


Figure 8.3: Steal customers' Credit Cards Tree; graphic source [118]

containers; Figure 8.4 shows the rule. Technically, to monitor actions of root and non-root users trying to list docker containers, we monitor the EXECVE<sup>6</sup> system calls (upon system call exit) of the docker executable (i.e., /usr/bin/docker) with the command line arguments 'container ls'. If we aspire to exclude root users from monitoring, we can add a filter, euid != 0. The generated log statements are annotated with 'find\_container' tag.

Label  
Monitor docker ls  
Max 50 characters 17/50

Event: syscall Tool: audit Service: audit\_audit

action: always filter: exit field-1 key: path field-1 operator: = field-1 value: /usr/bin/docker Max 20 characters 15/20

field-2 key: euid field-2 operator: != field-2 value: 0 Max 20 characters 1/20

syscall-1: EXECVE

key: find\_container args: container ls Max 30 characters 14/30 Max 40 characters 12/40

Figure 8.4: An example of Linux Audit Rule; graphic source [118]

Similarly, we used MariaDB Audit monitoring plugin to configure rules for the database. Specifically, the rule monitors queries reading the owners table issued from within the

<sup>6</sup>[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/security\\_guide/sec-audit\\_record\\_types](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-audit_record_types)

container. After applying the monitoring rules, we attacked as modeled. In the beginning, the attacker is listing the docker containers and uses a pipe to filter the database container. On the next step, she uses the container id and enters the container. Then she executes a command to use the 'petclinic' database and finally executes an SQL select statement on the owners table to read the credit card information accompanied by the first name and last name of our customer (i.e., the cardholder). When inspecting the log, we verified the existence of the evidence generated for the attack actions. Each step triggered one monitoring rule, and for each rule, exactly one piece of evidence is displayed. As we have at least one audit entry for each of the attack actions, by our definition, the effectiveness is 100%. This is the expected result considering our assumption that the model contains all the possible attack steps. As we saw in Chapter 7, these models can be continuously generated and updated. Also, this result shows the success of our approach in automating the mentoring using threat models as an input.

### 8.3.2 Efficiency

In this experiment, we assess the efficiency of our solution in comparison with a "monitor-everything" approach. Our notion of efficiency concerns the number of log entries that are sufficient to set the context of a causal query regarding a known attack according to our reasoning solutions. We conducted experiments in two settings. The first corresponds to our effectiveness experiments, in which we assume optimal usage of our approach; the second setting assumes a regular operation of the system that is configured to monitor and log all the events in the system. Both methods will capture the attacks; however, we aspire to show the impact of using a systematic methodology to advocate the logging granularity. We can configure the second system to log nothing, but this leads to incomplete logs.

As discussed, we defined two Linux Audit monitoring rules for our first setting: One to audit the listing of docker containers and one to audit the action of entering a docker container. For each rule, the Linux Audit generated 6 log entries instead of 1 that we desire. However, from the Linux Audit perspective, those 6 entries are part of one event. As part of our solution, We aggregate all such entries into one actual evidence. Similarly, using the MariaDB monitoring tool, we defined monitoring rules to monitor connections to the database from within the container and execution of 'select' statement on the owners table. We have one log entry for the connect action and 5 log entries for the query action. We aggregate those entries using logging configuration, which collects only the events that match the user-defined monitoring rules. Using model-driven monitoring, we fix the set of interesting events to log; then, using logging rules, we instruct our system to aggregate those events to reduce their verbosity.

In the second setting, we are monitoring everything. For that, we configured the Linux Audit to audit all system calls and file changes as well as MariaDB to audit all possible operations, issued by any user, on the database. Then we conducted the same experiment to steal the customer credit card. We made sure not to carry out any other operations. We have no guarantee that the generated log is only related to the events of the attack.

---

Approach	LINUX AUDIT	MARIADB AUDIT	TOTAL
Monitor Everything	21910	445	22355
Model-driven Approach	12	6	18

Table 8.1: Numbers of Log Entries using the Two Settings

In Table 8.1, we see the number of log entries of Linux Audit and MariaDB Audit for the system, which monitors all actions (second row), while in the last row we see the respective numbers when our monitoring solution is used. In the first setting, the Linux log contained 12 entries while the MariaDB log contained 6 entries. This is a significant reduction in comparison to the 21910 Linux entries, and 445 DB entries when monitoring everything.

In summary, it is effective to leverage our causal models (attack trees or attack graphs) to advocate monitoring a system automatically. Then, the resulting logs are sufficient to contextualize the causal queries (e.g., Section 6.4.3 and Section 7.4.3). We can also achieve this by monitoring everything in the system, but this is more expensive than our solution, as we saw in this section. That said, when monitoring everything, we can catch *unknown attacks*, but this comes with a trade-off with the cost of the forensic investigation. Unknown attacks are a challenge for the presented solution. However, as we have in this chapter and Chapter 6, we are mainly concerned with insiders' attacks. With such attacks, we usually know their patterns but cannot deploy preventive countermeasures. Further, we think security modeling and monitoring are learning processes; the more we know our systems, the better policies we can establish and continuously improve our models. Within such a continuous learning process, our model-driven monitoring system can also be combined with other approaches. The right candidate for that would be our work in Chapter 7 i.e., a method to generate models, and another to deploy monitoring rules based on new models.

## 8.4 Summary

Logging and monitoring are crucial to contextualize causality reasoning engines (contributed in Chapter 3-Chapter 5) in modern system. Such concepts differ among domains, technologies, and applications. Thus, we focus on the domain of microservices to present an automated solution to tackle contextualization challenges. Microservice-based systems are utilizing very dynamic and complex environments. Among other threats, malicious insiders can take advantage of their advanced rights to exploit such systems' vulnerabilities. We treat the issue of logging granularity in the presence of many diverse monitoring tools. To that end, we presented an approach to determine the logging granularity automatically, driven by models of incidents. With this approach, we could orchestrate the monitoring of containerized microservices using a monitoring system that extends the latest ecosystems in the domain. Given a complete model, our monitoring system was effective in contextualizing models of known attacks. It also demonstrated a significant reduction in logging resources than typical monitoring approaches.





**Part IV**

**A Framework for Accountable  
Systems**



## 9 A Framework for Operationalizing Actual Causality

*This chapter presents a unifying framework for operationalizing actual causality. Parts of this chapter have previously appeared in publication [96], co-authored by the author of this thesis.<sup>1</sup> The rapid deployment of digital systems into all aspects of daily life requires embedding social constructs into the digital world. Because of the complexity of these systems, there is a need for technical support to understand their actions. Social concepts, such as explainability, accountability, and responsibility, rely on a notion of actual causality. Encapsulated in the Halpern and Pearl's (HP) definition, actual causality conveniently integrates into the socio-technical world if operationalized in concrete applications. To the best of our knowledge, theories of actual causality such as the HP definition are either applied in correspondence with domain-specific concepts (e.g., a lineage of a database query) or demonstrated using straightforward philosophical examples. On the other hand, there is a lack of explicit automated actual causality theories and operationalizations to help understanding the actions of systems. Therefore, this chapter integrates the different contributions of this thesis into a unifying framework to address the problem of operationalizing actual causality for different domains and purposes. We apply this framework in such areas as aircraft accidents, unmanned aerial vehicles, and artificial intelligence (AI) systems for purposes of forensic investigation, fault diagnosis, and explainable AI. We show that with minimal effort, using our general-purpose interactive platform, actual causality reasoning can be integrated into these domains.*

### 9.1 Introduction

We consider enabling accountability in modern digital systems to be indispensable. Thus, developing systems' (forensic) capabilities in identifying causal factors (possibly misbehaving parties) responsible for violations is the pillar of our work. We invested the second part

---

<sup>1</sup>Parts of this chapter are reprinted from *Frontiers in Artificial Intelligence and Applications*, 325, Ibrahim et al., *Actual Causality Canvas: A General Framework for Explanation-Based Socio-Technical Constructs*, 2978 - 2985, (2020), with permission from IOS Press. The publication is available at IOS Press through <http://dx.doi.org/10.3233/FAIA200472>

of the thesis to build general causal reasoning approaches. Then, we focused on the other requirements of causality that can be only solved domain specifically; we showed examples in the context of the insider threat within microservice-based information systems in Chapter 6, Chapter 7, and Chapter 8. Accountability is also necessary for cyber-physical systems such as drones or airplanes [95, 120, 104, 167]. A related notion to accountability is the explainability in systems with AI components (e.g., machine learning applications). Since such systems are tasked with making daily decisions or predictions for humans, interest in explaining their results is growing [141, 143]. Similar to accountability, explainability is inherently causal. For all such applications, this chapter generalizes the ideas presented throughout this thesis to expand on how to operationalize actual causality.

Actual causality is well formalized by Halpern and Pearl’s (HP) definition of actual causality [74], efficiently checked using the approaches in Chapter 3 - Chapter 5, practically supported by technical domain-specific models and tools as we saw in Chapter 6 - Chapter 8, and thus suitable for the socio-technical world. However, to the best of our knowledge, explicit actual causality theories and operationalizations have not been utilized, in an automated fashion, to enable socio-technical purposes such as accountability and explanation. Although HP describes a cause in a way that matches human thinking, it was either applied in relation to domain-specific technical artifacts (e.g., a lineage of a query [134], counter-example of a model checker [123]) or demonstrated using simple philosophical examples [79]. In this chapter, we aim to answer the question *How can actual causality theories be operationalized for different domains?* This would entail establishing a general framework and automating the parts that can be automated. Actual causality reasoning, while used across different disciplines, currently lacks a clear methodology and especially tools to build, transfer, and reason over causal models. To this end, based on the concepts presented earlier in this thesis, we propose a unifying methodology to enable automated causality reasoning and demonstrate its utilization for forensic investigation, fault diagnosis, and explainable AI (xAI).

We argue that a semi-automated framework of actual causality serves as a starting point to achieve the goal of enabling complex interdisciplinary concepts such as accountability. A unifying framework diminishes the barrier to embedding causality reasoning in new domains because it allows *reuse*. As we shall see in this chapter, the framework consists of tasks that interweave social and technical boundaries. Some of these tasks reuse domain-specific methodologies and knowledge sources. Also, the framework automates solely technical tasks (e.g., reasoning) so that can be *reused* among different domains. Consequently, attention shifts to how such technical tasks serve goals such as enabling accountability. To this end, this chapter contributes the following **a)** a generalized unifying framework to operationalize causal reasoning; **b)** a general-purpose, open-source, interactive platform called the *Actual Causality Canvas* (short: *Canvas*),<sup>2</sup> which aggregates all the tools contributed in this thesis; and **c)** three new use-cases that instantiate the framework in different domains.

---

<sup>2</sup><https://github.com/tum-i22/causal-canvas>

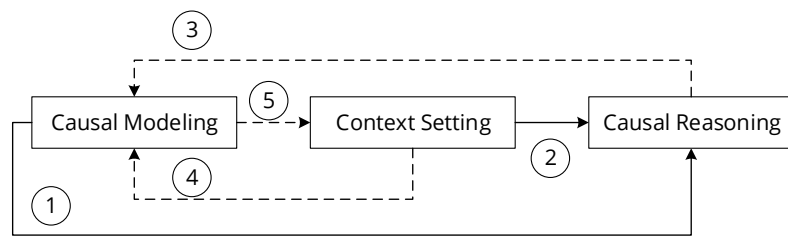


Figure 9.1: A Process View of the Framework

## 9.2 A Framework of Actual Causality

In this section, we consider the methodological aspects of the theory of actual causality. Later, we see how Canvas automates each aspect. Throughout the chapters of this thesis, we identified the three main decoupled activities of actual causality—*modeling*, *contextualization*, and *reasoning*. Figure 9.1 shows a process perspective of these components. The solid arrows (1 and 2) are inherited from the theory itself, i.e., we need a model and a context to reason about a cause. However, we also argue that interconnecting these components (dashed arrows 3, 4, and 5) may address some of their challenges. In the following, we focus on the key tasks and questions of each phase; this also serves as a requirement elicitation step for Canvas.

### 9.2.1 Causal Modeling

Causal modeling makes our understanding of causal factors explicit. Most probably, as with any modeling process, the resulting model is incomplete, hence, “there is no right model [75].” Halpern and Pearl themselves have shown several times the difficulties of coming up with a proper model and its considerable influence on the result of cause evaluation [81]. However, a causal model is nothing but the modeler’s assumption of how factors relate and influence each other, thus necessitating the ability to incrementally edit the models. Causal modeling platforms and processes must allow for the potential change in them. As such, we designed Canvas as an interactive learning system where modelers refine their causal model (according to Definition 2.2), hence *dashed arrow 3* in Figure 9.1.

Since causal models are flexible (from the rock-throwing example [126] to the model checkers’ results [123]) and intuitive, many approaches exist for model discovery. Model discovery refers to the act of creating *causal models* from some source of knowledge. For type causality in domains other than we consider, Pearl examines this topic in [157]. Understandably, model discovery by *transformation* is domain-specific and purpose-specific.

In a specific area, we need to identify existing models matching the properties of a causal one. For this thesis, such models have to be *acyclic*, *causal*, and either *binary* or *numeric*. Further, the purpose of the models must be mapped to actual causality reasoning. For instance, the domain of system security has about 30 different graphical *threat models* [112], many of which already express (type) cause-effect relations but are not directly adequate for actual causality reasoning (e.g., against insiders) as we showed in Chapter 6. A formal model-transformation function can be plugged into the process to create the causal model from other sources, considering the differences. Among the best-known threat models are attack trees, which we used as sources in Chapter 6, and attack graphs which we generated in Chapter 7. Usually, attack trees are constructed by security experts to assess the risk on a system. The ability to automatically generate attack graphs from other sources even eliminates the manual process. For *accountability* purposes, we can re-use and combine such knowledge to reason about attacks that have already occurred. The same applies, for instance, to hazard models such as fault trees and their automated generation [205]. In Section 9.4, we see additional sources of causal models facilitated by how Canvas supports their transformation.

Regardless of whether the model is created manually, transformed from other sources, or automatically generated, we emphasize that it should be augmented with *preemption* relations. As we have seen in Chapter 2 and Section 6.3.3, *preemption* should be expressed in the model when possible (the connection between Suzy's and Billy's hits); especially in symmetrical models such as the ones transformed from attack trees (Chapter 6) or fault trees [95, 205]. This reflects a discrepancy among coinciding disjunctive events in confusing situations. We think that preemption relations can stem from different requirements or facts, hence hard to model. They can reflect dynamic *temporal* order of events as was the case in the rock-throwing example; they can be *functional*; for instance, a command by a drone remote-controller takes priority over an auto-pilot command; they can also be *contractual*; for example, a sensor manufacturer is obliged to notify a drone admin about a patched software library. These relations are crucial to conclude a cause in certain situations. However, we think that they have not been sufficiently highlighted in the literature. Hence, they are explicitly noted as part of our approach.

As part of this thesis, we contributed the following open-source tools in the domain of causal modeling.

- ATCM: a command-line tool that transforms attack or fault trees to causal models according to the approach in Chapter 6.
- Attack Graph Generator: a command-line tool that generates attack graphs for containerized microservices, and transforms them to causal models according to the approach in Chapter 7.

To conclude, the questions to be answered as part of this task in a particular domain are the following: Q1: *Which transferable sources (models or data) of causal knowledge exist?* Q2:

What is the formal mapping between the source and destination syntax and semantics? Q3: How can preemption relations be identified and expressed?

### 9.2.2 Contextualization

Context setting is the act of describing an event's circumstances as an assignment of values to *exogenous* variables. For example, based on the black-box recordings of aircraft that collided near Ueberlingen [185], the investigators knew that the ground air traffic controller (ATC) had alerted the first aircraft's crew of traffic but on a wrong direction. In the accident's causal model, such information would set the value of a variable such as *Air traffic control correctly alerts the crew* to *false*. In a digital forensic investigation of cyber-attacks (similar to the one presented in Chapter 6), experts try to retrieve trustworthy log files from different systems to *set the context*. The logged events aid in understanding the occurrences. For example, a log statement like `.. "MACHINE-ID" : "8a7", "CMDLINE" : "gdb -nx -batch -ex attach.."` is interpreted as an admin with a specific ID has attached a debugger to a running process; this sets the model variable *admin attached a debugger* to *true*. These examples are meant to show that *context setting* varies among domains; however, there are established methods to help in this task. With diverse sources, from recordings, and eyewitness reports to systems' logs, we see two primary methodological *patterns* to *context setting*.

The first *pattern* is considered in scenarios where a line of "trust" exists between an agent, such as a system-admin, and a system like a company, or a citizen and traffic police. In such situations, we have an intuition about typical misbehavior; for example, an admin leaks sensitive data, or a driver goes over the speed limit. Our knowledge of such behavior can be presented as causal models that guide our monitoring effort (hence dashed arrow 5 in Figure 9.1). We presented a domain-specific technical example of this approach (have a model, monitor it) in Chapter 8, where we used attack trees to guide our logging requirements in microservices. Such methods address the principal challenge of logging and auditing capabilities, i.e., the granularity of logging or monitoring. It is expensive to log everything, and if we log less, logs can be incomplete. Further, in a specific context such as insiders, these approaches can be deterrent.

Things are not that simple, however. It is not safe to assume that we always have an intuition about the typical structures of unwanted behavior ("unknown unknowns"). In the second *pattern*, domain-specific systematic processes normally start by analyzing sources of truth and narrowing the events. Then, they structure the information so that it can be transformed into a causal model that embeds the context, hence arrow 4 in Figure 9.1. An example of this method, in the domain of accidents' investigation, is the why-because-analysis (WBA), which is introduced in Section 9.4.2 [119, 120]. This pattern does not address the granularity of monitoring since it only deals with after-the-fact sources. In addition, it faces the inherent problem of possibly ending-up with incomplete logs which is tackled in HP with the concept of probabilistic contexts [75].

The two patterns are not mutually exclusive; we can leverage both for the same system. For example, known typical malicious insider (security) attacks are modeled and monitored,

while unknown external attacks are investigated and modeled. As we will see in the case of xAI, context setting can be neither and is as simple as field assignment.

Regardless of the pattern, Canvas enables contextualization in a general way (Section 9.3). Other systems can read available causal models and set their contexts directly through the file system or an interface call. As such, a wrapper component is built into these systems to relate values to variables. One example of how to achieve this is our contribution to microservice-based systems (Chapter 8).

### 9.2.3 Causal Reasoning

Recall that reasoning includes both checking and inference. Causal checking involves verifying if a hypothesized cause is an actual cause of an effect. Inference, on the other hand, means finding a cause with no hypothesis. Both notions must be available as part of actual causality operations. Checking is already beyond NP [6], and intuitively, inference is at least as hard. The complexity results have limited the application of the actual causality theory; however, because of our approaches in Chapter 3-Chapter 5, efficient methods to check and infer causality are available for different types of models.

In the domains we consider, causal reasoning is mainly motivated by a goal of liability attribution [89], future prevention [178], or explanation [141]. Regardless of the target, causal reasoning answers a causal *query*, which consists of a context, a hypothesized cause (in the case of checking), and an effect. Since causal reasoning is automated in Canvas, the crucial question is *What is the query for each goal?*

For liability attribution purposes, we are interested in hypothesized causes that include humans. For example, *is admin "Bob" the cause of stealing the document?* For such purposes, we focus on the responsibility (Chapter 2) of the cause in the case of multiple causes. Additionally, we tend to consider negligence or failure to do an expected job as a potential cause in such situations. For example, *is the Air Traffic Controller's failure to use a cell phone the cause of the collision?*

Future *prevention* requires identifying all sufficient causes regardless of counterfactuality or minimality [126, 120] and putting countermeasures in place.<sup>3</sup> To this end, a causal query would collect all causes by trying different hypotheses regardless of their responsibility.

According to a recent survey by Miller [141], humans seek contrastive *explanations*. In other words, people would not phrase their causal queries as *Why did event P happen?* but rather as *Why did P happen instead of Q?* [141]. Miller also concludes that explanations are selected and social. We think a contrastive query can be constructed by phrasing the effect  $\varphi$  in a way that expresses this distinction. For example,  $\varphi$  will be a formula like  $\neg Q$ .

As part of this research, we contributed the following open-source tools in the domain of causal reasoning.

- AccBench: a Java library that implements non-HP causality algorithms, namely two

---

<sup>3</sup>Sometimes prevention is too expensive; sometimes it cannot be done, as shown by the example of malicious insiders in Chapter 6.



causality algorithms (based on [67] and [65]) and one policy compliance algorithm (based on [140]). The details of this work are not included in the thesis but can be inspected in [167].

- HP2SAT: a Java library that can model and solve binary causality checking questions using SAT solving as presented in Chapter 3.
- HP2Opt: a Java Library that can model and solve binary causality inference questions using optimization solving as presented in Chapter 4.
- HP-NUM: a Java library that can model and solve linear numeric causality checking questions using ILP as presented in Chapter 5.

To conclude, query formulation is a crucial part of this phase. We use the same language to formalize a causal query for different purposes. However, we adapt to the goal and include responsibility, collect all causes, or phrase the effect in contrast to reality.

#### 9.2.4 The Technical Framework of Actual Causality

Before presenting Canvas, which condenses the concepts in this thesis into an interactive tool, we summarize the toolchain behind it. Figure 9.2 shows a diagram of the different tools (in the dashed box), and technologies stack (in the lower part) used in this thesis to support and evaluate the contributed ideas. A significant emphasis is put on the computational aspects of actual causality reasoning. This can be seen in the dark boxes in Figure 9.2. In the upper part of the diagram, we list the elements of the actual causality solver that include the different encoding and formulations of the problem. Namely, as part of this thesis, we propose a SAT-based, Brute-Force based, MaxSAT-based encoding of the problem of actual causality checking in binary models. Moreover, we have ILP formulations for actual causality checking and inference in binary models, as well as numeric models. All these formulations are standardized to support the interactive formation of causal queries. For reproducibility, the solver is equipped with an automated benchmarking ability. A default data-set (detailed in Appendix A) can be benchmarked with different parameters against all the approaches in the solver. The data-set can be extended with new causal models easily. Similarly, controlled experiments using different settings can be constructed leveraging the benchmarking parameters. The lower part (solid border) of Figure 9.2 shows the stack of technologies and tools that the solver is built on. We use a set of efficient and robust solvers as default tools; however, since we use the standard format for each respective paradigm (e.g., ILP), the default tools can be replaced with more recent solvers.

The second part of Figure 9.2 summarizes the concepts for causal modeling and contextualization. For modeling, we utilized domain-specific models such as attack trees (as modeled by ADTool [111]) and graphs for (semi-)automated causal model generation. We extended the tool support for fault trees as well (as modeled by EMFTA). Further, our modeling capabilities include the ability to import models from other domains such as

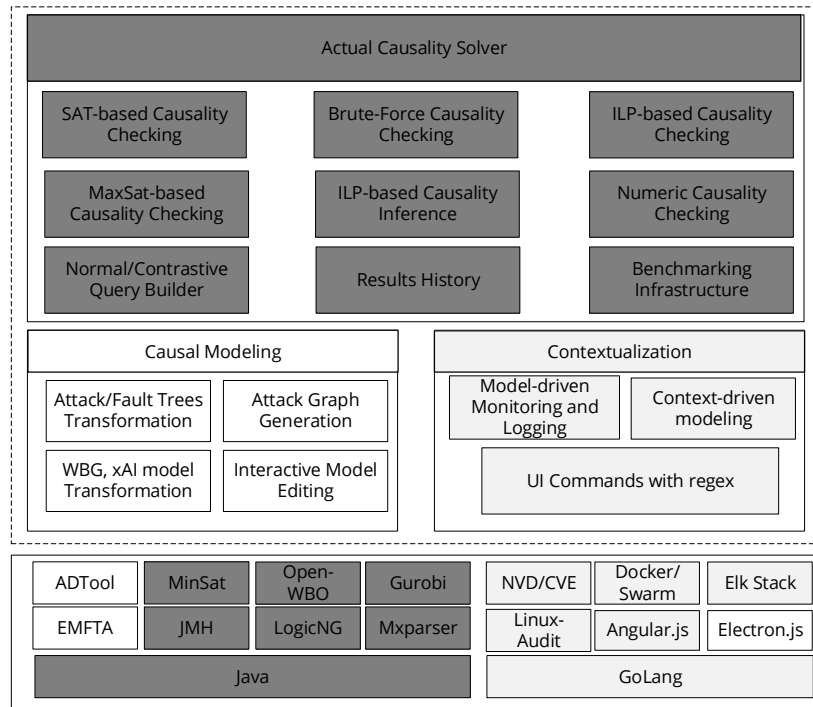


Figure 9.2: The Technical Framework of Actual Causality. The dashed box shows the toolchain we built, using existing tools and technologies shown in the diagram’s bottom part.

machine learning models and graphs of accidents’ investigations. These models will be shown as part of the use-cases in Section 9.4.

On the other hand, we contributed one tool to support the activities of contextualization. Although the concept of model-driven monitoring is general, the tool itself is specific to the domain of microservice-based systems. As such, the tool extends the state-of-the-art ecosystem in that domain (represented by the light gray boxes). Also, we explained the other type of contextualization, which is the context-driven modeling in Section 9.2.2. Without any specific tool support, an example of this type is given in Section 9.4.

### 9.3 The Actual Causality Canvas

Recall that we are interested in solving problems fixated around causality, such as finding out why a drone crashed or explaining a classifier result. Instantiating the three activities—model, context, and reasoning— we can solve such problems. We have seen that each

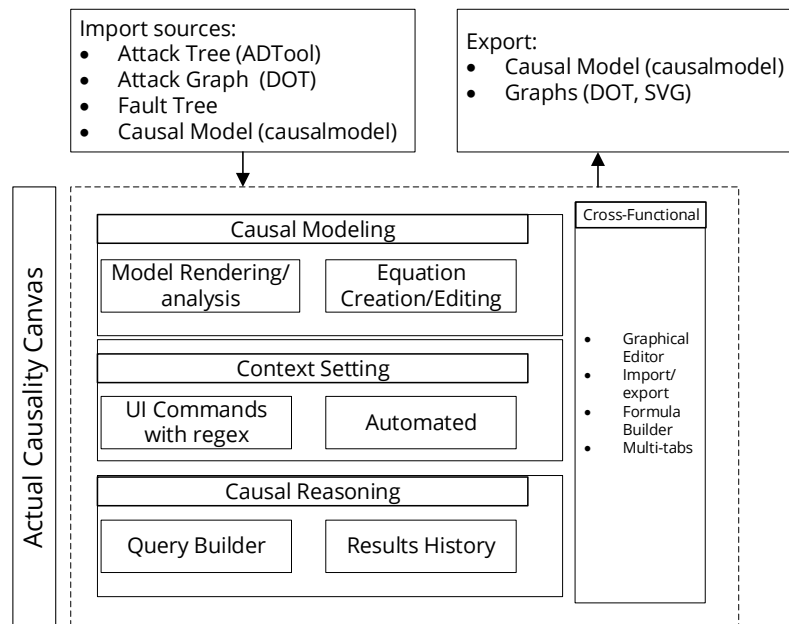


Figure 9.3: The Components of the Actual Causality Canvas

phase has its own challenges, methodological decisions, and tools. However, we believe that a unified platform with a plug-in architecture that enables each phase is crucial for deploying more causality socio-technical applications. Such a platform must be general enough to accommodate different practices, models, and queries. Thus, we present *Canvas*, an extensible, open-source, interactive platform that automates the abovementioned tasks and activities. It is in part a modeling tool that supports typical modeling activities and provides methods to transfer domain-specific models to causal ones. Also, *Canvas* allows an analyst to perform an interactive causal analysis of a particular event. Obviously, graphical editors are common in many domains; however, *Canvas*'s contribution lies in encapsulating the crucial tasks needed to specifically answer causal queries that are similar to the queries posted by humans. Figure 9.3 shows the main components of *Canvas*; the right-hand side shows the standard features for the three steps.

The generality of the modeling mode in *Canvas* stems from building it on the basis of Definition 2.2 (extension `.causalmodel`). We used a machine- and human-readable format (JSON) for this purpose; the newly created causal models are written and saved using this format. *Canvas* already transforms sources such as an attack tree modeled using ADTool [111], or a fault tree modeled using EMFTA tool [43], or any graph formatted using DOT. Intuitively, *Canvas* can also read already transformed or created causal models. The

import functionality is implemented using a plugin-based architecture, keeping the door open for an easy extension to include new sources of knowledge. Alternatively, wrappers can be written to generate a (.causalmodel) file directly from other sources. Besides model transformation, Canvas renders the models using different layout algorithms like *d3* and *dagre*. A set of visual and textual tools to create nodes, edges, and formulas *from scratch* are implemented. Furthermore, since human readability is a crucial aspect, Canvas is equipped with features that enable the user to grasp larger models by focusing on parts of the graph.

Context setting is enabled with a specific field in the (.causalmodel) format. Programmatically, the context can then be set by writing the values into the respective field in the file. Alternatively, Canvas provides a *command input function* that allows the user to set the values of the *exogenous variables* (context). This is implemented using a practical filter-and-set functionality that uses regular expressions to select the variables. Lastly, users can also edit their contexts as part of their query set-up.

For the reasoning mode (Figure 9.4), Canvas is an interface of the technical framework's solver. This tool offers different solvers for actual causality that differ in technology (SAT, ILP, MaxSAT) and accounting for responsibility with a minimal  $\vec{W}$  and either check or infer causality. The back-end is embedded with Canvas bundle, and it promptly answers queries (less than 6 s for models of 8000 nodes). The reasoning mode is activated using a specific button that displays a special screen for the query construction. The different elements of the query (context, cause, and effect) are easily manipulated on this screen. Once a query is ready, a request to the solver is sent, and the result is then shown back to the user. The result details whether each condition from Definition 2.4 is fulfilled or not, along with a  $\vec{W}$ . To realize the requirement of interactivity, Canvas tracks all causal queries in the same session; the user can navigate back and forth within them. This way, users can adapt their queries and play with different assumptions, contexts, and effects.

## 9.4 Use Cases

In this section, we show how the framework to operationalize actual causality is instantiated in xAI, accident investigation, and drone crash diagnosis. For each use case, we detail the modeling process, the context setting, and the reasoning aspect.

### 9.4.1 Explainable AI

Explanation and interpretability in AI (xAI) and machine learning are attracting attention because they are necessary for regulation compliance, system improvements, and trust enhancement [143]. The work on xAI focuses on model-based approaches that approximate the true criteria of classifiers as gradient-based (binarized like in [170]) or decision tree-based models [143]. Recent articles by Miller [141] and Mittelstadt et al. [143] suggest that existing approaches have not yet been built on relevant definitions from philosophy, social science, and cognitive science. Instead, they provide “general scientific explanations” The authors

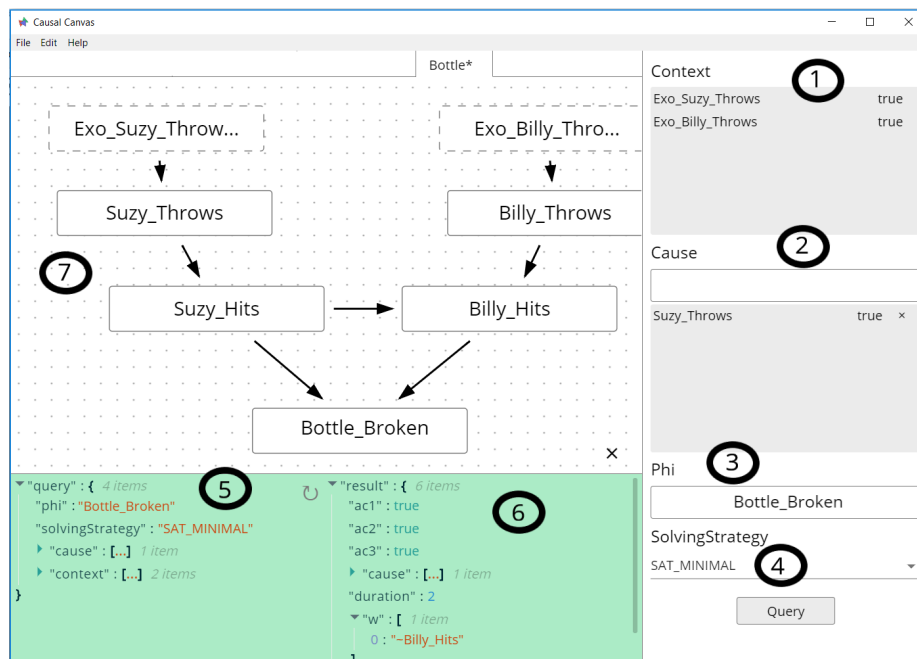


Figure 9.4: The Reasoning Mode in Canvas:  
 1- Context, 2- Hypothesized Cause, 3- Effect Formula, 4- Solving Strategy, 5- Query Request, 6- Query Result, and 7- Model

argue that humans would seek an “everyday contrastive, social explanation” that explains “why particular facts occurred” [141, 143]. Following Miller, we see HP as an enabler of such explanations. We understand that there is more to xAI than what we are suggesting here; however, we only show how explicit representations from the literature (like the one in our example) can be incorporated into Canvas, knowing that this is incomplete. We discuss the three requirements: a representation of the classifier’s behavior, i.e., a causal model; the exact information about the classified point, i.e., the context; and the reasoning machinery. We show one simple transformation to causal models and straightforward query construction.

### Causal Model

To illustrate this approach, we use an example provided by Miller [141] and highlighted again in [143]. The example (displayed in Table 9.1) considers the features and parameters learned by an algorithm to classify types of arthropods. Some features express binary facts such as whether an arthropod has a stinger or not while others are integer-based, e.g., the number of legs. We use a binary representation of the integer values. Although there may be other elegant ways to handle decimal features, we believe that a binary model captures what we want to explain. For instance, the fact that an arthropod has eight legs makes the

Table 9.1: A Lay Model for Classifying Arthropods [141]

Type	No. Legs	Stinger	No.Eyes	Compound Eyes	Wings
Spider	8	✗	8	✗	0
Beetle	6	✗	2	✓	2
Bee	6	✓	5	✓	4
Fly	6	✗	5	✓	4

algorithm classify it as a spider; the number *eight* is not needed in an arithmetic way.

Depending on the algorithm, different representations of the learned models exist. We use the one presented in the example: a tabular set of features and the values corresponding to one class. Each class and each binary feature is presented by an *endogenous* variable. To accommodate for its values, a nonbinary feature variable is presented by a set of bits. For example, the “number of wings” ( $N.W$ ) feature has the values 0, 2, or 4, and then it will be represented with *two* bit-variables,  $N.W_1, N.W_0$ . The different values of the two bits correspond to an index of the real values, i.e., 00 means 0, 01 means 2, and 10 means 4. We are not transforming the decimal values in the table into their binary representation but rather assigning them a binary value of some number of bits. The number depends on the count of distinct values presented. In the example,  $N.W$  has three values, and hence, we use two bits. This way, we use fewer variables. There are more rigorous ways to express non-propositional relations in boolean logic such as the Ackermann reduction, this approach can be seen as one-case of this reduction. We also express each feature (or feature bit) with an exogenous variable.

In addition to the variables, we create the propositional equations from the table. For instance, a *spider* is an arthropod with 8 legs, *and* not a stinger, 8 eyes, no compound eyes, and no wings. Then, each class variable is a conjunction of the features’ values, e.g.,  $spider = (no.wings = 0) \wedge (!stinger)$ . Note that  $(no.wings = 0)$  is presented as explained above. We create a causal structure (shown in Figure 9.5) of the factors leading to classifying an arthropod. Lastly, although not part of the example, the variance of the features’ importance (e.g., weights) is a candidate for a preemption relation. Using Canvas, we manually created the model; this could be automated with the import functionality.

### Context Setting and Causal Reasoning

We can now provide *explanations* for specific classifications. The specific case (e.g., an image  $J$  of an arthropod) is seen as a vector of features. This is precisely the *context* of a causal query. Formally, the context is an assignment of exogenous variables; we can easily see that it maps to the values of the features we consider. For example, *image J* contains 8 legs, no stinger, 8 eyes, no compound eyes, and no wings. What remains is the phrasing of the causal query. Miller [141] argues that the human perception of an explanation often refers

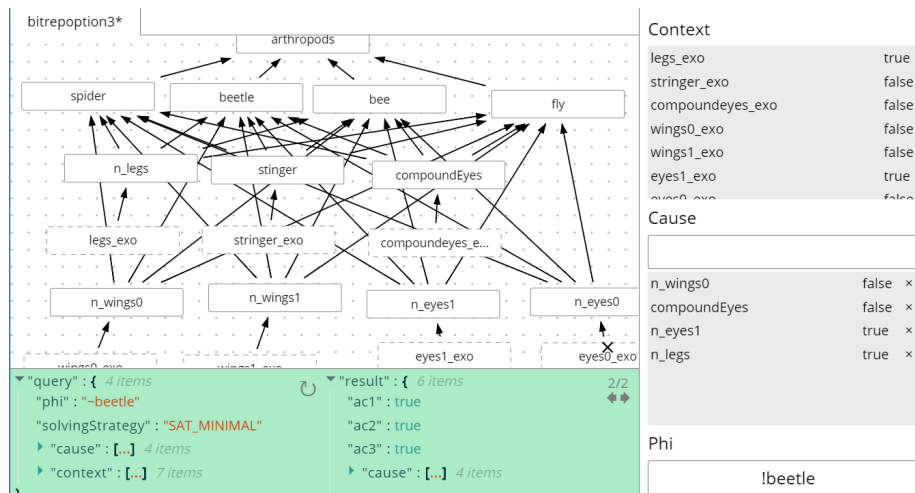


Figure 9.5: The Causal Graph of a Classifier. The context shows a specific feature vector.

to a contrastive question, i.e., why  $P$  rather than  $Q$  (e.g.,  $J$  labeled as a *spider* rather than a *beetle*). We think such a question can be formulated by focusing on what is the effect  $\varphi$ . The most basic contrastive question can then be seen as  $(\neg Q)$ , e.g.,  $\neg$ *beetle*. The result of the causal query can then be considered a contrastive explanation. For example, the answer to the question *Why is image J labeled as a spider rather than a beetle?* [141] is the list of causes (as seen in Figure 9.5): the compound eyes,  $N.W$ , number of eyes, and number of legs.

Obviously, the decision-tree nature of the example simplifies the AI part; there is tremendous work on the area of summarizing sophisticated AI models such as neural networks to similar models [170, 141]. However, with this example, our goal is to show the ability to incorporate ideas from the literature of approximating classifier behaviors into simpler models and augment it with contrastive reasoning capabilities. Note that if we want to consider image recognition as part of the explainable system, we then have also to create a causal model for it or expand the model in Figure 9.5 to describe its behavior.

#### 9.4.2 Überlingen mid-air Collision<sup>4</sup>

On the night of the first of July 2002, two aircraft collided near the German town of Überlingen.<sup>5</sup> The collision happened between a Tupolev Tu154M passenger jet (Bashkirian Airlines Flight 2937 from Moscow to Barcelona), and a Boeing 757-200 cargo jet (DHL Flight 611 from Bergamo to Brussels). A total of 71 passengers and crew members died in the accident [149]. The context of the collision comprised many confusing factors. We will summarize some of them; the complete list (sources: [149, 185], and [186] (German)) is shown in Table 9.2. The area of the accident is under the control of Zurich Air Traffic

<sup>4</sup>A shortened version of this case-study can be found in Appendix A

<sup>5</sup>[https://en.wikipedia.org/wiki/2002\\_Uberlingen\\_mid-air\\_collision](https://en.wikipedia.org/wiki/2002_Uberlingen_mid-air_collision)

Controller (ATC), who is in charge of keeping the routes clear. An ATC is equipped with a radar and a short term collision avoidance system (STCA), that alerts visually and aurally about 2-3 mins before a collision [178]. Also, both aircraft were equipped with traffic alert and collision avoidance systems (TCAS). TCAS warns the crew in aircraft in under 50 seconds before a collision, with a complementing resolution advisory commands (RA) to either climb or descends [178, 185]. According to the regulations, the ATC is responsible for keeping aircraft separated; TCAS is an additional system, and the regulations on whether to follow its advisory or not were not standardized yet in 2002.

On the accident day, a maintenance operation at Zurich ATC office deactivated STCA, ran the radar in degraded mode, and caused the telephone system (used to communicate with nearby airports and other ATCs) to run in fallback mode. Given the low density of flights at night, these limitations were approved by the managers. The ATC at Karlsruhe (Germany) was also monitoring the path of the collision with a fully operational radar. Due to the problem with the telephone system, ATC at Karlsruhe could not communicate with Zurich [178]. Additionally, ATC at Zurich spent around 5 minutes guiding an unexpected late flight to Friedrichshafen airport (Germany). The ATC was working alone that night because the operating company tolerated the case of taking long breaks at night [149].

Still, a collision can be avoided with all these systems installed. The ATC noticed the potential collision on the radar and instructed the Tu154M to descend flight level at 21:34:49; the Tu154M descends (21:34:56) exactly when TCAS generates an RA to the Tu154M to climb and to the B757 to descend. Then, the B757 descends (21:34:58) also, and the Tu154M crew discusses the contradictory commands (ATC to descend and TCAS to climb). The confusion is complicated by the fact that the Russian and the European procedures are not standardized in such a situation. Nineteen seconds before the collision, the ATC repeats descent advisory to the Tu154M, and wrongly advises the crew of traffic at "2 o'clock" while the DHL is at 10 o'clock. It is not clear whether the wrong location would have changed anything or not. The aircraft collided at 21:35:32. The original story has more factors that we omitted for simplicity.

With all these human, technical, and organizational factors, it is hard to draw conclusions. The official investigation by the German Federal Bureau of Aircraft Accident Investigation (BFU) was issued 2 years after the accident [149]. It concluded that both, the ATC (late intervention) and the TU154M crew (followed the ATC instruction contrary to the TCAS RA), made a series of mistakes that are considered as immediate causes, but the primary systematic cause is the negligence by the Air Traffic Control company of Switzerland. This was explained by the poor management of the maintenance event, along with the tolerance of allowing only one controller at night [149, 185]. They also pointed out the ambiguity in the handling TCAS commands, as a systematic cause. In 2005, researchers conducted a Why-Because-Analysis of the accident and presented a model that contained 95 factors [187] which we used in our work.<sup>6</sup> They reported an effort of 240 hours to design that model.

---

<sup>6</sup><https://rvs-bi.de/research/WBA/>



## Causal Model and Context Setting

The why-because-analysis (WBA) process is a systematic procedure to organize facts related to an accident [119, 120, 178]. The process results in a graphical understanding, called Why-Because-Graph WBG, of all the related facts and their causal relations. We can use WBG with some adaptations as a causal model (we discuss the differences in Chapter 10). The complete causal model is shown in Figure 9.6, and the description of each node in Table 9.2. In the following, we use  $e_i$  to refer to the event with ID= $(i)$  in the table. We consider each node in the WBG to be an endogenous variable. For each leaf in the graph, we create an exogenous variable that sets its value. The WBG embeds the context because it is created based on facts, thus the exogenous variables are always true [119]. To come up with the equations that describe each variable, we manually inspected each one to decide on its equation. Omissions are explicitly modeled in a WBG, and then we can directly use them with negation in the equation. For example, event  $e_{14}$ , *Air traffic controller not responding to B757 radio message*, is an omission event according to its description.

In disjunctive equations (events that disjunctively depended on other events), preemption relations are crucial to infer actual causality [95]. These relations are especially important in contexts where events coincide. Some preemption relations express temporal order of events, but others may reflect a discrepancy of the causal importance among events. For example, in the rock-throwing example, the fact that Suzy threw the rock slightly earlier is modeled using a preemption relation. We use the same concept to edit the WBG to add preemption relations among the events leading to  $e_6$ : *Conflict resolution failed*. This discrepancy merely reflects the fact that keeping the routes clear for aircraft is the mission of the ATC [149]. Accordingly, the TCAS is a last resort that should resolve last-minute situations, and hence, causally, a failure by the ATC *preempts* a failure by the TCAS.

Understandably, a large part of the WBG focused on the factors of the late intervention of the ATC ( $e_{49}$ ). Five direct factors coincided and led to the state of late intervention, namely:  $e_{52}$  *Control strips do not warn of crossing routes on the radar*,  $e_{53}$  *No visual warning from STCA*,  $e_{55}$  *21:35:00 Acoustic STCA signal was not detected in control room*,  $e_{56}$  *Heavy load on the ATC*, and  $e_{62}$  *Crossing routes*. Each factor can be thought of as a sufficient cause of  $e_{49}$ ; however, when thinking of actual causes, people tend to consider *exceptional* events to be the probable causes [75]. Then, we argue that the exceptional heavy load on the ATC (due to another late landing in a nearby airport, and the faulty phone system) is more influential than the normal technical problems with STCA system ( $e_{52}$ ,  $e_{53}$ , and  $e_{55}$ ). Also, having a potential route crossing ( $e_{68}$  B757 3 minutes past expected time,  $e_{69}$  Tu154 2 minutes ahead of expected time) is plausible in aviation. Accordingly, we added preemption relations among these events.

Our sole aim from these steps is to show that a causal modeling methodology, like WBA, quickly yields comparatively large models. In some cases, the steps above seem biased or forced, but they explicitly represent the investigators' knowledge as documented in their report [149].

## 9 A Framework for Operationalizing Actual Causality

ID	Description	ID	Description
(2)	Crash B757	(63)	optical STCA not active
(4)	B757 vertical tail destroyed	(64)	no correlation of flight plan data and radar data
(1)	Crash Tu154M	(70)	MV9800 computer is not available
(3)	Fuselage Tu154 severed	(79)	Radar system in fallback mode
(5)	Collision	(91)	System work in the ADAPT system
(6)	Conflict resolution failed	(41)	ATC must restore separation
(13)	Conflict resolution by air traffic controllers failed	(44)	The role of the air traffic controller to ensure separation
(7)	Resolution of conflict by crews failed	(35)	TCAS training DHL
(8)	Conflict resolution by TCAS failed	(22)	B757 TCAS RA only reports 23 sec after RA
(12)	Tu154 controls against TCAS RA	(31)	Only one radio channel for Tu154 and B757
(29)	B757 complies with TCAS RA	(57)	Radio messages from AeroLloyd 1135 to ACC Zurich
(11)	TCAS does not reverse RA	(75)	Land approach from AeroLloyd 1135 to Friedrichshafen
(18)	Avoidance of unnecessary RAs	(71)	Technical limitations of the workstation in relation to radio
(14)	ATC not responding to B757 radio message	(77)	Handover procedure to Friedrichshafen telephone based
(21)	ATC not responding to B757 maneuver	(76)	Pilot does not successfully use any of three telephone systems
(19)	Avoidance maneuver of the Tu by sinking	(61)	Requirements for warning of crossing routes not met
(20)	Evasive maneuver of the B757 by sink	(87)	Night staffing
(30)	Radar system does not display transponder S information	(85)	Assumption: attempt to retrieve timetable
(33)	PIC encourages PF to descend	(101)	ATC approves deviation from course
(32)	TCAS Tu154: climb	(37)	TCAS detects danger of collision
(62)	crossing routes	(45)	Criteria for TCAS RA fulfilled
(69)	Tu154 2 minutes ahead of expected time (control strip)	(43)	Aircraft below staggering
(68)	B757 3 minutes past expected time (control strip)	(46)	Further approximation of aircraft
(84)	delayed departure	(47)	B757 follows original flight path
(78)	Deviation from planned exchange rate	(48)	Tu154 maintains rough flight direction
(38)	Training PIC BTC	(50)	approx. 21:35 Tu154 starts right turn
(36)	Decision-making of the PIC	(51)	approx. 21:33 Tu154 changes course to left
(34)	TCAS B757: descending	(59)	Assumption: Crew mainly concerned with conflict situation
(72)	Understaffing in the ACC, only one instead of 4 controllers	(58)	Autopilot switched off
(80)	Rest of the second controller	(60)	unknown cause
(81)	2 controllers in control room instead of 4 during day shift	(10)	Visual identification of conflict traffic does not solve conflict
(94)	Sectorisation work	(16)	B757 does not understand the conflict traffic message.
(92)	Replacement telephone had to be switched	(15)	contradictory conflict information in the Tu154
(93)	Changeover to telephone system	(17)	B757 does not recognize flight maneuvers of conflict traffic
(82)	Replacement telephone of ATC not in working order	(27)	Perceived size of conflict traffic
(89)	faulty switching of the replacement telephone	(26)	Night flight affects visibility and accuracy
(83)	no release of the service telephone after conversion	(25)	B757 visually identifies conflict traffic at 2am
(73)	ATC had a System manager at their disposal as an assistant	(23)	Tu154 visually identifies conflict traffic at 10am
(74)	ATC was not aware that he had an assistant at his disposal.	(24)	ATC: "rapidly sink to FL350, conflict traffic 2 o'clock."
(86)	Common practice in ACC Zurich	(88)	Usually low traffic during the night
(49)	ATC detects crossing routes too late	(42)	Misinterpretation of the radar image by air traffic controllers
(56)	Heavy load on the air traffic controller	(95)	Bypass system cannot be used
(53)	No visual warning from implement	(96)	Mobile phone not used
(55)	acoustic STCA signal was not detected in control room	(99)	Telephone Switch-02 is not used
(52)	Control strips do not warn of crossing routes	(100)	Telephone Switch-02 not operational
(66)	ATC does not request support from System manager	(98)	Emergency manual lists 3 telephone systems available
(65)	ATC checks at two workplaces	(97)	ATC not aware of mobile phone availability
(67)	Transfer of landing approach from AeroLloyd 1135 to Friedrichshafen not successful		

Table 9.2: Accident's Facts; Originally in German [187]; ordered as they appear in Figure 9.6

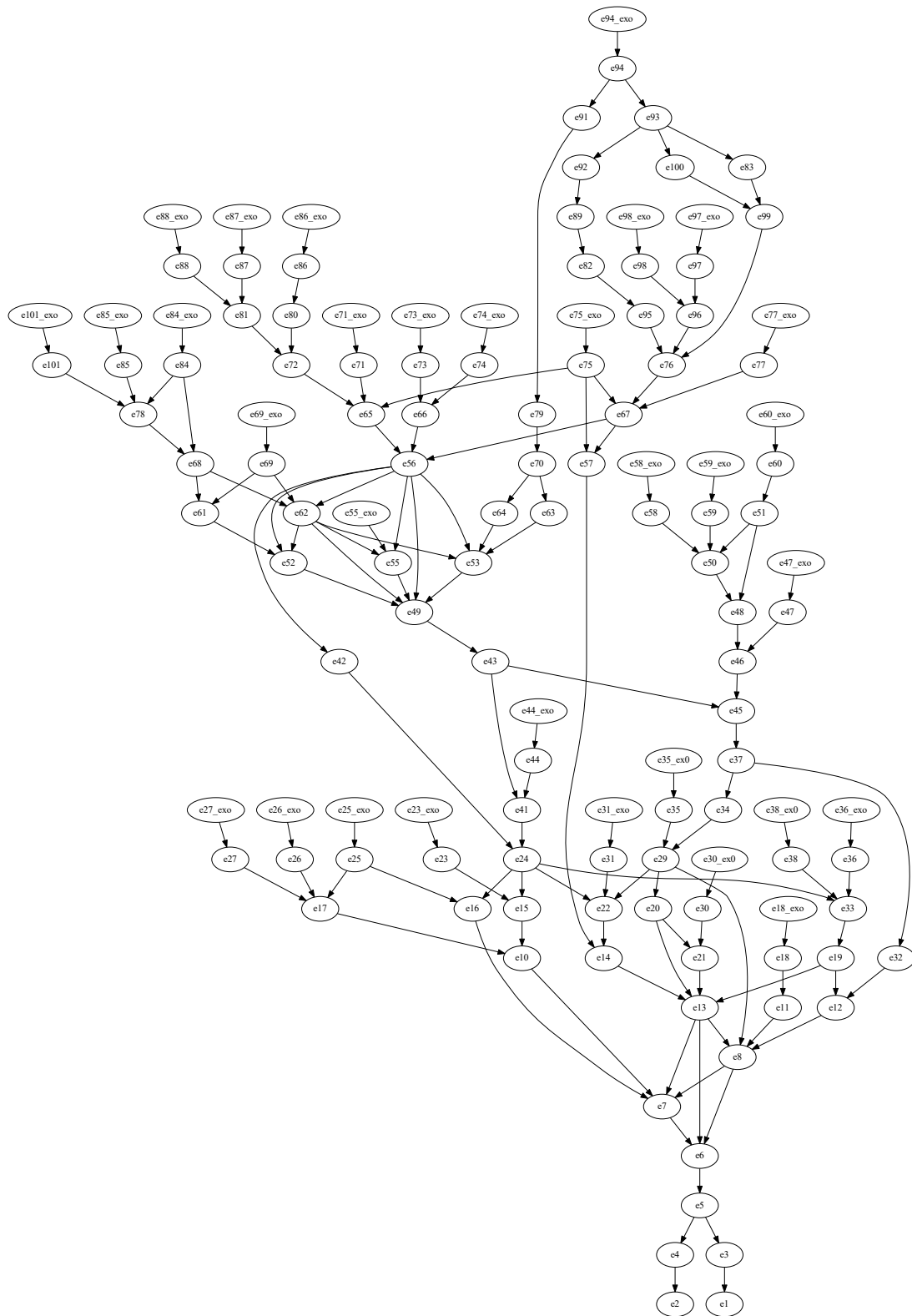


Figure 9.6: Causal Network of the Überlingen accident [187]; events are described in Table 9.2

### Causal Reasoning

With the knowledge of causal factors made explicit using a model, we now can use the actual causality definition to check for cause(s). It is worth noting that, at least according to HP, multiple causes of an effect are possible [75]. Other concepts can then be used to compare such causes, such as responsibility [33] and normality [77]. As a first causal check, we used a simple causal model that expresses all the relations as disjunctions, i.e., assumes that any factor is enough to cause the other side of the connection. Especially in confusing situations (in which events have coincided), such a model is not conclusive or over-determined. This check replaces the manual verification step of the Why-Because analysis [120], in which the model is checked against a sufficiency test to verify that the effect eventually happens, given that all the root causes (graphs leaves) occurred. We performed this check:  $Q_1$ : *Is  $\vec{X}$  a cause of  $e_5$  (collision)?* where  $\vec{X}$  is the set of 31 leaf events. The check passed the three HP conditions. This check shows that HP can be a part of WBA.

The interesting checks were performed on the *edited* model (with preemption and logical combination). The first check was the same as  $Q_1$  (the effect is the collision, and the cause is a set of 31 root causes). The result was a violation of AC3, i.e., the cause is not minimal. A minimal cause of 14 variables was returned by our solver. These are the details that resulted in the late intervention of ATC. This actual cause conforms with the immediate cause reported by the BFU [149]. However, this check is fine-grained. For example, one of the root causes in the check is  $e_{85}$  *attempt to retrieve the timetable*, which is assumed to have delayed the take-off of the DHL flight. Thus, we checked the actual causality to find a minimal cause on a coarse-level. The question this time is  $Q_2$ : *Is  $\{e_{13}, e_{70}, e_{74}\}$  a cause of  $e_5$  (collision)?*; the set of causes are chosen arbitrarily to represent different levels of granularity about the ATC where  $e_{13}$  *Conflict resolution by ATC failed*,  $e_{70}$  *MV9800 computer is not available*,  $e_{74}$  *ATC was not aware that he had an assistant at his disposal*. The result was that this is not a minimal cause and the minimal one was only  $e_{13}$ . Thus, at a coarse level, we conclude the failure of the ATC as an actual cause; this can be further explained into detailed events as we saw in  $Q_1$ . Note that we omitted the  $\vec{W}$  set for simplicity's sake in this description. Similarly, we conducted checks focusing on an intermediate event  $e_{49}$ : *Air traffic controller detects crossing routes too late*, as an effect. On the fine-grained level, we found an actual cause comprised of 11 leaf events (root causes) that conformed with the BFU systematic causes of the accident. For example  $e_{86}$ : *the common practice in ACC Zurich* and  $e_{77}$ : *Telephone-based handover procedure to Friedrichshafen*. On a higher level, we found an actual cause of three events ( $e_{65}, e_{66}, e_{67}$ ); they mainly lead to the load on the ATC.

#### 9.4.3 Malicious Insiders<sup>7</sup>

From a different domain, we consider an example of malicious insiders. Insiders are among potential attackers of information systems, although they are mostly not malicious.

---

<sup>7</sup>This is a shortened version of the example in Chapter 6. The reiteration here is meant to put accountability of microservices-based system in perspective with other domains.

However, reports [169] show that their attacks are the most significant and lengthy to detect. The problem is that preventive measures have a high likelihood of failing because insiders ought to have sufficient privileges for their jobs. Thus, accountability in the sense of attack attribution is potentially a deterrent measure. In this use-case, we consider an example from an industrial partner in the domain of micro-services.

### Causal Model

Attack trees [179] and attack graphs [182] are widely used to model attacks on a system; they are a promising starting point to create causal models. Mainly because they are an acyclic propositional combination of attack strategies. Typically, they do not include the attacker, but since we are considering insiders, we can add them to the models. We use the formal transformation of attack trees to causal models presented in Chapter 6, in which each node in the attack tree is transformed into an endogenous variable, and each leaf node is set by a new exogenous variable. This feature is automated in Canvas with an import option. We provide a method to augment the model with preemption relations among insiders. The relations are based on a *suspiciousness metric* (SM) that is related to the modeler's judgment of the case.

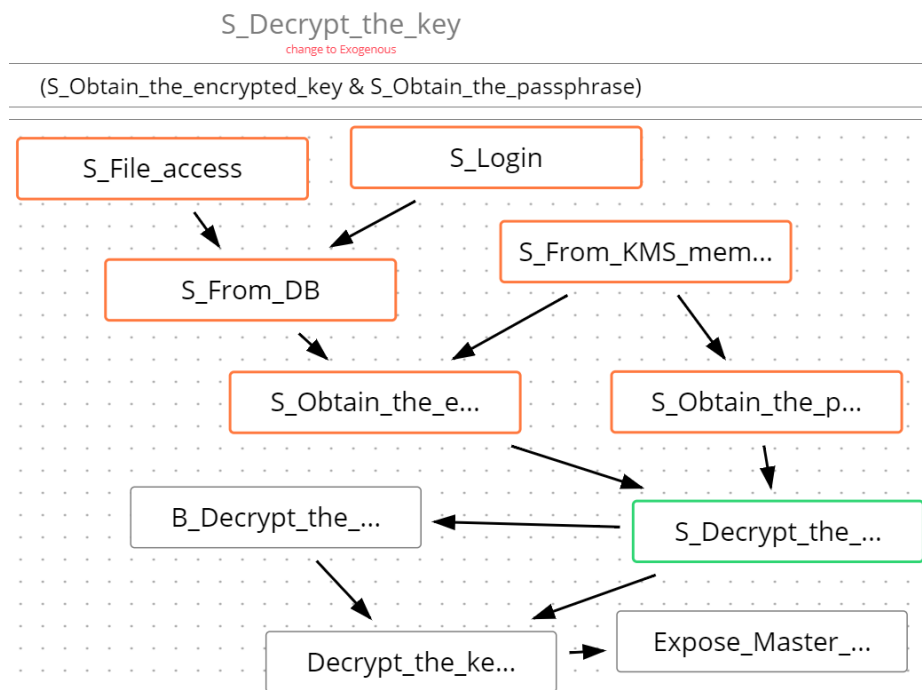


Figure 9.7: The Insider Model in Canvas

We used a model by importing the corresponding attack tree (provided as part of an

analysis step in a company). The model describes an insider behavior that leads to *stealing a master encryption key* in a production environment. An excerpt of the model is shown in Figure 9.7; it highlights one variable *S\_Decrypt* and its equation. The figure represents one way to expose a master key by obtaining its encrypted version. This strategy can be executed by one of two insiders Suzy (*S*) or Billy (*B*).

### Context Setting and Reasoning

We tested one scenario, in which Suzy stole the key. We simulated the attacks in a micro-services environment. To set the *context*, we utilized monitoring tools like auditD to monitor file accesses, and Couchbase audit to monitor queries. We used the logs to set the exogenous variables. For example, a sentence from auditD like ... "MESSAGE" : "PATH name=.../script.txt"..*..aid= 1001 uid= 1001..* is translated into *S.From\_Script\_exo= 1* (Suzy's id=1001). As a query we checked *Q1: is Suzy the cause of stealing the key?* The scenario represented a situation with coinciding possible causes. The results matched our ground truth, i.e., we concluded that Suzy was responsible for the incident.

#### 9.4.4 Drone Crash Diagnosis

Drones, such as quadcopters, recently found widespread use; however, their safety is a significant concern. In the case of drone failure, it is essential to identify the cause and prevent it in the future. In this use case, we consider a realistic example where Canvas is used to model a system from scratch and assist in investigating incidents.

Drones have several physical and software components, including actuators, sensors, and controllers. The components that interact with the physical world are called *actuators*, e.g., the electrical engine. *Sensors* are devices that measure physical properties; for example, GPS measures the location and altitude. Finally, *software components* are virtual units that organize all hardware components and process the information to keep the drone stable in flight. For example, the sensor fusion module receives readings from sensors and estimates an approximate value based on the readings. The course of a flight comprises several coinciding events related to different components. The diversity of such events and their causal connection render the diagnosis difficult. As we see in this use case, Canvas is a practical method, especially when investigation from scratch.

### Causal Model

Each node in our model describes an action of a specific component, such as the failure of the GPS sensor or when the drone was being pushed by the wind. The provided model is abstract to keep it understandable; each node can be decomposed into more detailed events. The model is built based on domain knowledge or data-driven approaches. In previous work, we deduced a *fault tree* from the drone's architecture [205]. Here, we use a similar fault tree while adding preemption relations based on the results from a practical

course we held with computer science students [206]. The students used Canvas to create their causal models. The preemption rules originate from the nature of the control loop that is being executed repeatedly during flight. In this sense, the failure of the actuators preempts that of the controller software, which then preempts the failure of the sensors. Moreover, among the software components, path tracking failure preempts path planning failure. After adding these relations to the fault tree (imported to Canvas), we obtain our model depicted in Figure 9.8.

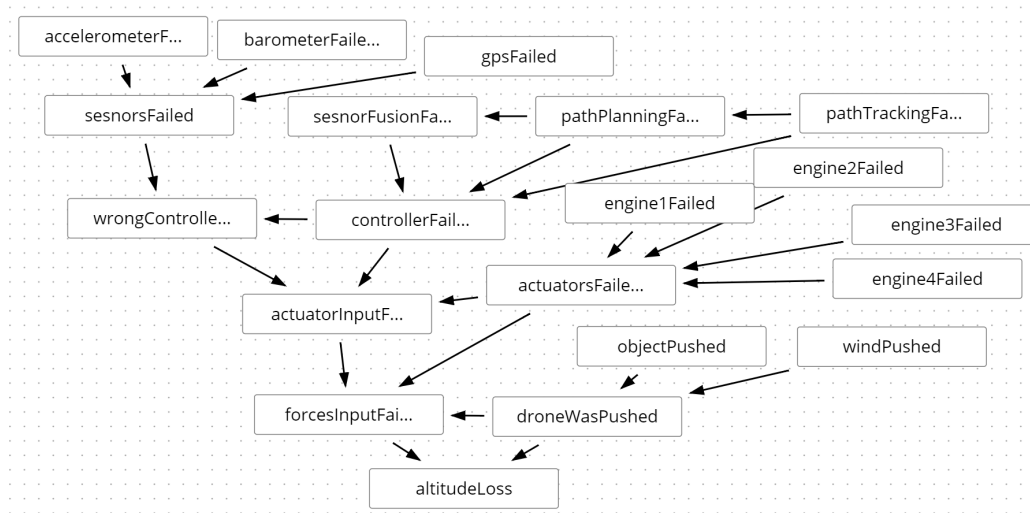


Figure 9.8: Causal model for Drone Crash

### Context Setting and Reasoning

All the nodes in our causal model are events. Simply put, an event describes an action performed by a specific component. For example, the number of detected satellites by the GPS sensor dropping below 9 is an indication of a *gpsFailed* event. For context setting, first, detection analysis should be run over the data to detect the events that occurred. If an exogenous event is found in the flight logs, its value is set to true in Canvas. The endogenous variables, such as *sensorsfailed* or *actuatorsfailed*, will be computed based on their respective equations implied by fault tree semantics. We consider two scenarios based on real flight logs collected from users of an open-source quadcopter [57].

In the first scenario, we analyze a case of engine failure that resulted in a crash. Engines have an essential role in keeping the drone in the air, and their failure leads to altitude loss. If the commanded signal to an engine is set to the maximum value for more than a second, then we can assume that the engine has failed. We set *engine1Failed* to true on the basis of our assumption. Moreover, *altitudeLoss* can be detected when the altitude drop rate is more intense than a threshold. This is seen in the mentioned flight log. Other nodes such as

*accelerometerFailed* or *windPushed* are set to false either because they did not occur in that specific log, or there were no relevant sensors to record them. Now, constructing a query with a hypothesized cause *engine1Failed* for *altitudeLoss* returns true. Other hypothesized causes, such as *gpsFailed*, result in a negative response in Canvas.

In a second scenario, which was also seen in the real flight logs, both *the path tracking and path planning* modules of a drone failed. Our objective was to query which one was the actual cause. We set the value of these two events to true. Also, the value of *altitudeLoss* is set to true since this event occurred according to the log. Although it is not trivial that *pathTrackingFailed* is the actual cause, we can easily deduce this using Canvas. This is because of the preemption relation between the *pathPlanningFailed* and *pathTrackingFailed* nodes. This preemption rule originates from the domain knowledge where path tracking is closer to the final physical output of the drone than the path planning module in the control loop.

Although the logs are not labeled, i.e., the causes are not known to us, the added value of Canvas lies in its ability to import fault trees and compute the causality in large models where it is nonintuitive for the investigator to deduce causality between events. Moreover, when several events coincide, Canvas distinguishes between their causal roles using the embedded preemption relations.

### 9.5 Evaluation

In addition to utilizing the framework and Canvas in different use cases, we briefly report our evaluation of Canvas.

**Display performance.** We tested how Canvas performs when displaying different, randomly generated models. The models vary in size between 10 and 4000 nodes. We tested critical functions such as the graph layout, graph navigation, and zooming. All tests were executed on an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 2001Mhz, 4-core(s) Processor, and 16GB RAM. Canvas uses two algorithms to lay out the imported graphs, *dagre*, and *d3-force*. For the quality of rendering, the *dagre* layout is suitable for smaller causal models (fewer than 50 nodes). The nodes here are nicely separated with sufficient space. The *d3-force* layout is suitable for almost any model size up to 600 nodes. For the rendering *time*, the layout methods differ as well. *Dagre* is usable only for models smaller than 400 nodes, whereas *d3-force* responds within a reasonable time with models up to 4000 nodes (taking around 8 s to render). Canvas automatically switches its layout method on the basis of the model size. The other features perform well with causal models up to around 800 nodes, after which some lag is seen with zooming and navigation. As human cognition is limited, operators will not grasp large models. Thus, we verified that Canvas facilitates focusing on parts for models up to 4000 nodes. We manually tested inspecting such models by highlighting and zooming into their parts. This is an effective feature where the user highlights a reachable part of the graph based on a filtering function. Of course, we do not see a point in using a graphical editor with very large models, but the goal is to document



the performance of Canvas.

**Usability.** We also conducted a preliminary user study with 10 computer science graduate students (working in five groups). As part of a practical course, they used Canvas to develop their custom causal models and queried the actual causes based on real drone crash data [206]. An overview of the HP definition, illustrated by examples bundled in Canvas, was sufficient for the students to grasp the concept of causality and use it in their drone crash scenarios. We collected the students' feedback through written forms and face-to-face interviews. The five groups reported an *effective* usage of Canvas for the task. Some of their feedback included "*powerful utility that is easy to operate*" and "*a great tool to quickly analyze if the proposed model is indeed correct.*" They stressed its power, especially when dealing with large models or confusing situations. The students also suggested some enhancements that we are considering such as an undo feature, a custom highlight of preemption relations, and a programmable interface that exposes all phases to other systems.

## 9.6 Summary

This chapter provides a unifying framework that generalizes existing approaches to accountability and explainability, which applies to different contexts. As modern systems could harm people, damage their assets, or decide their loan adequacy, such systems ought to be at least explainable. To that end, our framework is intended to solve explanation-based problems for a wide range of systems in the future. Advancing operationalizations, the framework is bundled as an interactive platform. We have shown how different knowledge sources can be transformed into structural-equations models and then used for an automated analysis—using HP actual causality. We conclude that our framework is generalizable enough to accommodate explanation-based socio-technical constructs, and with tool support, it is amenable to be incorporated into different domains.



## **Part V**

# **Related Work and Conclusion**



## 10 Related Work

*This chapter reviews the related work in the fields of accountability, actual causality reasoning, and other related fields. Parts of this chapter have been published in the following publications [167, 98, 97, 96, 94, 95, 99], co-authored by the author of this thesis.*

### 10.1 Accountability

The term *accountability* has been described in different ways, especially in the domain of data usage (for instance, [145, 24, 200]). Kacianka et al. provide an overview of the literature on accountability [105, 103]. In computer science, the term accountability was popularized by Weitzner et al. [200]; the authors proposed that *accountability* is ensuring appropriate usage of information by providing the ability to determine how it has been used. This approach is an alternative to restricting access to information. Following Weitzner et al., the research community focused on accountability as a privacy mechanism or as a property of cryptographic protocols. With the rise of cloud computing, privacy concerns were naturally applied to the cloud, and accountability was seen as a way to ensure data security, e.g., by Pearson et al. [159]. As we described in the introduction of this thesis, we consider accountability as a valuable property of a system in general and not only limited to the usage of information. We refer to the definition used in [13]. The authors think of *accountability* as a capability of socio-technical systems to answer questions regarding the cause of occurred unwanted behavior. We see accountability as a mechanism that aids in understanding and enhancing systems' operations.

We see a lack of technical tools and practical methodologies to enable accountability. In a mapping study, Kacianka et al. surveyed accountability implementations [103]. They found that none of the surveyed papers have evaluated their tools for performance. This is important because a key factor that could limit the adoption of accountability mechanisms, in addition to the lack of established methodologies, is performance efficiency. The reason is that the origin of unwanted events is typically tracked using logging and analysis of "interesting" system events. Depending on the complexity of the analysis algorithm and the size of the logs, accountability implementations could be costly in terms of computation. Another gap they report on is the missing link between the high-level unwanted events that take place in an environment (e.g., personal and medical data is leaked in a Healthcare domain application) and the low-level unwanted events that are logged in the running

technical systems (e.g., system calls reading from confidential files and writing to a socket in a network connection).

**Gap 1** *In summary, we see a gap in the literature around accountability regarding the lack of methodologies and tools to enable accountability in digital systems.*

## 10.2 Causality

In general, we distinguish two notions of causality, namely *type causality* and *actual causality* [75]. Type causality describes general causal relationships, whereas actual causality examines causal relationships among specific observed events. To get a general overview of the literature on causality, this section briefly reviews essential literature in this field.

Scholars have been struggling with formulating a definition of causality since Hume's proposal [92] in the 18<sup>th</sup> century. With his formulation, Hume set the foundation of a *counterfactual* approach to causality. Hume's notion of counterfactuals leads to the counterfactual definition of causality, which can be formulated the following way: *A* is a cause of *B*, if, if *A* had not happened, *B* would not have happened either [74]. Another important proponent of counterfactuals was David Lewis [127], who extended the notion by the possible-world semantic using intensional logic. Lewis also assumed the transitivity of causal relationships. However, the approach has appeared to have many problems with famous examples (some given by Wright [203]). There has been other criticism of a mere counterfactual definition, like Dawid [41] and Collins et al. [35].

A number of actual causality definitions were formulated to cope with the cases that challenged Lewis's definition [127]. Those included: the *causal beam* definition by Judea Pearl [156], Hall's *H-account* [72] and Hitchcock's definition of actual causality [86]. All three definitions are based on the idea of counterfactuals but add further mechanisms dealing with problematic cases.

The aforementioned definitions influenced the definition that we use in this thesis, i.e., The *Halpern-Pearl Definition of (Actual) Causality*. The first version of HP was published in 2001 [78]. It was updated due to a problem with the first version in 2005 [82]. Additionally, the second part of the 2005 paper introduces *explanations* as a principle for causal reasoning with uncertain or unknown information [83]. In 2015, the HP definition was modified by Halpern again, yielding a shorter formulation dealing better with some special cases [74]. The HP definition has been taken up in a number of computer science publications (e.g., [117, 123, 15]). A mental model of the relationships between the literature around HP is depicted in Figure 10.1, which serves as a reference guide.

### 10.2.1 Actual Causality Reasoning According to HP

A central goal of this thesis is to advance the utilization of actual causality definitions to enable accountability for different system qualities. As such, in this section, we discuss all

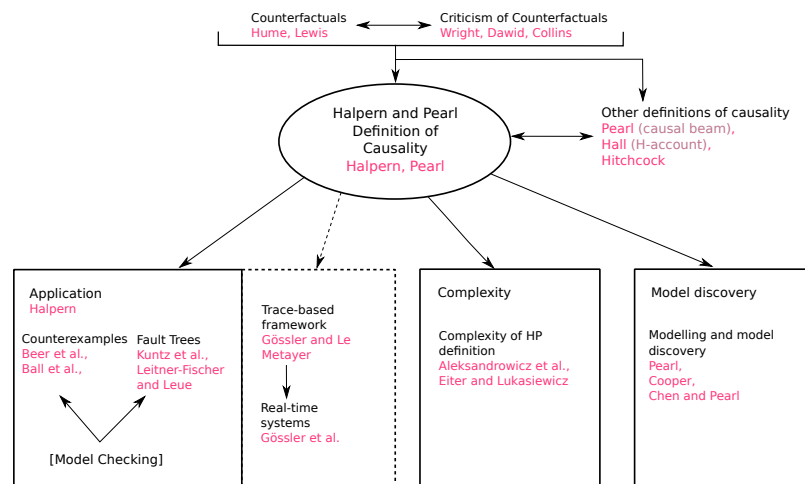


Figure 10.1: Overview of literature on (actual) causality and their relationship

the references to HP in the literature. An abstract overview of some of these references is presented in Halpern’s book [75] (chapter 8). We, on the other hand, present a thorough summary of each field, focusing on the computational adaptations made in each field. We use the modified HP definition because it solves problems described with the previous versions, and reduces the computational complexity [74]. To the best of our knowledge, no previous work has tackled the technical implementation of the (*modified*) version of HP. Conversely, the first *two* versions were used in different applications. Although they use different versions, we still consider them related. As we shall see, these applications used the definition as a refinement of other technologies.

First, in the domain of databases [134, 136, 18, 177], a simplified HP (the updated version) was used to refine provenance information and explain database conjunctive query results. This line of work shows how causality can be applied to explain why a database query answer does or does not contain certain results [135]. For that, the authors introduce a definition of functional causes based on the *updated* HP definition [81, 83]. This work was extended in [134, 136] when the authors generalized their definition to answer what caused the specific output of a query in a database; and showed the complexity results [136]. In [138], the authors bring their previous work together and explain how causality may support data provenance. In [137], the authors refine their concept of causality in databases to a new approach, which they call view-conditioned causality. This concept has already been mentioned in [134] as well.

In summary, the work in database causality is fully tailored towards database queries and thus a particular application of the HP definition. All the proposed approaches depend heavily on the correspondence between causes and domain-specific concepts like lineage, database repairs, and denial constraints. In contrast to our work, they do not directly

implement and apply the definition, but also simplify it. The simplification in this line of work is the limitation to a single-equation causal model based on the lineage of the query in [134], or no explicit equations but rather the authors infer a cause from the computation of a database repair in [18, 177]. Also, they eliminate the treatment of preemption using contingency sets. Similar simplification has also been made for Boolean circuits [34].

Second, in the context of model verification, a concept that enhances a counterexample returned by a model checker with a causal explanation based on a version of the updated HP is proposed in [16]. However, they are adapting the definition such that dependencies between variables are not considered. Thus, their domain-specific simplification comes from the fact that no dependencies between variables, and hence no equations, are required. Moreover, they used the definition of singleton causes. Their approach is integrated into a tool called *RuleBase PE*, a formal verification platform by IBM.

Similarly, some researchers proposed to generate fault trees out of probabilistic counterexamples determined by a model checker using causality analysis [117]. Thus, they modify the *updated* HP definition to consider the ordering of events. This work was extended in [124] to provide formal proofs and refined the definitions such that their approach can be integrated into model checkers and applied on transition systems representing concurrent systems. The integration into model checkers as compared to the analysis done in [117] leads to a performance improvement regarding runtime as well as memory consumption. An implementation of this approach is made in the *SpinCause* tool [125]. The authors of [14] further extend this line of work. The authors implemented different flavors of causality checking (based on the updated HP) using Bounded Model Checking to debug models of safety-critical systems. However, in contrast to our work, they employed SAT solving indirectly in the course of model checking.

All the aforementioned HP applications use acyclic binary models; they depend on older versions; augment their domain-specific technologies with causality such as Kripke structures, lineage formula, or counterexamples. These applications, to our knowledge, adapted the theory to a domain-specific context, with simplifications and restrictions on the definition. For instance, they relax the definition of counterfactuality by either removing the mechanics of the model, i.e., no-equations model, one-equation model, or by considering singletons only, i.e., the cause is one tuple. Thus, the complexity of the causality definition is relaxed, either because AC2 is more straightforward (no  $\vec{W}$ ) or AC3 is not needed. Therefore, in contrast to our approach, these applications can not be used outside their respective domains due to the restrictions or dependency on other concepts. Obviously, that is sufficient for the particular use-case, but we argue that a general approach towards actual causality may enable new socio-technical applications; such an approach is lacking in the literature. In contrast, the approaches presented in this thesis are general causality reasoning algorithms. Hence, in the evaluation, we use examples from different domains of philosophy, security, safety, accident investigation, and explainable AI. We provide methods that support the effective usage of the formal definition without compromising the generality of the theory. We employ optimization problem solving, which was not utilized



before in this context. Alternatively, previous work used SAT indirectly [14], or answer set programming [18]. To prove the complexity classes, Halpern [74] used the relation between the conditions and the SAT problem. However, the actual encoding in SAT with a size that is linear to the number of variables is still missing. We close this gap in Chapter 3. Lastly, to the best of our knowledge, no previous work tried to address actual causality in non-binary models. As part of this thesis, we propose a method to formulate a causal query in numeric models as an optimization problem.

Similar to our aim, [89] Hopkins (in 2002) evaluated search-based strategies for determining causality according to the original HP definition. Hopkins proposed ways to explore and prune the search space for computing  $\vec{W}$ ,  $\vec{Z}$  that were required for that version, and considered properties of the causal model that makes it more efficient for computation. The results presented are of models that consist of less than 30 variables; in contrast, we show SAT-based strategies that compute causality for models of thousands of variables.

### 10.2.2 Non-HP Causality Reasoning

Gössler and Le Metayer [66], propose a general framework for causality analysis of system failures based on observed traces and behavioral models. The authors differentiate their approach from HP in two ways. The first difference is their utilization of traces as first-class citizens in their approach, whereas HP requires mapping the abstract notion of events to properties. The second difference is regarding (utilizing) the temporal ordering, which is not explicitly regarded in HP. Their work resulted in some publications on causality in real-time systems([64],[199]). While we think this is an interesting approach, we still think that coming-up with fully specified behavioral models of the systems is a daunting task (especially in domains other than real-time systems). In contrast to this approach, we aspire to generalize causality into more domains once we have it as a technical component.

In recent work, LeBlanc et al. have a similar aim to ours in advancing efficient actual causation applications [121]. The authors criticize the counterfactual criterion of a cause. They argue that a causality approach should distinguish the laws of a situation's state and the events that cause the states to change. That work is similar to approaches such as [12] in which languages like situation calculus or action and change languages are proposed to compute actual causality. While a comparison of causality definitions is beyond the scope of this thesis, we think that one comprehensive general definition of causality is an ambitious goal. Thus, we believe that our approach facilitates such comparison by efficiently computing causality according to the HP definition. Further, Mark Hopkins [90, 88] studies the shortcomings of the structural equations models. Specifically, he emphasizes the expressiveness of this language and how useful it would be to use situation calculus. Although this is interesting research, we are focusing on structural equations.

Fault tree analysis (FTA) is an established design-time method to analyze the risks related to the safety and reliability of a system [175]. Similar to HP, FTA is a model-based activity. It comprises a model, i.e., a fault tree, and quantitative and qualitative analysis tools. The primary qualitative approach to analyze a fault tree is the computation of

minimal cut sets MCSs; a cut set (CS) is a set of events that, taken together, lead to the top-level event. These sets provide insights about system vulnerabilities. In [175], the authors surveyed 150 papers on FTA; they classified the approaches to determine MCS to either use Boolean manipulation, Binary decision diagrams, or other methods. From a goal perspective, these methods are similar to the actual causality computations; however, the conceptual difference is the definition of a cause. While a cause typically covers two notions: sufficiency and necessity, a CS in a tree presents a sufficient cause only. The occurrence of the events in a CS leads to the occurrence of the top-level event (formally  $\vec{X}$  s.t.  $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}]\varphi$ ). This roughly corresponds to AC1 in the definition, while the minimality of the cut set corresponds to AC3. The difference lies in the necessity of the cause (AC2), i.e., the counterfactual relation in which the non-occurrence of the cause leads to the non-occurrence of the effect. An MCS computation does not include this step, which is the core of actual causality computation. Cut sets are all the enumerations that make the effect true. Some of these enumerations will be an HP cause and others will not depending on necessity. If an HP cause in a fault-tree causal model exists it will be one of the minimal cut sets. Not all the minimal cut sets are HP causes, e.g., cases of overdetermination (conjunctions).

Furthermore, FTA does not reason about an arbitrary combination of events as an effect and does not account for the absence of events as a cause. That said, for tree-based models, FTA and actual causality can complement each other; e.g., the list of MCSs can be used as a set of hypothesized causes for inference, or causal analysis and model checking can be used to construct the fault tree, like in [117].

Additionally, model-based diagnosis (MBD) aims to detect faulty components to explain anomalies between observed and correct system behavior [168, 42]. MBD uses a model that describes a system as a set of logical expressions over a set of components. MBD is similar to actual causality in requiring a set of observations that correspond to the context  $\vec{U}$ ; using logical inference, MBD outputs a set of hypotheses for how the system differs from its model, i.e., diagnoses. While MBD can be considered as an approach to infer causality, it does not embed counter-factuality of the cause in its reasoning. Note that although MBD uses a notion of intervention (setting some components to abnormal, and the rest to normal), this is not counterfactual reasoning. Instead, it is a sufficiency check since MBD uses a behavioral model, i.e., a representation of the correct behavior of the system. As with FTA, diagnoses are a set of sufficient causes, but not all are actual causes according to HP.

The Why-Because-Analysis (WBA) method, through the WBG graph, aims to highlight, potentially missed, factors in classical investigations, and aspire to come up with counter-measures to prevent future accidents [119, 120, 178]. To that end, WBA concludes a list of factors that are *necessary* and *sufficient* to cause the effect. Although very relevant, the result of this process is different from an actual, minimal, counterfactual cause concluded by HP. First, WBA is an accident-investigation methodology via manually checking counter-factual relation (AC2 without considering  $\vec{W}$ ) between each pair of factors in one accident. Thus, the result is a concrete causal model that embeds the context and may not generalize to the

same events in a different accident. Second, WBA accounts for sufficient (AC1 in HP), but not necessary causes, which resembles the preemption situations in HP [178]. This is justified by the fact that WBA aims to enhance safety by putting countermeasures, and hence for such cases, WBA either merges the two confusing causes (if they were similar factors) or creates two different WBGs (in the case of dissimilar factors). Third, each connection in the graph implies a necessary causal factor (NCF) relation, i.e., counter-factually related. In HP causal models, on the other hand, an edge between two factors means a causal relation (not exclusively NCF) exists between the factors. This relation is explained by the equations in the model. Fourth, WBA starts by stating a top-node (mishap) which represents the accident (singleton) event under-investigation. On the other hand, the effect in HP is any single or combination of primitive events.

**Gap 2** *Based on the above, we argue that the literature around actual causality reasoning lacks comprehensive (no restrictions on the model, the cause, or the query), general (no domain-specific artifacts in computation), scalable (performs well with large causal models) algorithms for reasoning about actual causality.*

## 10.3 Causal Models for Accountability

As we stated earlier, causal modeling refers to the act of creating *causal models* (Section 2.2), possibly, from other sources of knowledge. Gregory Cooper published an article on causal discovery using Bayesian networks [38]. Further literature examining model discovery is discussed by Chen and Pearl [31]. These approaches are mainly data-driven methods to discover (type) causal relations. To the best of our knowledge, no previous work has tried to address HP causal modeling in security or safety, specifically, by transforming other models (e.g., attack trees) to causal models. In this section, we discuss sources and domains of causal modeling related to this thesis.

### 10.3.1 Insider Threat and Threat Models

To the best of our knowledge, no previous work has tried to generate HP models for malicious insiders. However, the thorough work on attack and defense modeling is interesting. Kordy et al. [112] surveyed the DAG-based models. Their main classification of the models is either tree or Bayesian network (BN) based. Although a BN is similar to a causal model, there are two differences in utilizing them in security. First, BNs are used for the probabilistic inference of an attack likelihood and prediction. However, we aim to use the causal models for inferring *actual* causality. Second, a causal model contains a semantic perspective represented by the structural equations, while BN only contains a dependency relation supported by the conditional probability table. In this direction, we see the work by Qin et al. [163] which indeed converts attack trees to BN to correlate alerts to predict attacks. Similarly, Althebyan and Panda [7] present a BN model to evaluate and analyze a

system after an insider attack. Both their evaluation and analysis do not include attributing the attacker. Poolsapassit and Ray [161, 164] use AT in a similar way. They do not convert it to other models but rather combine it with the insider’s intent to predict malicious activity. In [161], they use AT to investigate logs. These two papers are related to our goal but differ in the approach of converting AT to causal models annotated with possible suspects. Most of the work reporting on insiders [176, 160] aims to detect the attacks at run-time [109]. Although our work can be combined with such approaches, this is fundamentally different since we consider postmortem attribution. Chinchani et al. [32] proposed a modeling language for insiders. This is interesting, however, we used AT for reasons of industry utilization and tool support [111].

Previous studies have examined *attack graph generation*, primarily relative to computer networks [100, 182, 171, 150], where multiple machines are connected to each other and the Internet. One early study of attack graph generation was conducted by Sheyner et al. using model checkers with a goal property [182]. Model checkers use computational logic to determine if a model is correct; otherwise, if the model is incorrect, the model checkers provide a counterexample. A collection of these counterexamples forms an attack graph. However, model checkers have a computational disadvantage. Amman et al. extended this work with some simplifications and more efficient storage [171]. Ou et al. used a logical attack graph [150] and Ingols et al. [100] used BFS algorithm to tackle the scalability issue. Ingols et al. discussed the redundancy of full and predictive graphs and modeled an attack graph as a multi-purpose graph with contentless edges and three node types. They used BFS technique to generate the attack graph. This approach provides faster results compared to using model checkers. We have extended the work of Ingols [100] and Aksu [5] in conjunction with the Clair OS to generate attack graphs for microservice architectures. Despite their increasing popularity, containers and microservice architectures have demonstrated serious security risks, primarily due to their connectivity requirements and a lesser degree of encapsulation [36, 44]. To the best of our knowledge, no previous study has been conducted targeting attack graph generation for micro-services. More importantly, no previous study tried to generate causal models from attack graphs.

For attack attribution, researchers [181, 93] have identified three techniques: digital forensics, malware based analysis, and indirect attribution techniques that use statistical models to identify attackers. Most of these techniques target outsider attackers. Unlike our approach, digital forensics tools mainly face the challenge of scalability with the size of logs [181], whereas we can elicit the requirements of logging from our nodes. That is, we only monitor the properties that set our context. Malware based analysis targets a different attack vector than us. Indirect attribution techniques are interesting since they use a statistical model. However, they require massive amounts of data. In contrast, we make use of explicit knowledge represented in attack trees.

### 10.3.2 Safety, Fault Trees, and WBA

In the domain of accidents (aircraft, railways) investigation, the WBA tools and methodologies are relevant to our work [119, 120, 178]. The (WBA) Software Toolkit provides functionalities that support an incident investigation, especially in modeling and structuring the occurred factors. Our approach, on the other hand, differentiates the modeling and the context, since it is plausible to use models of re-occurring behavior among incidents. Thus, WBA embeds the context in the model and hence does not support this within its toolkit. Also, since WBA aims at listing all the sufficient causal factors of the accident, the toolkit does not provide an actual causality reasoning capability. Similarly, threat and hazard modeling tools, such as ADT for attack trees [111] and EMFTA for fault trees [43], provide model editors and analysis tools for the user. However, explicit context setting and causality reasoning are not part of the editors. Also, the ability to import other sources of knowledge and transform them into causal models is not supported in all these tools.

A significant body of work is published around xAI; for an overview about post-hoc human explanations see [143, 141]. In our use-case, we did not propose a complete solution like the local explanation in [170]. However, our goal is to emphasize the connection between our approach and xAI. Still, significant work is needed in the domain of modeling for xAI, possibly using our framework.

**Gap 3** *With the various instances of domain-specific sources for causal modeling, the literature lacks practical and (semi-)automated approaches to construct causal models relevant to specific domains.*

## 10.4 Model-driven Contextualization

Most of the papers around forensic readiness tackle the issue of policy organization and did not focus on automated procedures like our approach [189, 197, 49, 172]. The number of logs that have to be analyzed after a security incident motivated Poolsapassit et al. to use Augmented Attack trees and known attack signatures to filter the generated logs for malicious actions [162]. We also use threat models, but to control and automate the logging process rather than filtering the logs produced as part of a forensic investigation process.

Security of microservices has become a popular research area in the past few years on account of the increased interest in security-related issues of the architecture itself and the DevOps practices. In response to the arisen security challenges, researchers studied the runtime monitoring of microservices. Sun et al. [188] proposed in 2015 a security-as-a-service solution for cloud-based microservices. More precisely, in their work, they introduce a cloud-based network security framework to enable the administrators to monitor the network traffic of the communicating microservices. However, their research does not take into account one of the most prominent practices in microservices and DevOps, the containerization, and does not consider the forensic investigation of successful attacks.

Torkura et al. [192] proposed a methodology to integrate continuous security assessment in microservices by introducing a Security Gateway to enforce security policies. Their solution uses the Health Endpoint Monitoring Pattern [87] and acts as a security scanner to identify vulnerabilities in the microservices. Compared to our work, the proposed monitoring system focuses only on the application layer and neither takes into account infrastructure attacks nor deals with computer forensics.

**Gap 4** *The literature lacks domain-specific solutions to tackle the problem of logging granularity and contextualization of causal reasoning in modern systems.*

## 10.5 Summary of the Gaps

We summarize all the gaps identified in this chapter in the following list. Each gap is thoroughly explained in its corresponding section above.

- Gap 1. *Accountability* literature lacks a clear methodology and tools to enable accountability in digital systems.
- Gap 2. The literature around actual causality reasoning lacks comprehensive (no restrictions on the model, the cause, or the query), general (no domain-specific artifacts in computation), scalable (performs well with large causal models) algorithms for reasoning about actual causality.
- Gap 3. With the various instances of domain-specific sources for causal modeling, the literature lacks practical and automated approaches to create causal models.
- Gap 4. In various relevant domains, the literature lacks domain-specific solutions to tackle the problem of logging granularity and contextualization of causal reasoning in modern systems.

# 11 Conclusions

*This chapter concludes the work in the thesis. It summarizes the contributions proposed throughout the chapters. We state the results of the thesis and the lessons learned during the development of this work. Afterward, we discuss limitations and avenues for future work.*

## 11.1 Thesis Overview

This doctoral thesis presents a unifying framework to operationalize the concept of actual causality in modern systems. A concept that we deem crucial for enabling these systems' abilities to explain mishaps, attribute responsibility, and be accountable. In the first part, we discussed the factors that contribute to the inevitability of the system's mishaps, hence the need to design them to be accountable. We presented a general model of the mechanisms that we aspire to add to systems to achieve our goal. The pillar of these mechanisms was an operational ability to reason about the actual causality of events. We discussed the theory of actual causality in Chapter 2, showing its amenability to automated reasoning. However, we also illustrated the practical and computational difficulties that are potentially, preventing its adoption for purposes related to accountability.

The second and core part of the thesis tackled the computational aspects of actual causality. We started, in Chapter 3, with a basic brute-force approach to illustrate the hardness of the problem of causality checking, i.e., deciding if a candidate cause is an actual cause of some observed effect. Then, limiting ourselves to binary models, we presented a sound method to encode both counterfactuality, and minimality conditions, given a candidate cause, as propositional formulae. This, in turn, allows us to use off-the-shelf SAT solvers to answer actual causality queries. Further, we presented multiple extensions to the approach that dealt with additional requirements such as degree of responsibility or aimed to optimize the performance. Even with tiny models (30 variables), checking causality in a brute force manner is incomputable, whereas the SAT-based approach scaled to large models (4000 variables) within seconds. This result seems relevant when causal models are generated from other artifacts, as done several times in this thesis.

To support other notions of reasoning, we showed, in Chapter 4, how the problem of checking, semi-inference, and inference could effectively and efficiently be solved as an optimization problem. Keeping our restriction to binary models, we presented quantifiable notions of causal inference within counterfactual computations and showed how to

encode them within an optimization problem. We contributed a MaxSAT formulation that outperforms all other approaches in answering causality checking queries and an ILP formulation that eliminates the need for a candidate cause in the query. Using models with 8000 variables, which we deem realistic and necessary for automatically inferred causal models, we show that our approaches answer checking queries in seconds, and inference queries in minutes.

We concluded the part on computational aspects of actual causality by eliminating the restriction to binary models. In Chapter 5, we presented a novel reasoning method given numerical causal models. The method presented a generalization of the concepts used in the previous techniques and showed how to use them in the context of numerical variables and equations. We discussed the impact of this generalization on the efficiency of answering actual causality queries.

After establishing a sound foundation for causality reasoning in the second part of the thesis, in its third, we focused on practical methods of causal modeling and contextualization in one domain of applications, i.e., microservice-based information systems. Although both, modeling and contextualization, are generic requirements for our solution, addressing them cannot be done in general; thus, we showed examples of how can domain-specific modeling and contextualization work in practice. We started, in Chapter 6, by capturing the interaction between domain-specific artifacts such as attack trees and causal models. For purposes of accountability, especially in the context of insiders, we showed that such models are beneficial, and we considered attack trees as one source to construct them. We proceeded then to explore methods to generate these artifacts automatically in Chapter 7. Specifically, we presented a mechanism to generate attack graphs on a continuous base, which can then be transformed into causal models. The semi-automated construction of causal models from attack trees and the automated construction from attack graphs illustrate practical causal modeling methodologies. We concluded this part of the thesis by presenting an architecture, in Chapter 8, that uses the models to guide the contextualization of insiders' actions and determines the logging granularity, as one instance of the general problem of contextualization.

In the fourth part of the thesis, we presented a general framework that consists of reasoning, modeling, and contextualization abilities. We showed how these abilities are effective in enabling accountability or explainability of a system. We discussed different possible patterns of modeling, contextualization, and reasoning that can be orchestrated within the framework. We illustrated these patterns using case studies in domains of explainable AI, aircraft, drones, and information systems forensics.

## 11.2 Main Results

**Despite the computational complexity of the theory, actual causality for binary models, according to HP, is efficiently and effectively checked with SAT solving.** Our encodings verify whether a candidate cause is an actual cause of an event (or a combination of



events), for a large class of models, within seconds. We used a dataset that we deem representative of our target systems. However, the performance is impacted when tackling the minimality. Both minimalities of the contingency set ( $\vec{W}$ ) for responsibility computation or the minimality condition of the cause requires a longer time to calculate.

**Actual causality is efficiently inferred when formulated as an optimization problem.** Keeping the limitation to binary models, our formulation of checking queries as optimization problems, answers such queries almost instantly. Also, it introduces the ability to semi-infer actual causality. Especially using MaxSAT, the efficiency of obtaining answers is remarkable. Furthermore, the inference of actual causality is efficiently computed as a multi-objective integer linear program. We show to a large scale that queries are answered in a timely fashion given sufficient memory resources.

**The efficient checking of actual causality as an optimization problem is generalizable to non-binary models.** The same concepts of quantifiable counterfactual computations of binary models are also applicable to numerical models—however, the solution does not scale to the size that the binary approaches reach.

**Causal modeling and contextualization can be operationalized utilizing domain-specific artifacts.** The domain of micro-service-based systems, for instance, requires enabling accountability in the context of insiders. Attack trees, depicting potential attacks by insiders, are useful templates to extract causal models semi-automatically. The extraction considers domain-specific attributes such as collusion attacks and preemption scenarios. Similarly, fault trees, depicting typical failures of safety-critical systems, are another source for creating causal models. Further, such templates and causal models, in turn, can be generated automatically leveraging the network topology. This domain-specific operationalization of modeling enhances contextualization as well. In this domain, contextualization is effectively and efficiently achieved through model-driven methodologies that advocate the right level of logging.

**Different, yet related, patterns of operationalizing actual causality are observed within modern systems.** Our unifying framework showed multiple ways to orchestrate modeling, contextualization, and reasoning. For instance, similar to the case of insiders in information systems, fault trees of UAVs also operationalize modeling and guide contextualization for causality and accountability. In contrast, the context collected from logs and reports of aircraft accidents drives the construction of why-because-graphs. More straightforward than the two cases, the model and the context are both available in the domain of explainable AI. The reasoning for all the aforementioned is effective because it caters to the specific requirements (e.g., responsible insiders, contrastive queries, or set of sufficient causes) of each field.

### 11.3 Limitations

- Causal Models Types: The types of the variables and their relations in a causal model impact the efficiency of the computation algorithms.
  - In this thesis, we limited our focus to acyclic models. This limitation follows the literature and most of Halpern’s examples. Although all the models that we found are indeed acyclic, the theory of actual causality supports cyclic models. All the algorithms in this thesis can, in principle, be adapted to support cyclic models according to the treatment proposed by Halpern (see Chapter 6 [75]).
  - At the beginning of this thesis, we restricted our focus to binary models. A restriction that is deemed reasonable in the literature due to the scarcity of non-binary models. However, we also illustrated an extension to numerical models, which we expect will be of relevance to explainable AI purposes in the future. In this thesis, we only consider binary models with propositional formulae or numerical models with linear equations.
  - All the queries in this thesis assume a deterministic model and context. Although the need for probabilistic models may arise in practice, our tools do not support probabilities out of the box. Since HP treats probabilistic models by pulling probability out of the model, we do not consider it a severe limitation.
  - Our utilization of the notion of responsibility is based on the definition given by Chockler and Halpern in [33]. While there are other notions in the literature, we use it due to its compatibility with the deterministic models we use.
- Model Incompleteness: HP is a model relative theory, which is further complicated by the fact that causal models are necessarily incomplete. In this thesis, we do not assess the quality of the causal model itself; instead, we transform existing knowledge captured in other models and argue about the validity of the transformation. Further, the tools contributed within this thesis support various ways to allow the change in the models, for instance, the continuous modeling for microservices, or the interactive modeling platform within *Canvas*.
- External Validity: The results of the empirical evaluation are bound to our dataset. While we remain positive this dataset is realistic and representative of complex cases, the results may not generalize to other models with different structures. Also, with the continuous development of SAT, MaxSAT, ILP solvers, the reported results may differ with newer versions or recent new solvers. Most probably, the differences will be an enhancement of our reported results.

### 11.4 Future Work

In this section, we mention several aspects that we consider important for future work.

The substantial influence of the *model* on the inferred causality requires quality control on the modeling part. A potential future research direction is to explore the qualities of a causal model for accountability (*what constitutes a valid model for an accountable system?*). Automated construction of such models for accountability is another useful follow-up of this thesis. Possibly, this automation may use the reasoning machinery provided in this thesis to construct valuable causal models.

Also, the impact of the model's *structure* on the efficiency of the computation is another direction for further research. A thorough characterization of causal model classes that affect the efficiency of the proposed approach is a useful follow-up. This might be helpful in understanding and possibly predicting the required time of computing the satisfiability of different formulas generated by our algorithms.

The core part of this thesis focused on *reasoning*. We think this part can be extended in several ways. One way is to develop all the algorithms to treat non-deterministic causal models and contexts. While reasoning under uncertainty is tackled in the HP definitions, and hence, supported by the algorithms, still, potential computational problems may arise.

A sophisticated selection of the *counterfactual world* is another direction for future work. The solid foundation of our computations is a description of the counterfactual world; the more quantifiable aspects (e.g., minimality, degree of responsibility) we incorporated in the description, the better abilities (e.g., inference) of reasoning we got. Applying additional restrictions (e.g., normality) may result in picking "better" counterfactual worlds, and hence, better answers to causal queries.

A recent topic of interest in the research community is *explainable AI*. Extending our framework with a formal account of *explanation* in AI grounded on the definitions of actual causality is a potential contribution. Further, we consider the construction of structural equation models based on machine-learned models, an interesting research area. We see our established framework as an enabler of such research, especially with the support of numerical models.



# Bibliography

- [1] Microservice architecture. <https://microservices.io/articles/whoisusingmicroservices.html>, 2018. Retrieved September 4 2018.
- [2] Phpmailer 5.2.18 remote code execution. <https://github.com/opsxcq/exploit-CVE-2016-10033>, 2018. Retrieved September 4 2018.
- [3] Sambacry rce exploit for samba 4.5.9. <https://github.com/opsxcq/exploit-CVE-2017-7494>, 2018. Retrieved September 4 2018.
- [4] Mohsen Ahmadvand and Amjad Ibrahim. Requirements reconciliation for scalable and secure microservice (de) composition. In *Requirements Engineering Conference Workshops (REW), IEEE International*, pages 68–73. IEEE, 2016.
- [5] M Ugur Aksu, Kemal Bicakci, M Hadi Dilek, A Murat Ozbayoglu, et al. Automated generation of attack graphs using nvd. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 135–142. ACM, 2018.
- [6] Gadi Aleksandrowicz, Hana Chockler, Joseph Y. Halpern, and Alexander Ivrii. The computational complexity of structure-based causality. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.
- [7] Qutaibah Althebyan and Brajendra Panda. A knowledge-based bayesian model for analyzing a system after an insider attack. In *Proceedings of The Ifip Tc 11 23 rd International Information Security Conference*, pages 557–571. Springer, 2008.
- [8] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM, 2002.
- [9] Zaruhi Aslanyan and Flemming Nielson. Model checking exact cost for attack scenarios. In *International Conference on Principles of Security and Trust*. Springer, 2017.
- [10] Fahiem Bacchus, Matti Järvisalo, Ruben Martins, et al. Maxsat evaluation 2018, 2018.
- [11] Fahiem Bacchus and Nina Narodytska. Cores in core based maxsat algorithms: An analysis. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 7–15. Springer, 2014.

- [12] Vitaliy Batusov and Mikhail Soutchanski. Situation calculus semantics for actual causality. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [13] Kristian Beckers, Jörg Landthaler, Florian Matthes, Alexander Pretschner, and Bernhard Walzl. Data accountability in socio-technical systems. In *International Workshop on Business Process Modeling, Development and Support*, pages 335–348. Springer, 2016.
- [14] Adrian Beer, Stephan Heidinger, Uwe Kühne, Florian Leitner-Fischer, and Stefan Leue. Symbolic causality checking using bounded model checking. In *Model Checking Software - 22nd International Symposium, SPIN*, 2015.
- [15] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard Trefler. *Explaining Counterexamples Using Causality*, pages 94–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
- [16] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard J. Trefler. Explaining counterexamples using causality. *Formal Methods in System Design*, 40(1):20–40, 2012.
- [17] Pietro Belotti, Leo Liberti, Andrea Lodi, Giacomo Nannicini, and Andrea Tramontani. Disjunctive inequalities: applications and extensions. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.
- [18] Leopoldo Bertossi. Characterizing and computing causes for query answers in databases from database repairs and repair programs. In *International Symposium on Foundations of Information and Knowledge Systems*, pages 55–76. Springer, 2018.
- [19] Johannes Bisschop. *AIMMS optimization modeling*. Lulu. com, 2006.
- [20] Stefano Bistarelli, Fabio Fioravanti, and Pamela Peretti. Defense trees for economic evaluation of security investments. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8–pp. IEEE, 2006.
- [21] Andreas Blass and Yuri Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1-3):80–88, 1982.
- [22] Harold Booth, Doug Rike, and Gregory A Witte. The national vulnerability database (nvd): Overview, 2013.
- [23] James Bottomley. *What is All the Container Hype?*, 2014.
- [24] Mark Bovens. Two concepts of accountability: Accountability as a virtue and as a mechanism. *West European Politics*, 33(5):946–967, 2010.
- [25] Gerald G Brown and Robert F Dell. Formulating integer linear programs: A rogues’ gallery. *INFORMS Transactions on Education*, 7(2):153–159, 2007.

- [26] Glenn Bruns and Stuart Anderson. Validating safety models with fault trees. In *SAFECOMP'93*, pages 21–30. Springer, 1993.
- [27] Thanh Bui. Analysis of docker security. *arXiv preprint arXiv:1501.02967*, 2015.
- [28] Björn Butzin, Frank Golasowski, and Dirk Timmermann. Microservices approach for the internet of things. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–6. IEEE, 2016.
- [29] Tomas Cerny, Michael J Donahoo, and Michal Trnka. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review*, 17(4):29–45, 2018.
- [30] A. S. Cheliyan and S. K. Bhattacharyya. Fuzzy fault tree analysis of oil and gas leakage in subsea production systems. *Journal of Ocean Engineering and Science*, 2018.
- [31] Bryant Chen and Judea Pearl. Graphical tools for linear structural equation modeling. Technical report, DTIC Document, 2014.
- [32] Ramkumar Chinchani, Anusha Iyer, Hung Q Ngo, and Shambhu Upadhyaya. Towards a theory of insider threat assessment. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 108–117. IEEE, 2005.
- [33] Hana Chockler and Joseph Y Halpern. Responsibility and blame: A structural-model approach. *Journal of Artificial Intelligence Research*, pages 93–115, 2004.
- [34] Hana Chockler, Joseph Y Halpern, and Orna Kupferman. What causes a system to satisfy a specification? *ACM Transactions on Computational Logic (TOCL)*, 9(3):20, 2008.
- [35] John David Collins, Edward J. Hall, and Laurie A. Paul, editors. *Causation and counterfactuals*. Representation and mind. MIT Press, Cambridge, Mass, 2004. Elektronische Ressource.
- [36] Theo Combe, Antony Martin, and Roberto Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, 2016.
- [37] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.
- [38] Gregory Cooper. An overview of the representation and discovery of causal relationships using bayesian networks. *Computation, causation, and discovery*, pages 4–62, 1999.
- [39] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

- [40] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, volume 185, pages 403–424. IOS Press, 2009.
- [41] A. P. Dawid. Causal inference without counterfactuals. *Journal of the American Statistical Association*, 95(450):407–424, jun 2000.
- [42] Johan De Kleer and James Kurien. Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5):25–36, 2003.
- [43] Julien Delange. Emfta: an open source tool for fault tree analysis.
- [44] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.
- [45] Christof Ebert and Capers Jones. Embedded software: Facts, figures, and future. *Computer*, 42(4):42–52, 2009.
- [46] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing*, pages 61–75, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [47] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference.*, pages 502–518, 2003.
- [48] Thomas Eiter and Thomas Lukasiewicz. Complexity results for structure-based causality. *Artificial Intelligence*, 142(1):53–89, 2002.
- [49] Barbara Endicott-Popovsky, Deborah A Frincke, and Carol A Taylor. A theoretical framework for organizational network forensic readiness. *JCP*, 2(3):1–11, 2007.
- [50] Christian Esposito, Aniello Castiglione, and Kim-Kwang Raymond Choo. Challenges in delivering software in the cloud as microservices. *IEEE Cloud Computing*, 3(5):10–14, 2016.
- [51] Daniel Farmer and Eugene H Spafford. The cops security checker system, 1990.
- [52] Joan Feigenbaum, James A. Hendler, Aaron D. Jaggard, Daniel J. Weitzner, and Rebecca N. Wright. Accountability and deterrence in online life. In *Web Science 2011, WebSci '11, Koblenz, Germany - June 15 - 17, 2011*, pages 7:1–7:7, 2011.
- [53] Joan Feigenbaum, Aaron D. Jaggard, and Rebecca N. Wright. Towards a formal model of accountability. In *2011 New Security Paradigms Workshop, NSPW '11*, pages 45–56, 2011.



- [54] Christof Fetzer. Building critical applications using microservices. *IEEE Security & Privacy*, (6):86–89, 2016.
- [55] International Organization for Standardization and International Electrotechnical Commission. *Software Engineering-Product Quality*, volume 9126. ISO/IEC, 2001.
- [56] International Organization for Standardization and International Electrotechnical Commission. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*, volume 25010. ISO/IEC, 2011.
- [57] ArduPilot Discuss Forums. Ardupilot discourse.
- [58] Martin Fowler. *Microservices resource guide*, 2015.
- [59] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 24–33. ACM, 2014.
- [60] Olga Gadyatskaya, Ravi Jhavar, Piotr Kordy, Karim Lounis, Sjouke Mauw, and Rolando Trujillo-Rasua. Attack trees for practical security assessment: Ranking of attack scenarios with adtool 2.0. In *Quantitative Evaluation of Systems - 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings*, pages 159–162, 2016.
- [61] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [62] Tobias Gerstenberg. *Making a difference: Responsibility, causality and counterfactuals*. PhD thesis, University College London (University of London), 2013.
- [63] Mojdeh Golagha. *A Framework for Failure Diagnosis*. PhD thesis, Technische Universität München, 2020.
- [64] Gregor Gössler and Lăcrămioara Aștefănoaei. Blaming in component-based real-time systems. In *Proceedings of the 14th International Conference on Embedded Software - EMSOFT 14*. ACM Press, 2014.
- [65] Gregor Gößler and Lacramioara Astefanoaei. Blaming in component-based real-time systems. In *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*, pages 7:1–7:10, 2014.
- [66] Gregor Gössler and Daniel Le Métayer. A general trace-based framework of logical causality. In *International Workshop on Formal Aspects of Component Software*, pages 157–173. Springer, 2013.

- [67] Gregor Gößler and Daniel Le Métayer. A general trace-based framework of logical causality. In *Formal Aspects of Component Software - 10th International Symposium, FACS 2013, Nanchang, China, October 27-29, 2013, Revised Selected Papers*, pages 157–173, 2013.
- [68] Jim Gray. A conversation with werner vogels. *ACM Queue*, 4(4):14–22, 2006.
- [69] Orna Grumberg, Assaf Schuster, and Avi Yadgar. Memory efficient all-solutions SAT solver and its application for reachability analysis. In *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, pages 275–289, 2004.
- [70] Jayanth Gummaraju, Tarun Desikan, and Yoshio Turner. Over 30% of official images in docker hub contain high priority security vulnerabilities. In *Technical Report*. BanyanOps, 2015.
- [71] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.
- [72] N. Hall. Structural equations and causation. *Philosophical Studies*, 132(1):109–136, 2007.
- [73] Joseph Y Halpern. Defaults and normality in causal structures. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 198–208. AAAI Press, 2008.
- [74] Joseph Y. Halpern. A modification of the Halpern-Pearl definition of causality. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 3022–3033, 2015.
- [75] Joseph Y. Halpern. *Actual causality*. The MIT Press, Cambridge, Massachusetts, 2016.
- [76] Joseph Y. Halpern and Christopher Hitchcock. Actual causation and the art of modeling. In *Causality, Probability, and Heuristics: A Tribute to Judea Pearl*, pages 383–406. London: College Publications, 2010.
- [77] Joseph Y Halpern and Christopher Hitchcock. Graded causation and defaults. *The British Journal for the Philosophy of Science*, 66(2):413–457, 2014.
- [78] Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural model approach – part i: Causes. 2001.
- [79] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach - part I: causes. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001*, pages 194–202, 2001.

- [80] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach - part II: explanations. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 27–34, 2001.
- [81] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part i: Causes. *The British Journal for the Philosophy of Science*, 56(4):843–887, 2005.
- [82] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach, part i: Causes. 2005.
- [83] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part ii: Explanations. *The British Journal for the Philosophy of Science*, 56(4):889–911, 2005.
- [84] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.
- [85] Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for sat and qsat. *Journal of Artificial Intelligence Research*, 53:127–168, 2015.
- [86] Christopher Hitchcock. The intransitivity of causation revealed in equations and graphs. *The Journal of Philosophy*, 98(6):273–299, 2001.
- [87] Alex Homer, John Sharp, Larry Brader, Masashi Narumoto, and Trent Swanson. *Cloud design patterns: Prescriptive architecture guidance for cloud applications*. Microsoft patterns & practices, 2014.
- [88] Mark Hopkins. *The actual cause: From intuition to automation*. PhD thesis, University of California, Los Angeles.
- [89] Mark Hopkins. Strategies for determining causes of events. In *AAAI/IAAI*, 2002.
- [90] Mark Hopkins and Judea Pearl. Clarifying the usage of structural models for commonsense causal reasoning. 2003.
- [91] Ross Horne, Sjouke Mauw, and Alwen Tiu. Semantics for specialising attack trees based on linear logic. *Fundamenta Informaticae*, 153(1-2):57–86, 2017.
- [92] David Hume. An enquiry concerning human understanding. *History of Economic Thought Books*, 1748.
- [93] Jeffrey Hunker, Bob Hutchinson, and Jonathan Margulies. Role and challenges for sufficient cyber-attack attribution. *Institute for Information Infrastructure Protection*, 2008.

- [94] Amjad Ibrahim, Stevica Bozhinoski, and Alexander Pretschner. Attack graph generation for microservice architecture. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1235–1242. ACM, 2019.
- [95] Amjad Ibrahim, Severin Kacianka, Alexander Pretschner, Charles Hartsell, and Gabor Karsai. Practical causal models for cyber-physical systems. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods*, pages 211–227, Cham, 2019. Springer International Publishing.
- [96] Amjad Ibrahim, Tobias Klesel, Ehsan Zibaei, Severin Kacianka, and Alexander Pretschner. Actual causality canvas: A general framework for explanation-based socio-technical constructs. In *ECAI 2020, the 24th European Conference on Artificial Intelligence*, *Frontiers in Artificial Intelligence and Applications*, pages 2978 – 2985. IOS Press, 2020.
- [97] Amjad Ibrahim and Alexander Pretschner. From checking to inference: Actual causality computations as optimization problems. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis*, pages 343–359, Cham, 2020. Springer International Publishing.
- [98] Amjad Ibrahim, Simon Rehwald, and Alexander Pretschner. Efficient checking of actual causality with sat solving. *Engineering Secure and Dependable Software Systems*, 53:241, 2019.
- [99] Amjad Ibrahim, Simon Rehwald, Antoine Scemama, Florian Andres, and Alexander Pretschner. Causal model extraction from attack trees to attribute malicious insider attacks. In *Graphical Models for Security*, pages 3–23, Cham, 2020. Springer International Publishing.
- [100] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 121–130. IEEE, 2006.
- [101] Ponemon Institute. 2015 cost of cyber crime study: Global.
- [102] David Jaramillo, Duy V Nguyen, and Robert Smart. Leveraging microservices architecture by using docker technology. In *SoutheastCon, 2016*, pages 1–5. IEEE, 2016.
- [103] Severin Kacianka, Kristian Beckers, Florian Kelbert, and Prachi Kumari. How accountability is implemented and understood in research tools. In *International Conference on Product-Focused Software Process Improvement*, pages 199–218. Springer, 2017.
- [104] Severin Kacianka, Amjad Ibrahim, Alexander Pretschner, Alexander Trende, and Andreas Lüdtke. Extending causal models from machines into humans. In *4th Workshop on Formal Reasoning about Causation, Responsibility, & Explanations in Science & Technology*, 2019.

- [105] Severin Kacianka, Florian Kelbert, and Alexander Pretschner. Towards a unified model of accountability infrastructures. 2016.
- [106] Severin Kacianka, Florian Kelbert, and Alexander Pretschner. Towards a unified model of accountability infrastructures. In *Proceedings of CREST@ETAPS 2016*, pages 40–54, 2016.
- [107] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- [108] Samantha Kleinberg and George Hripcsak. A review of causal inference for biomedical informatics. *Journal of Biomedical Informatics*, 44(6):1102–1112, 2011.
- [109] Li Ling Ko, Dinil Mon Divakaran, Yung Siang Liau, and Vrizlynn LL Thing. Insider threat detection and its future directions. *International Journal of Security and Networks*, 12(3), 2017.
- [110] Thorsten Koch, Alexander Martin, and Marc E Pfetsch. Progress in academic computational integer programming. In *Facets of Combinatorial Optimization*. Springer, 2013.
- [111] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. Adtool: security analysis with attack–defense trees. In *International conference on quantitative evaluation of systems*, pages 173–176. Springer, 2013.
- [112] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. Dag-based attack and defense modeling: Don’t miss the forest for the attack trees. *Computer science review*, 13:1–38, 2014.
- [113] Nane Kratzke. About microservices, containers and their underestimated impact on network performance. *arXiv preprint arXiv:1710.04049*, 2017.
- [114] Alexandr Krylovskiy, Marco Jahn, and Edoardo Patti. Designing a smart city internet of things platform with microservice architecture. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 25–30. IEEE, 2015.
- [115] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. Quantitative attack tree analysis via priced timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 156–171. Springer, 2015.
- [116] Robert Künnemann, Ilkan Esiyok, and Michael Backes. Automated verification of accountability in security protocols. *CoRR*, abs/1805.10891, 2018.
- [117] Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. From probabilistic counterexamples via causality to fault trees. In *Computer Safety, Reliability, and Security - 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings*, pages 71–84, 2011.

- [118] Stavros Kyriakopoulos. Model-driven monitoring orchestration for microservices, 2019.
- [119] Peter Ladkin and Karsten Loer. Why-because analysis: Formal reasoning about incidents. *Bielefeld, Germany, Document RVS-Bk-98-01, Technischen Fakultät der Universität Bielefeld, Germany*, 1998.
- [120] Peter B Ladkin. Causal reasoning about aircraft accidents. In *International Conference on Computer Safety, Reliability, and Security*, pages 344–360. Springer, 2000.
- [121] Emily LeBlanc, Marcello Balduccini, and Joost Vennekens. Explaining actual causation via reasoning about actions and change. In *European Conference on Logics in Artificial Intelligence*, pages 231–246. Springer, 2019.
- [122] Florian Leitner-Fischer. *Causality Checking of Safety-Critical Software and Systems*. PhD thesis, University of Konstanz, Germany, 2015.
- [123] Florian Leitner-Fischer and Stefan Leue. *Causality Checking for Complex System Models*, pages 248–267. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [124] Florian Leitner-Fischer and Stefan Leue. Probabilistic fault tree synthesis using causality computation. *IJCCBS*, 4(2):119–143, 2013.
- [125] Florian Leitner-Fischer and Stefan Leue. Spincause: a tool for causality checking. In *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 117–120, 2014.
- [126] David Lewis. Causation. *Journal of Philosophy*, 70(17):556–567, 1973.
- [127] David Lewis. Counterfactuals and comparative possibility. *Journal of Philosophical Logic*, 2(4):418–446, 1973.
- [128] Chu Min Li and Felip Manyà. Maxsat, hard and soft constraints. *Handbook of satisfiability*, 185:613–631, 2009.
- [129] Ruiming Li, Dian Zhou, and Donglei Du. Satisfiability and integer programming as complementary tools. In *Proceedings of the 2004 Asia and South Pacific design automation conference*, pages 879–882. IEEE Press, 2004.
- [130] Algirdas Antano Maknickas. Linear programming formulation of boolean satisfiability problem. In *Transactions on Engineering Technologies*, pages 305–321. Springer, 2014.
- [131] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 438–445. Springer, 2014.

- [132] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*, pages 186–198, 2005.
- [133] Genc Mazlami, Jürgen Cito, and Philipp Leitner. Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531. IEEE, 2017.
- [134] Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y. Halpern, Christoph Koch, Katherine F. Moore, and Dan Suciu. Causality in databases. *IEEE Data Eng. Bull.*, 33(3):59–67, 2010.
- [135] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. Why so? or why no? functional causality for explaining query answers. *CoRR*, abs/0912.5340, 2009.
- [136] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.
- [137] Alexandra Meliou, Wolfgang Gatterbauer, Suman Nath, and Dan Suciu. Tracing data errors with view-conditioned causality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 505–516, 2011.
- [138] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. Bringing provenance to its full potential using causal reasoning. In *3rd Workshop on the Theory and Practice of Provenance, TaPP’11, Heraklion, Crete, Greece, June 20-21, 2011*, 2011.
- [139] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.
- [140] U. S. Mian, J. I. den Hartog, S. Etalle, and N. Zannone. Auditing with incomplete logs. In *Proceedings of the 3rd Workshop on Hot Issues in Security Principles and Trust 2015, London, UK*, pages 1–23, Eindhoven, April 2015. Technische Universiteit Eindhoven.
- [141] Tim Miller. *Explanation in artificial intelligence: Insights from the social sciences*. *Artificial Intelligence*, 2018.
- [142] Hans D Mittelmann and P Spellucci. *Decision tree for optimization software*, 2005.
- [143] Brent Mittelstadt, Chris Russell, and Sandra Wachter. Explaining explanations in ai. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 279–288. ACM, 2019.
- [144] Christoph Molnar. *Interpretable Machine Learning*. 2019. <https://christophm.github.io/interpretable-ml-book/>.

- [145] Richard Mulgan. 'accountability': An ever-expanding concept? *Public Administration*, 78(3):555–573, 2000.
- [146] Sam Newman. *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.
- [147] Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, and Laurent Simon. Impact of community structure on sat solver performance. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 252–268. Springer, 2014.
- [148] Andrew Nicholson, Helge Janicke, and Tim Watson. An initial investigation into attribution in scada systems. In *ICS-CSR*, 2013.
- [149] German Federal Bureau of Aircraft Accident Investigation. Investigation report ax001-1-2/02, 2004.
- [150] Xinming Ou, Wayne F Boyer, and Miles A McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345. ACM, 2006.
- [151] Claus Pahl and Pooyan Jamshidi. Microservices: A systematic mapping study. In *CLOSER (1)*, pages 137–146, 2016.
- [152] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of facets (and some facets of complexity). *J. Comput. Syst. Sci.*, 28(2):244–259, 1984.
- [153] Nick Papanikolaou and Siani Pearson. A cross-disciplinary review of the concept of accountability a survey of the literature. 2013.
- [154] Mike P Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12. IEEE, 2003.
- [155] Judea Pearl. Causation, action and counterfactuals. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge, De Zeeuwse Stromen, The Netherlands, March 17-20 1996*, pages 51–73, 1996.
- [156] Judea Pearl. On the definition of actual cause, 1998.
- [157] Judea Pearl. *Causality: models, reasoning and inference*, volume 29. Springer, 2000.
- [158] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic Books, 2018.



- [159] Siani Pearson and Andrew Charlesworth. Accountability as a way forward for privacy protection in the cloud. In Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong, editors, *Cloud Computing*, pages 131–144, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [160] AH Phyto and SM Furnell. A detection-oriented classification of insider it misuse. In *Third Security Conference*, 2004.
- [161] Nayot Poolsapassit and Indrajit Ray. Investigating computer attacks using attack trees. *Advances in digital forensics III*, pages 331–343, 2007.
- [162] Nayot Poolsapassit and Indrajit Ray. Investigating computer attacks using attack trees. In *IFIP International Conference on Digital Forensics*, pages 331–343. Springer, 2007.
- [163] Xinzhou Qin and Wenke Lee. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 370–379. IEEE, 2004.
- [164] Indrajit Ray and Nayot Poolsapassit. Using attack trees to identify malicious attacks from authorized insiders. In *ESORICS*, volume 3679, pages 231–246. Springer, 2005.
- [165] Martin Reháč, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Thomas Engel. Runtime monitoring and dynamic re-configuration for intrusion detection systems. In *International Workshop on Recent Advances in Intrusion Detection*, pages 61–80. Springer, 2009.
- [166] Simon Rehwald. A technical framework for actual causality inference, 2018.
- [167] Simon Rehwald, Amjad Ibrahim, Kristian Beckers, and Alexander Pretschner. Ac-benchmark: A framework for comparing causality algorithms. In *CREST@ETAPS 2017, Uppsala, Sweden, 29th April 2017.*, pages 16–30, 2017.
- [168] Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.
- [169] IBM® X-Force® Research. 2016 cyber security intelligence index.
- [170] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.
- [171] Ronald W Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 156–165. IEEE, 2000.

- [172] Robert Rowlingson. A ten step process for forensic readiness. *International Journal of Digital Evidence*, 2(3):1–28, 2004.
- [173] Arpan Roy, Dong Seong Kim, and Kishor S Trivedi. Act: Attack countermeasure trees for information assurance analysis. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–2. IEEE, 2010.
- [174] Arpan Roy, Dong Seong Kim, and Kishor S Trivedi. Cyber security analysis using attack countermeasure trees. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, page 28. ACM, 2010.
- [175] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15:29–62, 2015.
- [176] Malek Ben Salem, Shlomo Hershkop, and Salvatore J Stolfo. A survey of insider attack detection research. In *Insider Attack and Cyber Security*, pages 69–90. Springer, 2008.
- [177] Babak Salimi and Leopoldo Bertossi. From causes for database queries to repairs and model-based diagnosis and back. 2014.
- [178] Dipl-Inform Jan Sanders. Introduction to why-because analysis. *Dipl.-Inform*, February, 2012.
- [179] Bruce Schneier. Attack Trees - Modeling security threats. *Dr. Dobb's Journal*, December 1999.
- [180] Uwe Schöning and Jacobo Torán. *The satisfiability problem : algorithms and analyses*. Mathematik für Anwendungen ; 3. Lehmanns Media, Berlin, 2013. English translation of a slightly extended version; Erscheint: Juli 2013.
- [181] Jawwad A Shamsi, Sherali Zeadally, Fareha Sheikh, and Angelyn Flowers. Attribution in cyberspace: techniques and legal implications. *Security and Communication Networks*, 9(15), 2016.
- [182] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Proceedings-IEEE Symposium on Security and Privacy*, page 273. IEEE, 2002.
- [183] Rui Shu, Xiaohui Gu, and William Enck. A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 269–280. ACM, 2017.
- [184] Jörn Stuphor. Amazonas midair collision - wba of public information.
- [185] Jörn Stuphor. Handout of the 2002 ueberlingen mid-air.

- [186] Jörn Stuphor. Kausale untersuchung der kollision zweier verkehrsflugzeuge über dem bodensee, 1. juli 2002.
- [187] Jörn Stuphor. The wbg of 2002 ueberlingen mid-air.
- [188] Yuqiong Sun, Susanta Nanda, and Trent Jaeger. Security-as-a-service for microservices-based cloud applications. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 50–57. IEEE, 2015.
- [189] John Tan. Forensic readiness. *Cambridge, MA:@ Stake*, pages 1–23, 2001.
- [190] Takahisa Toda and Takehide Soh. Implementing efficient all solutions sat solvers. *Journal of Experimental Algorithmics (JEA)*, 21:1–12, 2016.
- [191] Kennedy A Torkura, Muhammad IH Sukmana, and Anne VDM Kayem. A cyber risk based moving target defense mechanism for microservice architectures. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 932–939. IEEE, 2018.
- [192] Kennedy A Torkura, Muhammad IH Sukmana, and Christoph Meinel. Integrating continuous security assessments in microservices and cloud native applications. In *Proceedings of the 10th International Conference on Utility and Cloud Computing*, pages 171–180. ACM, 2017.
- [193] Manghui Tu, Dianxiang Xu, Eugene Butler, and Amanda Schwartz. Forensic evidence identification and modeling for attacks against a simulated online business information system. *Journal of Digital Forensics, Security and Law*, 7(4):4, 2012.
- [194] Tony J Van Roy and Laurence A Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1):45–57, 1987.
- [195] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. *Fault tree handbook*, 1981.
- [196] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57, 2015.
- [197] Sebastiaan Von Solms, Cecil Louwrens, Colette Reekie, and Talania Grobler. A control framework for digital forensics. In *Advances in Digital Forensics II*, pages 343–355. Springer, 2006.
- [198] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.

- [199] Shaohui Wang, Anaheed Ayoub, BaekGyu Kim, Gregor Gössler, Oleg Sokolsky, and Insup Lee. *A Causality Analysis Framework for Component-Based Real-Time Systems*, pages 285–303. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [200] Daniel J Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information accountability. *Communications of the ACM*, 51(6):82–87, 2008.
- [201] David A Wheeler and Gregory N Larsen. Techniques for cyber attack attribution. Technical report, INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA, 2003.
- [202] Eberhard Wolff. *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016.
- [203] Richard W Wright. Causation, responsibility, risk, probability, naked statistics, and proof: Pruning the bramble bush by clarifying the concepts. *Iowa Law Review*, 73:1001, 1988.
- [204] Weinan Zhao and Weimin Wu. ASIG: an all-solution SAT solver for CNF formulas. In *11th International Conference on Computer-Aided Design and Computer Graphics, CAD/Graphics 2009, Huangshan, China, August 19-21, 2009*, pages 508–513, 2009.
- [205] Ehsan Zibaei, Sebastian Banescu, and Alexander Pretschner. Diagnosis of safety incidents for cyber-physical systems: A uav example. In *2018 3rd International Conference on System Reliability and Safety (ICSRS)*, pages 120–129. IEEE, 2018.
- [206] Ehsan Zibaei and Alexander Pretschner. Automated diagnosis of drone crashes. <https://www22.in.tum.de/en/teaching/automated-diagnosis-drone/>. Accessed: 2019-11-25.
- [207] Saman A Zonouz, Himanshu Khurana, William H Sanders, and Timothy M Yardley. Rre: A game-theoretic intrusion response and recovery engine. In *IEEE/IFIP International Conference on Dependable Systems Networks*, pages 439–448. IEEE, 2009.

# List of Figures

1.1	A Conceptual Model of Accountability . . . . .	6
1.2	A Model of an Accountability Mechanism <sup>1</sup> . . . . .	7
1.3	Abstract Architecture of the Solution . . . . .	11
2.1	Rock-Throwing Example . . . . .	21
3.1	Rock-Throwing Example . . . . .	41
3.2	Cactus Plots of Execution Time and Memory Results on the Larger Models .	47
4.1	The Program and the Solution of the Example . . . . .	60
4.2	Execution Time and Memory Results on the Larger Models . . . . .	67
4.3	Log-log Scatter Plot of $ILP$ vs $MaxSAT$ . . . . .	68
4.4	Cactus Plot including $ILP_{why}$ Execution Time and Memory Results on the Larger Models. . . . .	69
5.1	A snippet of the ILP program generated for the example . . . . .	85
6.1	An Abstract Architecture of the Solution focusing on Causal Modeling. . . .	94
6.2	An Excerpt of the Steal Master Key Causal Model . . . . .	96
6.3	Steal Key Attack Tree (drawn using ADTool [111, 60]) . . . . .	99
6.4	L-2 Unfolding . . . . .	99
6.5	L-3 Unfolding . . . . .	100
7.1	An Abstract Architecture of the Solution focusing on Causal Modeling. . . .	110
7.2	Attack Graph Generation Example . . . . .	113
7.3	Attack Graph Generator . . . . .	115
8.1	An Abstract Architecture of the Solution focusing on Contextualization. . .	126
8.2	System Architecture . . . . .	129
8.3	Steal customers' Credit Cards Tree . . . . .	131
8.4	An example of Linux Audit Rule . . . . .	131
9.1	A Process View of the Framework . . . . .	139
9.2	The Technical Framework of Actual Causality . . . . .	144
9.3	The Components of the Actual Causality Canvas . . . . .	145
9.4	The Reasoning Mode in Canvas . . . . .	147
9.5	The Causal Graph of the Example . . . . .	149

9.6	Causal Network of the Überlingen accident [187]	153
9.7	The Insider Model in Canvas	155
9.8	Causal model for Drone Crash	157
10.1	Overview of literature on (actual) causality and their relationship	165
A.1	Causal Graph of Rock-throwing Example	200
A.2	Causal Graph of Forest Fire Example	201
A.3	Causal Graph of Prisoners Example	201
A.4	Causal Graphs of Assassin Example	202
A.5	Causal Graph of Railroad Example	203
A.6	Causal Graphs of Abstract Model 1 & 2	204
A.7	Fault Tree for Leakage in Subsea Production System	206
A.8	Causal Graph of one Variant of the Binary Tree Example	208
A.9	Causal Graph of Abstract Causal Model 1 Combined with Binary Tree	209
A.10	Causal Graph of Steal Master Key Example	210
A.11	Steal Master Key Attack Tree	211

# List of Tables

3.1	Complexity of AC1, AC2, and AC3 . . . . .	29
3.2	Truth Assignments of Formulae $F$ and $G$ . . . . .	37
3.3	Execution Time and Memory Allocation of the Scenarios . . . . .	44
4.1	Checking and Semi-inference Evaluation Scenarios. . . . .	66
4.2	The Performance of a Representative Set of Scenarios for Causality Inference . . . . .	69
4.3	Summary of the Approaches . . . . .	72
5.1	Disjunctive Constraints Cases . . . . .	78
6.1	Unfolding AND . . . . .	101
6.2	Use Cases of the Evaluation . . . . .	104
6.3	Efficiency Evaluation of the Model Creation. . . . .	105
6.4	Models From HP Examples . . . . .	107
7.1	Microservice Architecture Use-cases . . . . .	118
7.2	Scalability Results with Graph Characteristics and Generation Time (s) . . . . .	120
8.1	Numbers of Log Entries using the Two Settings . . . . .	133
9.1	A Lay Model for Classifying Arthropods [141] . . . . .	148
9.2	Accident's Facts; Originally in German [187]; ordered as they appear in Figure 9.6 . . . . .	152
A.1	Evaluated Causal Models . . . . .	199
A.2	Minimal Cut Sets of the Fault Tree for Leakage in Subsea Production System . . . . .	204





# A Evaluated Models

*This appendix provides a detailed overview of our evaluated dataset. Parts of this appendix have previously appeared in publications [98, 96], co-authored by the author of this thesis.*

## A.1 Introduction

In the following, we present and describe the examples which our tool is based on. In summary we prepared 37 different causal models. On the one hand, we took all those models from [74] which consist of binary variables only. Since these examples are rather small and therefore easy to understand, they mainly serve for testing our approaches and showing that they work as expected. On the other hand, we came up with some examples on our own, obtained one from an industrial partner and considered other literature. This leads to the list of causal models shown in Table A.1. In order to give a feeling for their size, we specified the number of endogenous variables they consist of.

<b>Causal Model</b>	<b>Source</b>	<b>Number of Endogenous Variables</b>
Rock-Throwing	[81, 74]	5
Forest Fire (conjunctive & disjunctive)	[81, 74]	3
Prisoners	[81, 74]	4
Assassin (first & second variant)	[74]	3
Railroad	[74]	4
Abstract Model 1 & 2	own example	8 & 3
Steal Master Key	industrial partner	36
Ueberlingen mid-air Collision	[187]	95
Leakage in Subsea Production System	[30]	41
Leakage in Subsea Production System with Preemption	based on [30]	41
Binary Tree	own example	15 - 4095
Abstract Model 1 Combined with Binary Tree	own example	4103

Table A.1: Evaluated Causal Models

## A.2 Description of the Evaluated Models

### A.2.1 Rock-Throwing

The first model is the Rock-Throwing example explained in [79, 81, 74]. According to the authors, we can assume that Suzy and Billy both throw a rock on a bottle which shatters if one of them hits. Furthermore, we know that Suzy’s rock hits the bottle slightly earlier than Billy’s and both are accurate throwers. This leads to the endogenous variables  $ST$  (“Suzy throws”),  $BT$  (“Billy throws”),  $SH$  (“Suzy hits”),  $BH$  (“Billy hits”) and  $BS$  (“bottle shatters”). Additionally, since the authors did not explicitly specify the exogenous variables of this example, we introduce the two exogenous variables  $ST_{exo}$  and  $BT_{exo}$ . In Figure A.1, we can see the corresponding causal graph and obtain the following equations:

- $ST = ST_{exo}$
- $BT = BT_{exo}$
- $SH = ST,$
- $BH = BT \wedge \neg SH.$
- $BS = SH \vee BH$

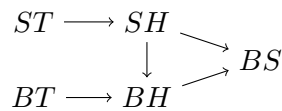


Figure A.1: Rock-throwing example (Source: [79])

### A.2.2 Forest Fire

Another one of Halpern and Pearl’s basic examples is a forest fire ( $FF$ ) that is caused by a lightning ( $L$ ) or a dropped match ( $MD$ , “match dropped”) (disjunctive scenario) or only if both occur at the same time (conjunctive scenario). Hence, he actually describes two causal models with this example. The causal graph, which is the same for both variants is depicted in Figure A.2 and the corresponding equations are as follows:

- $L = L_{exo}$
- $MD = MD_{exo}$
- $FF = L \vee MD$  (disjunctive scenario) or  $FF = L \wedge MD$  (conjunctive scenario)

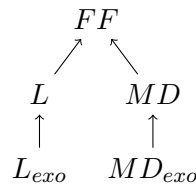


Figure A.2: Causal Graph of Forest Fire Example (Source: [81])

### A.2.3 Prisoners

An additional example found in [81] and [74] is about four prisoners. One of them dies (specified by variable  $D$ ) if prisoner  $A$  loads prisoner  $B$ 's gun which then shoots or if prisoner  $C$  both loads and shoots his gun. The equations in this causal model are straightforward; Figure A.3 shows the causal graph:

- $A = A_{exo}$
- $B = B_{exo}$
- $C = C_{exo}$
- $D = (A \wedge B) \vee C$

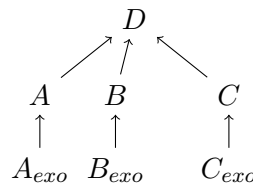


Figure A.3: Causal Graph of Prisoners Example (Source: [166])

### A.2.4 Assassin

An example very similar to the (disjunctive) forest fire example described previously is about an assassin putting poison into the coffee of its victim. However, the latter's bodyguard has an antidote for the poison which makes the victim survive. In [74], the author describes two variants of this example. In the first one, the assassin puts the poison into the coffee independently from what the bodyguard does. In the second variant, however, the assassin only then puts the poison into the coffee, if the victim's bodyguard uses his antidote. As [74] does not explicitly mention the variables within this example, we use the same ones introduced by [76], who consider this example as well. However, we specify  $A$  as "assassin does put in poison", and not "assassin does *not* put in poison",

because this makes it easier to model and understand the second variant of this example. The other variables are the same as in [76]:  $B$  “bodyguard puts in antidote” and  $VS$  for “victim survives”. Adding exogenous variables for  $A$  and  $B$ , we obtain the following equations (for both variants):

- $B = B_{exo}$
- $A = A_{exo}$  (first variant);  $A = A_{exo} \wedge B$  (second variant)
- $VS = \neg A \vee B$

The causal graph for the first variant (Figure A.4a) is structurally equal to the one of the forest fire example (Figure A.2). For the second variant, in which the assassin only then puts the poison into this victim’s coffee if the bodyguard does so with his antidote, we additionally have an edge from  $B$  to  $A$  in the corresponding causal graph (Figure A.4b).



Figure A.4: Causal Graphs of Assassin Example (Source: [166])

### A.2.5 Railroad

In this example, [74] describes an engineer that operates a switch which makes an approaching train use the right-hand track if flipped and the left-hand track otherwise. Variable  $F$  is 1 if the switch is flipped and 0 if it is not. Two additional variables  $LB$  and  $RB$  model whether the left- and right-hand track, respectively, is blocked by either being set to 1 (blocked) or 0 (not blocked). The author specifies that the two tracks finally converge. That is, the train arrives at its original destination no matter which of the tracks it took provided the respective track was not blocked. This is captured by variable  $A$ , which is 1 if the train arrives and 0 otherwise. The corresponding equations are as follows:

- $F = F_{exo}$
- $LB = LB_{exo}$
- $RB = RB_{exo}$
- $A = \neg((F \wedge RB) \vee (\neg F \wedge LB))$

Figure A.5 shows the causal graph. Unfortunately, [74] does not explicitly describe the equations; in particular not for  $A$ . Therefore, we assume that it has to be as denoted above: For  $A$  being 1 the engineer must flip or not flip the switch such that the train takes a non-blocked track provided that not both tracks are blocked. That is, it must not happen that the engineer flips the switch if the right-hand track is blocked or she does not flip it if the left-hand track is blocked.

Note that [74] describes additional variants of the railroad example. However, we do not consider them here as they are not described with the same degree of detail.

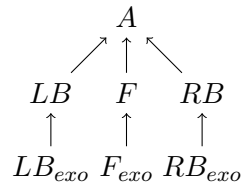


Figure A.5: Causal Graph of Railroad Example (Source: [166])

### A.2.6 Abstract Model 1 & 2

For these two models, we keep the example abstract and just provide variables and corresponding equations as well as the corresponding causal graphs (Figure A.6a & Figure A.6a).

Equations of Abstract Model 1:

- $A = A_{exo}$
- $B = B_{exo} \wedge \neg A$
- $C = A \vee B$
- $D = A$
- $E = \neg A$
- $G = \neg C$
- $H = \neg C \wedge \neg G$
- $I = C \vee D \vee E \vee G \vee H$

Equations of Abstract Model 2:

- $A = A_{exo}$
- $B = B_{exo}$
- $C = A \underline{\vee}^1 B = (A \wedge B) \vee (\neg A \wedge \neg B)$

<sup>1</sup>The operator  $\underline{\vee}$  is called *XNOR* and denotes that  $A \underline{\vee} B$  is true if both  $A$  and  $B$  are 1 or 0. Hence, XNOR is equivalent to the logical biconditional  $\leftrightarrow$ .

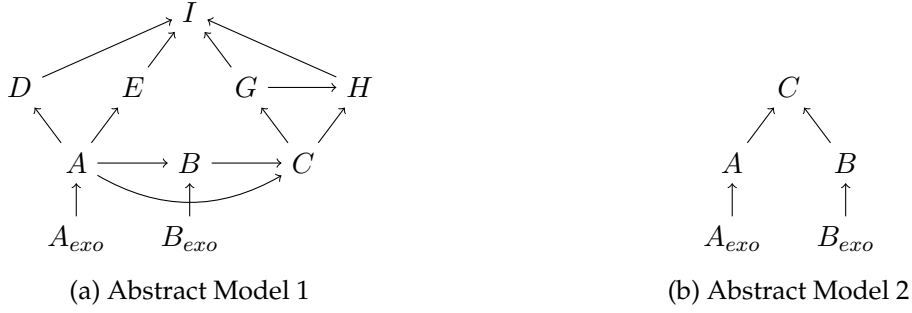


Figure A.6: Causal Graphs of Abstract Model 1 & 2 (Source: [166])

### A.2.7 Leakage in Subsea Production System

For this example, we use the *fault tree* [195] proposed by [30], in which the events that can lead to a leakage in an offshore pipeline system are modeled. We selected this example for several reasons. Firstly, the fault tree contains a relatively large amount of nodes (41 in total), i.e., the causal model is larger than some of the previous ones. Secondly, the fault tree is a real life example with semantics that are not artificially created. Hence, we can interpret causes and effects more intuitively. Thirdly, [30] did not only create the fault tree, but additionally compute its *minimal cut sets*. [195] define the latter as the “smallest combination of component failures which, if they all occur, will cause the top event to occur”. As we can see in Figure A.7, the top event of the current fault tree is the “leakage in an offshore pipeline system”. The minimal cut sets as specified by [30] are shown in Table A.2. For instance, the two events “overpressure in well” and “failure of control in

MCS	Events	MCS	Events	MCS	Events
$C_1$	$X_1, X_2$	$C_8$	$X_9, X_{11}$	$C_{14}$	$X_{16}, X_{17}$
$C_2$	$X_3, X_{11}$	$C_9$	$X_{10}, X_{11}$	$C_{15}$	$X_{18}, X_{19}$
$C_3$	$X_4, X_{11}$	$C_{10}$	$X_{12}, X_{17}$	$C_{16}$	$X_{20}, X_{21}$
$C_4$	$X_5, X_{11}$	$C_{11}$	$X_{13}, X_{17}$	$C_{17}$	$X_{22}, X_{23}$
$C_5$	$X_6, X_{11}$	$C_{12}$	$X_{14}, X_{17}$	$C_{18}$	$X_{24}, X_{25}$
$C_6$	$X_7, X_{11}$	$C_{13}$	$X_{15}, X_{17}$	$C_{19}$	$X_{26}$
$C_7$	$X_8, X_{11}$				

Table A.2: Minimal Cut Sets of the Fault Tree for Leakage in Subsea Production System (Source: [30])

well” form such a set: If they occur, the top event occurs as well. Although this notion of causality differs from the counterfactual definition of causality used within this thesis, we can use those minimal cut sets as reasonable scenarios for the evaluation of this example. For instance, we expect that “failure of control in well” is a counterfactual cause of the top event under a context such that “overpressure in well” and “failure of control in well” are the only basic events, i.e., leaf events, that occur. As they form a minimal cut set, if “failure

of control in well” does not occur anymore, the top event should not happen anymore as well. More details on the evaluated scenarios will be given below.

For the sake of readability, we denote each event within the fault tree from now on by  $X_i$  where  $i$  can be obtained from Figure A.7 (e.g., the top event is abbreviated by  $X_{41}$ ). Since the corresponding causal graph would look exactly the same as the fault tree from a structure perspective, we skip the former here and define the equations only that we obtain when transforming this example into a causal model:

- for each basic event  $X_i$  with  $i \in \{1, \dots, 26\}$ :  $X_i = X_i^{exo}$
- $X_{27} = X_3 \vee X_4$
- $X_{28} = X_5 \vee X_6$
- $X_{29} = X_7 \vee X_8$
- $X_{30} = X_9 \vee X_{10}$
- $X_{31} = X_{12} \vee X_{13} \vee X_{14} \vee X_{15} \vee X_{16}$
- $X_{32} = X_{18} \wedge X_{19}$
- $X_{33} = X_{20} \wedge X_{21}$
- $X_{34} = X_{22} \wedge X_{23}$
- $X_{35} = X_{24} \wedge X_{25}$
- $X_{36} = X_{27} \vee X_{28} \vee X_{29} \vee X_{30}$
- $X_{37} = X_{31} \wedge X_{17}$
- $X_{38} = X_1 \wedge X_2$
- $X_{39} = X_{36} \wedge X_{11}$
- $X_{40} = X_{37} \vee X_{32} \vee X_{33} \vee X_{34} \vee X_{35}$
- $X_{41} = X_{38} \vee X_{39} \vee X_{40} \vee X_{26}$

We additionally created causal model which extends the Leakage example described in the above by some preemption relations: We say that “Leakage in pipe ( $X_{39}$ )” preempts the events “Leakage in gas or oil well ( $X_{38}$ )”, “Leakage in key facilities ( $X_{40}$ )” and “Third Party Damage ( $X_{26}$ )”. In other words,  $X_{38}$ ,  $X_{40}$  and  $X_{26}$  can only occur, if  $X_{39}$  does not. Obviously, this example is made up. Nevertheless, we think it is reasonable to argue that if there is a leakage in the pipe of a pipeline system, then it is possibly counter-intuitive, if other events which might independently lead to the top event, are considered as (parts of

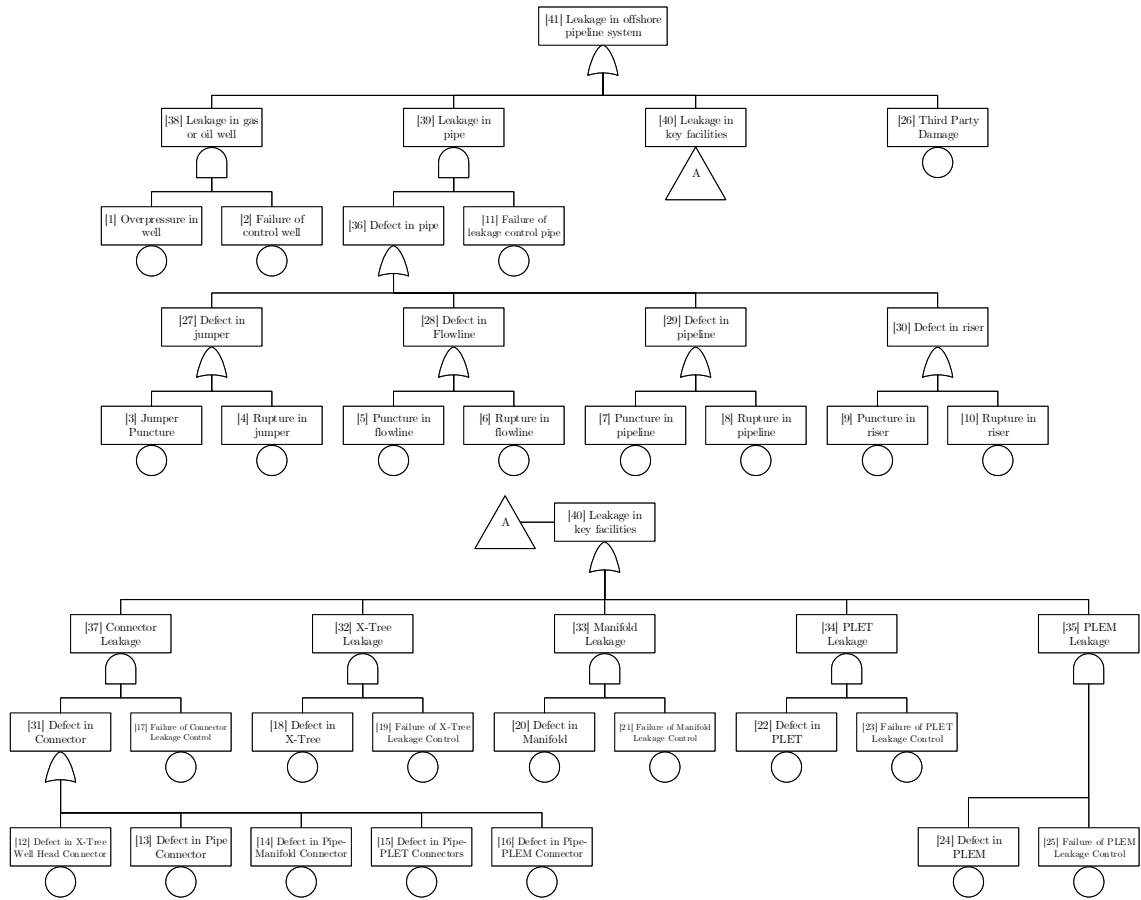


Figure A.7: Fault Tree for Leakage in Subsea Production System (Source: [30])

the) cause as well. This is similar to the Rock-Throwing example where we say we want to be able to call Suzy’s throw alone a cause for the bottle shattering even if Billy would have actually shattered the bottle when Suzy had not. Finally, we obtain the following new equations for the events preempted by  $X_{39}$ :

- $X_{26} = X_{26}^{exo} \wedge \neg X_{39}$
- $X_{38} = X_1 \wedge X_2 \wedge \neg X_{39}$
- $X_{40} = (X_{37} \vee X_{32} \vee X_{33} \vee X_{34} \vee X_{35}) \wedge \neg X_{39}$

### A.2.8 Überlingen Accident Investigation

On July 1, 2002, two aircraft (Tupolev Tu154M and a Boeing 757-200) collided in mid-air in southern Germany (Ueberlingen), killing all the people on board [149]. An investigation



by the German Federal Bureau of Aircraft Accident Investigation (BFU) and a WBA by a research group documented the many interweaving factors surrounding the accident [149, 185, 178]. Briefly, a series of coinciding events led to the collision, including an exceptional heavy load on the ground ATC, conflicting advisory commands of the ATC and the collision avoidance systems (TCAS) to the Tu154M crew, system degradation of the short term collision avoidance (STCA), and communication issues due to a maintenance operations at the ATC office.

**Causal model and context setting.** The WBA is a formal procedure introduced by Ladkin to investigate accidents and propose countermeasures for future prevention [119, 120, 178]. The result of a WBA is a graph called the why-because-graph (WBG), which structures causal factors (nodes) and their relations (edges) via analysis of official reports. The Ueberlingen WBG contained 95 factors (seen in [187]). With some adaptations, we use WBG as a source for causal models. We transform a node into the WBG to an endogenous variable in the causal model. Leaf nodes are considered as an exogenous variable and are always true because the WBA creates them from reports (actually happened). Lastly, we create the equations for each variable by manual inspection: was it conjunction or disjunction of variables that led to it? During this step, we also considered *preemption* relations, especially when events coincide. For example, two systems are implemented to avoid midair collisions—ATC aided with STCA, and *additionally*, the aircraft’s TCAS. Accordingly, the TCAS is the last resort that should resolve last-minute issues [149]. Thus, a failure by the ATC *preempts* a failure by the TCAS. Another example of preemption relations was added among the factors that led to the late ATC intervention (denoted as  $e_{49}$  in [187]). There were five coinciding factors, two of them were  $e_{56}$  *Heavy load on the ATC* and  $e_{62}$  *Crossing routes*. Arguably, people tend to consider *exceptional* events as probable causes and not as regular events [75]. Thus, we argue that the exceptional heavy load on the ATC (because of a late landing on a nearby airport and a faulty phone system) preempts other factors such as  $e_{62}$ . According to this argumentation, we added preemption relations among these events.

**Causal reasoning.** Since WBA aims to produce a list of countermeasures, we simulated our first check to automate the manual WBA sufficiency test [120], which checks if the effect eventually happens given the occurrence of all the root causes. Specifically, we checked  $Q_1$  : *Is  $\vec{X}$  a cause of the collision?* where  $\vec{X}$  is the set of 31 leaf events, which passed with an empty  $\vec{W}$ . Next, we looked for a minimal cause of the accident. Interactively, we found a minimal cause of 14 variables, which were mainly the events resulting in the ATC intervention delay. This cause conforms with the immediate cause reported by the BFU [149]. We formalized causal queries on a more abstract level of details and found minimal causes on a coarser level than the 14 events. Although knowledge around this accident already existed, the advantage of Canvas is that it automates the interactive analysis to investigate complex situations with large causal graphs. We saw how accountability is enabled by domain-specific methodologies such as WBA.

### A.2.9 Binary Tree

This example has the structure of a full binary tree and we specifically created it for measuring the efficiency of our approach. Such a model and various versions of it that exhibit a different height can be easily generated by a computer. For the sake of simplicity, we assume that the equation of each non-leave variable is defined as the disjunction of its two children. That is, the equation of  $n_1$  in Figure A.8 would be given by  $n_1 = n_3 \vee n_4$ . Analogously for  $n_{\text{root}}$  and  $n_2$ . All other variables, i.e., the leaves, are defined by an exogenous variable. Since the number of nodes in a full binary tree is  $n = 2^h - 1$  with

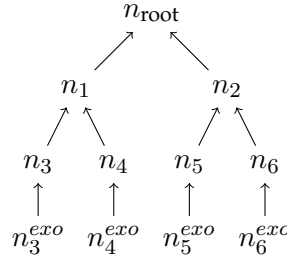


Figure A.8: Causal Graph of one Variant of the Binary Tree Example (Source: [166])

height  $h \in \mathbb{N}^2$ [39], the generation of causal models with a very high number of nodes is simple, which makes it even more interesting for benchmarking.

### A.2.10 Abstract Model 1 Combined with Binary Tree

The problem with a pure Binary Tree as causal model is that the semantics of the latter do not include preemption. Therefore, we combine two of our previous causal models, the Abstract Model 1 and a Binary Tree with  $h = 12$ . The causal graph in Figure A.9 illustrates how this combination works. Basically, we replace the equation of  $A$  of the Abstract Model 1, i.e.,  $A = A_{exo}$ , with  $A = n_{\text{root}}$ . That is, we connect  $A$  with the root node of the Binary Tree model; all other semantics of these causal models remain unchanged.

### A.2.11 Steal Master Key

The Steal Master Key example comes from an industrial partner and was originally represented as *attack tree* [179], which is shown in Figure A.11. Basically, it covers the steps an insider may perform for stealing a master key within a specific system. In particular, we assume that there exist three potential persons  $U_1, U_2$  and  $U_3$  who are able to perform the attack. The corresponding causal graph is depicted in Figure A.10 and the following equations are part of the causal model:

- $FS_{U_i} = FS_{U_i}^{exo}$  (“From Script  $U_i$ ”)

<sup>2</sup>A tree consisting of one node only has height 1. In Figure A.8, the binary tree out of which the causal graph was created has height 3.

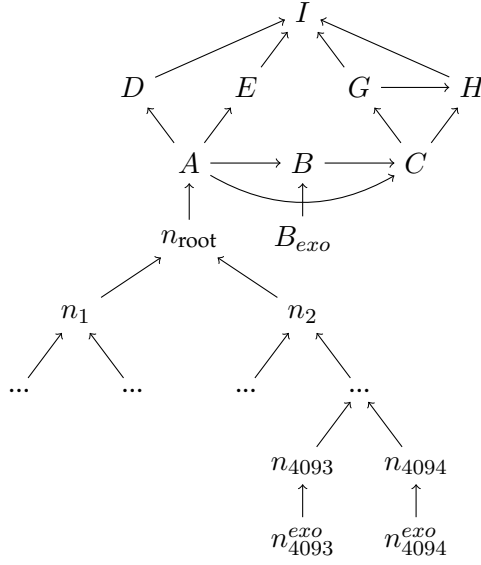


Figure A.9: Causal Graph of Abstract Causal Model 1 Combined with Binary Tree (Source: [166])

- $FN_{U_i} = FN_{U_i}^{exo}$  ("From Network  $U_i$ ")
- $FF_{U_i} = FN_{U_i}^{exo}$  ("From File  $U_i$ ")
- $FDB_{U_i} = FN_{U_i}^{exo}$  ("From Database  $U_i$ ")
- $A_{U_i} = A_{U_i}^{exo}$  ("Access  $U_i$ ")
- $AD_{U_i} = AD_{U_i}^{exo}$  ("Attach Debugger  $U_i$ ")
- $GP_{U_i} = FS_{U_i} \vee FN_{U_i}$  ("Get the Passphrase  $U_i$ ")
- $GK_{U_i} = FF_{U_i} \vee FDB_{U_i}$  ("Get the Key  $U_i$ ")
- $KMS_{U_i} = AU_i \wedge AD_{U_i}$  ("From Key Management Service  $U_i$ ")
- $DK_{U_1} = GP_{U_1} \wedge GK_{U_1}$  ("Decrypt the Key  $U_1$ ")
- $DK_{U_2} = GP_{U_2} \wedge GK_{U_2} \wedge \neg DK_{U_1}$  ("Decrypt the Key  $U_2$ ")
- $DK_{U_3} = GP_{U_3} \wedge GK_{U_3} \wedge \neg DK_{U_1} \wedge \neg DK_{U_2}$  ("Decrypt the Key  $U_3$ ")
- $SD_{U_1} = KMS_{U_1}$  ("Steal Decrypted  $U_1$ ")
- $SD_{U_2} = KMS_{U_2} \wedge \neg SD_{U_1}$  ("Steal Decrypted  $U_2$ ")
- $SD_{U_3} = KMS_{U_3} \wedge \neg SD_{U_1} \wedge \neg SD_{U_2}$  ("Steal Decrypted  $U_3$ ")
- $DK = DK_{U_1} \vee DK_{U_2} \vee DK_{U_3}$  ("Decrypt the Key")
- $SD = SD_{U_1} \vee SD_{U_2} \vee SD_{U_3}$  ("Steal Decrypted")
- $SMK = DK \vee SD$  ("Steal Master Key")

for  $i \in \{1, 2, 3\}$

All variables can obtain Boolean values only that denote whether the respective event occurred. As we can see, for stealing the master key ( $SMK$ ), we need to decrypt it ( $DK$ ) or steal it in decrypted form ( $SD$ ). For doing so an attacker  $U_i$  needs to either get the passphrase ( $GP_{U_i}$ ) and the key itself ( $GK_{U_i}$ ) or get the decrypted key from the key management service ( $KMS_{U_i}$ ). The passphrase can be obtained from a script ( $FS_{U_i}$ ) or from the network  $FN_{U_i}$ , while the key may be extracted from a file ( $FF_{U_i}$ ) or a database  $FDB_{U_i}$ . For stealing the decrypted key from the key management service, an attacker  $U_i$  needs to have access to it ( $A_{U_i}$ ) and additionally attach a debugger ( $AD_{U_i}$ ). Notice that we implicitly assume that attackers do not collaborate, i.e., the master key can only be stolen if one attacker alone performs all steps necessary. Additionally,  $U_1$  preempts  $U_2$  and  $U_3$  and  $U_2$  preempts  $U_3$ . That is, even if  $U_2$  or  $U_3$  were able to get the key and the passphrase or were able to obtain the key from the key management service, we say that their attack is only successful, if  $U_1$  did not decrypt the key ( $DK_{U_1}$ ) or steal the decrypted key ( $SD_{U_1}$ ). Analogously for the preemption of  $U_2$  towards  $U_3$ . Note that these preemption relationships are not modeled in the original attack tree in Figure A.11.

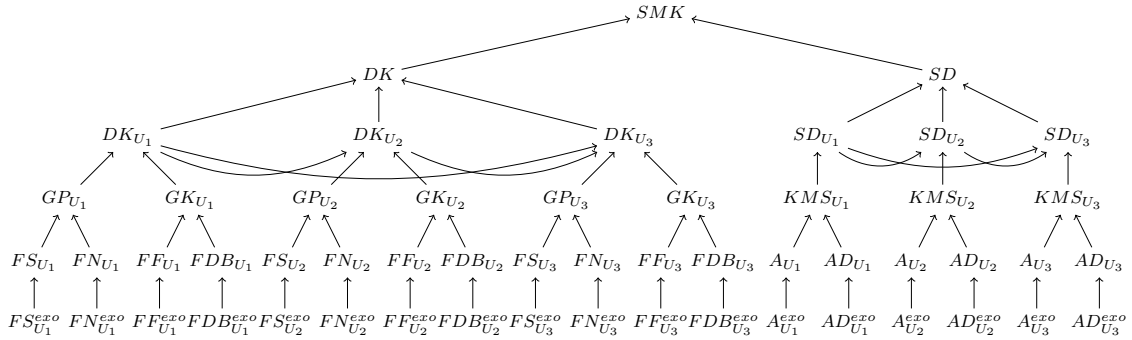


Figure A.10: Causal Graph of Steal Master Key Example (Source: [166])

In addition to the standard model with three insiders, we created another 13 other variant of the Steal Master Key model with varying number of users ranging from 50 to 650 (based on the employees of the partner). The general structure and semantics remain the same, but we now have models of  $11 \times n$  endogenous variables (and  $10 * n$  exogenous variables), where  $n$  is the number of employees. Regarding the preemption relationships,  $U_1$  now preempts  $U_2, \dots, U_8$ ,  $U_2$  preempts  $U_3, \dots, U_8$  and so on and so forth.

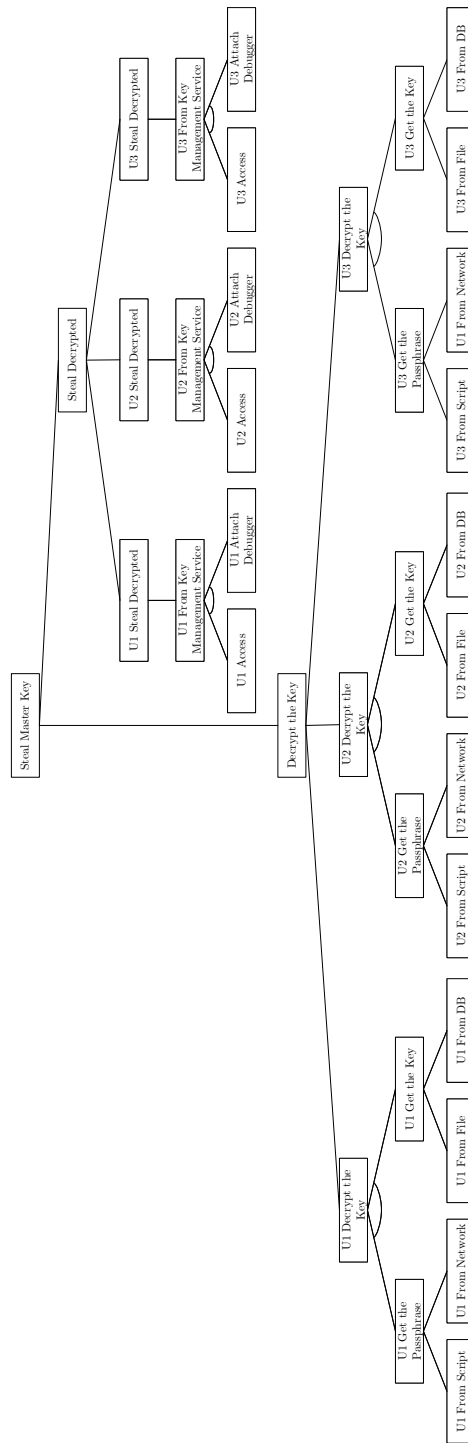


Figure A.11: Steal Master Key Attack Tree (Source: Industrial Partner); graphic source [166]