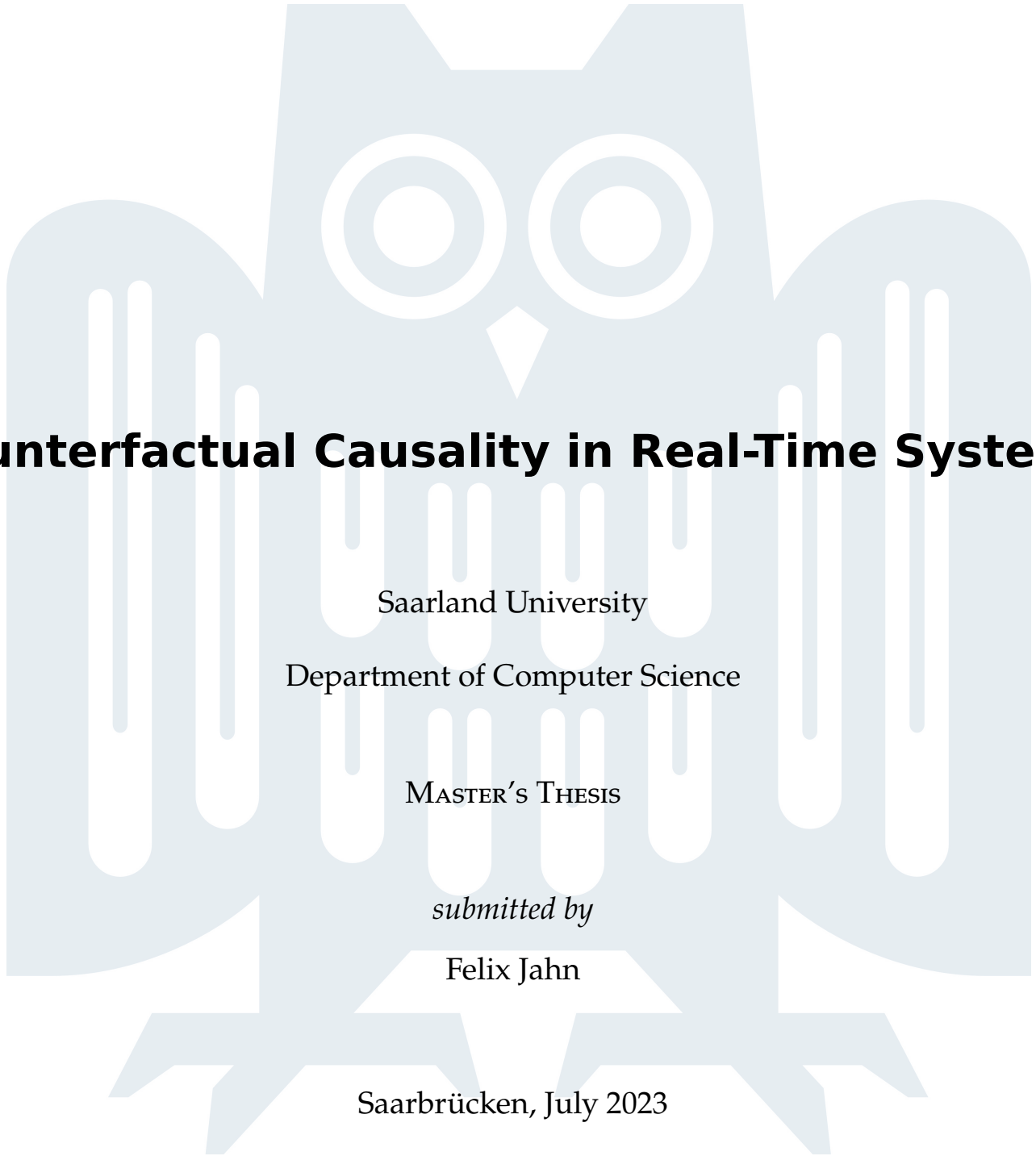


Counterfactual Causality in Real-Time Systems

Saarland University

Department of Computer Science

Master's Thesis

submitted by

Felix Jahn

Saarbrücken, July 2023

Supervisor:	Prof. Bernd Finkbeiner, Ph.D.
Advisor:	Julian Siber
Reviewer:	Prof. Bernd Finkbeiner, Ph.D.
	Dr. Anne-Kathrin Schmuck
Submission:	July 04, 2023

Abstract

Verifying specifications of computer systems is a well-explored task addressed, for instance, by model checking. Understanding a negative model checking result is, however, difficult. In such an instance, a counterexample witnessing the violation will be returned but yields only limited insights for debugging.

A promising technique to foster understanding of counterexamples is to generate explanations based on causal analyses. For real-time systems – which have to satisfy crucial real-time constraints and are increasingly deployed in various safety-critical contexts like traffic, manufacturing, or healthcare – causal analyses, however, are especially hard: not only the performed actions of the systems but also their real-time behavior contributes to the overall system behavior.

In this thesis, we present multiple notions of counterfactual causality for real-time systems that consider both actions and real-time behavior for potential causes. This then yields precise explanations for observed effects in the executions of the systems, which we model as timed automata. Our definitions are based on the seminal conceptual work on causality by Halpern and Pearl. We adapt their ideas for counterfactual reasoning to the real-time setting by representing counterfactual executions using counterfactual trace automata. This enables us to define progressively more refined causality notions and to thereby adopt different perspectives on the timing behavior of real-time systems.

We demonstrate the destined notions on numerous examples, present some basic theoretical properties, and provide algorithms for checking computing causal relationships. Furthermore, we analyze the complexity of both problems, which we show to be EXPSPACE-complete. Lastly, we report on the implementation of a Python tool for checking and computing causes for real-time systems modeled in Uppaal, enabling an automated explanation of counterexamples.

Acknowledgements

My sincere thanks go to Julian, who advised me closely from the very start of this project. Uncountable hours were spent together at the – always completely scribbled – whiteboard, uncountable possible causality notions were created, and uncountable philosophical discussions developed, giving me uncountable “aha”-experiences about the causal character of time. Thanks for the diverse scientific advice, for awaking my enthusiasm for the topic of causality, and for all the time spent bending my scarily nested English sentences by giving always great writing feedback – I’m afraid when reading these lines, you are also already eagerly waiting for a point again, so once again very precise: Thank you, Julian!

The supervision of this thesis – the last examination in my study – by Prof. Finkbeiner is, in some way, a closing circle as he also gave the very first computer science course I attended at university, the good old Programming 1. Apparently, he was able to excite me about computer science, and, almost six years later now, I am standing here and would like to thank him for his professional advice whenever necessary and scientific freedom whenever possible. Thanks in advance to him as well as to Anne-Kathrin Schmuck for reviewing the thesis.

I can only be grateful for my wonderful proofreaders, who gave me another boost of motivation in the last few days. Thanks to Dominik, Julia, and Lukas for your helpful remarks and a big sorry for the narrow time window I gave you! Particularly, I want to thank Yannick and Dominik for the mentoring you offered me throughout the years, that was, however, and especially exceptionally, actually not even close to being limited to scientific insecurities.

Thanks to all those who always helped me throughout my studies with my panic and clumsiness to install even the simplest things. Special thanks in this regard to Luca, but of course not only in this regard, hard to grasp all the things we have experienced in all the years. I cannot thank enough and definitely not name all the amazing people that make Saarbrücken what it means to me. Un merci particulier à tous les enfants terribles de la ville, on reste fou! And deeply connected greetings to all those near and far who do not stop dreaming. Very heartfelt thanks go as well to my in such short time insanely cherished roommates for always taking care of "De Klään", especially in the last days.

Whether as “Kläänа” or for some years now as “Großer”, I could always count on your full support in all areas and situations of life. How you have helped me along my educational journey, ink snakes slithering through lichen, mosses, and ferns probably are the best report. Mom, Dad, from the bottom of my heart, thank you both!

...as forever the sun always rises again!

Eidesstattliche Erklärung
Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath
I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung
Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent
I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 04 July, 2023

Erklärung
Ich erkläre hiermit, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement
I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Saarbrücken, 04 July, 2023

Contents

Chapter 1

Introduction

Gaining knowledge about the behavior of computer systems is a common goal in many areas of computer science research and of verification-oriented formal methods in particular. For various kinds of systems, gaining knowledge on *how* a certain system behaves is a well-explored task: In the past decades, different methods like testing, model checking, or deductive verification have proven their capability to provide guarantees about system behaviors.

A more recent line of research is concerned with the task of gaining knowledge on *why* systems behave in a certain way. Finding explanations – in the various ways this term might be understood – for the behavior of systems is of steadily increasing interest: Not only is such an understanding crucial to identify reasons for unexpected system behaviors, easing, for instance, system repair, providing explanations for the output of a system is also a desirable feature in itself as computer systems are increasingly entrusted with far-reaching decisions concerning people's lives.

This interest particularly applies to real-time systems. These systems find crucial application in contexts where the timing of the involved systems is crucial for correctness. Such contexts arise, for instance, in various traffic or healthcare applications in which processed data or events underlie certain time constraints.

Explaining the behavior of such real-time systems comes with an additional layer of complexity: Not only the performed actions but also the variable timing in the execution of the system might influence its overall behavior. Therefore, attempts in finding system explanations by playing through possible combinations of the actions and timing in the course of a system execution manually will quickly collapse. But also for discrete-time systems, gaining knowledge of the reasons for their behavior gets quickly manually infeasible with scaling system size. Consequently, recent research attempting to automatize this process using different ideas, approaches, and methods has begun.

Causality. The originally philosophical concept of *causality* was for this research of particular relevance from the very start. Loosely speaking, causality is concerned with the influence that certain objects have to the occurrence of another object. In the course of centuries, various concrete notions of causality emerged, differing especially in the way they captured the informal idea of "the influence to production".

Especially an approach called *counterfactual causality* gained particular popularity. Counterfactual causality tries to draw conclusions on what objects have an influence on the production of another object based on so-called counterfactual reasoning. Going back to considerations of David Hume already in the 18th century, he described the principle of counterfactual reasoning as follows:

> *"If the first object had not been, the second never had existed."*
>
> – David Hume[1] –

The "first object" is then understood as a cause for the occurrence of the "second object".

Retrieved by Lewis in the 1970s [53], it was then Halpern and Pearl who published a formalized notion of counterfactual causality [40] and gave the concept eminent attention in computer science, especially in those fields that work in a formal manner. Using a set of equations, Halpern and Pearl represent the causal behavior of the world in a causal model and consider certain events in the world ("first object") that might cause a certain effect ("second object"). For this to hold, they require three conditions to be fulfilled: (1) The events and the effect indeed occurred – the *satisfiability condition*, (2) if the events had not occurred, also the effect had not occurred – the *counterfactual condition* capturing exactly the idea of Hume, and (3) the events are minimal in this regard – the *minimality condition*.

As a further notable contribution of Halpern and Pearl, they extend the basic principle of counterfactual reasoning, referred to as *but-for causality*, in their formalization of the counterfactual condition with the idea of contingencies to tackle the so-called preemption problem. In a nutshell, the preemption problem occurs when an event causing an effect preempts a second event that would cause the effect as well. In this case, the first event will by the basic counterfactual reasoning not be recognized as a cause, as assuming the event to not occur leads on the occurrence of the second event, and this in turn still on the occurrence of the effect. Contingencies however allow fixing certain parts in the counterfactual reasoning to be as they had been in the actual world – in the case of the preemption problem to fix the non-occurrence of the second cause. With this refined causality concept named *actual causality* [38], the first event will now be recognized as a cause.

Error localization. Causality was successfully applied in various approaches to find explanations for the behavior of computer systems: Different kinds of explanations were obtained for different kinds of systems or system behaviors [35, 32, 22]. Influential

[1] In his book on *Philosophical Essays Concerning Human Understanding* [46].

in this regard was the work of Beer et al. who developed – for their setting of discrete Kripke structures – a notion of counterfactual causality in the flavor of Halpern and Pearl [16]. They used their adapted notion of but-for causality (i.e., without contingencies) to explain violations of *Linear Temporal Logic* (LTL) properties: Given a system execution violating a given LTL formula, they identified events in the execution that caused the violation. Finding such kinds of explanations is also subsumed under the term of *error localization* [14].

Real-time systems. While there is, as mentioned already, quite a broad range of developed causality notions for various kinds of computer systems, causality and error localization in the setting of real-time systems have been studied only sparsely. The class of *real-time systems* [55] describes computer systems that need to satisfy crucial real-time constraints regarding, for instance, their timing behavior relative to other systems or resulting from temporal deadlines for the computation of output that the systems should fulfill. With the capability to capture such timing behaviors, real-time systems gained increasing importance in various fields of application: In manufacturing, the correct temporal interaction between different processes is essential for the overall outcome, healthcare treatments are subject to critical time-related restrictions, and also in various traffic contexts like railway crossings or autonomous transportation, the correct dynamical timing with respect to (other) traffic participants is of indispensable importance.

A common formalism for modeling such real-time systems are *timed automata* [8, 17]. Timed automata express real-time constraints using real-valued clocks that measure certain times throughout the system execution, that can be updated, and on whose current values the further execution may depend. In contrast to discrete systems, the concrete delays spent in locations between the execution of subsequent actions can vary non-deterministically or depending, for instance, on the timing of received input.

Especially for this reason, finding explanations for the behavior of real-time systems is a particularly difficult task. Consider, for instance, the following real-time example from the medical sector: After a surgery, two further treatment options T1 and T2 can be conducted. For the optimal healing process of a specific disease, it is crucial that treatment T1 is applied exactly six hours after the surgery. However, imagine now that a doctor decides to apply treatment T2 twelve hours after the surgery which leads to a non-optimal healing process. Intuitively, we would declare both the false treatment as well as the time of the treatment application as aspects of this scenario that caused the non-optimal healing. For explaining the behavior of real-time processes it is, hence, essential to take both the performed actions as well as the timing component into account. Thereby, a major challenge arises from the infinite number of possible timing alternatives as known results on decidability and complexity of Halpern and Pearl's causality consider only models with finitely many alternative events [7].

The methodical research to overcome this obstacle for gaining explanations of real-time scenarios is, as mentioned, still in its infancy. Early causality-related works on real-time systems head into the direction of so-called system repair [64, 49], an approach that tries to identify the faulty parts of a system that caused a faulty behavior. In the field of error localization, existing works analyze the influence solely of certain delays or delay ranges to the execution [50], or apply other causality concepts using no counterfactual reasoning [56].

Contributions. This thesis aims to contribute to the study of causality in the context of real-time systems modeled as timed automata: We develop several notions of causality in the counterfactual flavor of Halpern and Pearl that consider *both*, performed actions and the timing of the system, to find explanations for an observed system behavior. Given a run of an automaton, we identify the actions and real-time events in the run that were causal for a certain effect. That is, we apply causality to obtain a form of error localization.

Those causal conclusions are drawn based on counterfactual reasoning that simulates the possible counterfactual executions of the system with respect to potential causes. We, therefore, take an automata-theoretical approach and use for the counterfactual simulation, following the ideas of Coenen et al. [22], the concept of *counterfactual trace automata*. Informally, a counterfactual trace automaton for a certain set of events represents exactly the possible counterfactual traces, i.e., those traces that allow alternative actions or timings for the events in the given set while the events not in the set get enforced to remain unchanged. If there is then a counterfactual trace in whose corresponding counterfactual run the effect is no longer present, we classify the events as a cause for the effect in the original run. Those effects in runs of timed automata are described using *Metric Interval Temporal Logic* (MITL) [9], a decidable – and therefore model checkable – extension of *Linear Temporal Logic* (LTL) to the real-time setting. Crucially, representing counterfactual simulations in terms of automata enables us to reduce the problem of checking causes to the MITL model-checking problem.

Furthermore, again based on a careful automata construction, we succeed in adding contingencies to the counterfactual reasoning to obtain not only a notion of but-for causality but even of actual causality in real-time systems. Again inspired by Coenen et al. [22], we do so by constructing *contingency automata* that in every step allow counterfactual runs to reset their configuration as it was in the respective step of the actual run. As in the foundational setting of Halpern and Pearl, contingencies solve, at least for some scenarios, also in our real-time setting the problem of preemption.

Early in the study of real-time causality, we noticed that the perspective taken on real-time events crucially affects the causal analysis. We thereby distinguish two perspectives: A *delay perspective* that considers the delays spent between two actions in the locations of the timed automaton as time events, and a *timestamp perspective* that considers the global timestamps of actions, that is, the overall time since the start of

the automaton as time events. For both perspectives, we present causality notions, that reflect the differing perspectives on events in differing proceedings for the counterfactual reasoning and compare the two notions in terms of how they perform in different scenarios.

In general, we will consider numerous examples throughout the thesis. This is in line with a remarkable statement by Joseph Halpern about a peculiarity in the theoretical research on causality and the process of developing definitions of causality:

"What's really hard about this area, I'm gonna give you a definition and would love to be able to prove a theorem that says: "This is the right definition." I'm a theoretician, that's what I do, I prove theorems. If I only knew the statement of the theorem, I might have a shot of proving it, but I don't know what it means to have the "right" definition of causality. I don't know what the theorem should say. [...] *The way you show your definition is good is to show that - gee - look at how well it does it all in all the examples."*

– Joseph Y. Halpern[2] –

Unsurprisingly, Halpern's observation applies to our real-time setting as well: We do not know how to prove our suggested causality notions to be "right", we can only demonstrate their usefulness by showing them to do the "right" thing in various examples, that is, that they detect causes in accordance with our intuitive causal judgments. We will do so for demonstration purposes mainly on synthetic examples but will also present some causal analysis of (simplified) real-world scenarios.

We additionally support our definitions by showing that they fulfill some basic properties that we would expect to hold in meaningful notions of causality. Moreover, we prove our notions to be decidable and computable by giving algorithms for those tasks and proving their correctness. For the computation of minimal causes, we present an optimized algorithm for computing causes. Showing this optimized algorithm to compute exactly the intended causes turned out to require a quite involved correctness proof. We furthermore report on the computational complexity of checking and computing causes and conclude via a reduction from model checking that the cause-checking problem is EXPSPACE-complete.

As a last major part of our work, we implemented a Python tool for checking and computing causes in the sense of our causality definitions. The project as well as installation and usage instructions can be found at

https://github.com/FelixJahnFJ/Real-Time-Causality-Tool.

[2]Recording of his talk at AAAI 2018: *Actual Causality: A Survey* [39].

The tool relies on the modeling and verification software for real-time systems Uppaal [2] and the Python library Pyuppaal [1] allowing to use most of Uppaal's functionalities in Python. More concretely, the tool processes real-time systems and effect descriptions specified in Uppaal. The key in the development was then to implement – based on self-written data structures for timed automata and timed traces – all the formally introduced automata operations like automata intersection, and the construction of counterfactual trace automata and contingency automata. Together with the discussed algorithmic methods and use of Uppaal's model checker, this enables behavior explanation of real-time systems by highlighting causal action and time events in the execution. Thus, we provide a fully automatized error localization technique for real-time systems.

Due to the dependence on Uppaal, the tool inherits its limitations. In particular, Uppaal's restricted specification language allows the tool to handle effects only for a small fragment of MITL. Nonetheless, we tested and experimented with the tool in various examples and measured its runtime performance extensively in different scenarios. Especially, we want to emphasize that all the examples presented in this thesis are available for the tool as well and can be accessed under

https://github.com/FelixJahnFJ/Real-Time-Causality-Tool/tree/main/Thesis_Examples.

Outline. This thesis is organized in the following manner: We start with preliminary remarks on Halpern and Pearl causality, real-time systems, and the real-time logic MITL in Chapter 2. Chapter 3 forms the heart of our formal development of causality notions in the real-time setting: we introduce the different definitions, show basic properties for them, present how they apply in numerous examples, and discuss further aspects and limitations of the notions. In Chapter 4, we focus on the discussion of the algorithms for checking and computing of causes, while Chapter 5 reports in more detail about the developed causality tool. In Chapter 6, we report on related work in the field of causality in computer science contexts and particularly real-time systems. Lastly, we comment on possible lines of future research in Chapter 7 and conclude with some final remarks in Chapter 8.

Chapter 2

Preliminaries

We start the theoretical presentation with preliminary notes about the concept of counterfactual causality in Section 2.1, real-time systems in Section 2.2 and their corresponding logics in Section 2.3.

→ Section 2.1, p. 7
→ Section 2.2, p. 13
→ Section 2.3, p. 20

2.1 Counterfactual Causality

The goal of the thesis is to introduce notions of counterfactual causality for real-time systems, hence, we first address some foundational background of causality. More precisely, we look at the counterfactual concepts of actual causality by Halpern and Pearl and its precursor, the more basic but-for causality. Originally published in 2001 [40], the definition of actual causality was refined and improved multiple times [41, 37] so that we focus on the latest definitions by Halpern from 2015. We thereby content to give the key intuition behind the important ideas in Halpern's work. For a more detailed discussion of the foundations of counterfactual and actual causality as well as various examples, we refer to the above cited publications, to Halpern's 2016th textbook "Actual Causality" [38], and especially to the recording of Halpern's excellent talk at AAAI-18 [39].

2.1.1 Causal Models and Formulas

First, we discuss Halpern and Pearl's formal causal setting we are working in. Informally speaking, we describe the world and its causal behavior by variables that affect each other depending on their values. We thereby split the variables in *exogenous variables*, whose values are taken as given (e.g., values that are determined by factors outside the model) and *endogenous variables*, that might be assigned to varying values and are

Def. exogenous variable

Def. endogenous variable

therefore the variables one considers as (parts of) potential causes[1].

Def. signature

Formally, we use a *signature* $S = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ that explicitly lists the exogenous variables $\mathcal{U}$, the endogenous variables $\mathcal{V}$ and for every variable $Y \in \mathcal{U} \cup \mathcal{V}$ its associated possible values $\mathcal{R}(Y)$. The *causal model* $M = (\mathcal{S}, \mathcal{F})$ consists then in addition to the signature also of a set of *structural equations* $\mathcal{F} = \{F_X \mid X \in \mathcal{V}\}$. For an endogenous variable $X \in \mathcal{V}$, the function $F_X : (\times_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\times_{Y \in \mathcal{V} \setminus \{X\}} \mathcal{R}(Y)) \to \mathcal{R}(X)$ specifies the value of X given the values of all other variables in $\mathcal{U} \cup \mathcal{V}$.

Def. causal model

Def. structural equation

External interventions in a causal model M are then mimicked by setting the values of some variable X to a fixed value x. This results in a new causal model $M_{X \leftarrow x}$, that is identical to M, except that the equation F_X is replaced by $X = x$. Interventions at multiple variables are then defined recursively as

$$M_{[X_1 \leftarrow x_1, \ldots, X_n \leftarrow x_n, X_{n+1} \leftarrow x_{n+1}]} := M_{[X_1 \leftarrow x_1, \ldots, X_n \leftarrow x_n][X_{n+1} \leftarrow x_{n+1}]}.$$

Def. recursive model

Like Halpern, we restrict the theory to *recursive models*, that is, models with a total ordering $\prec$ on the endogenous variables for which $X \prec Y$ implies X to be independent of Y, i.e., the value of Y cannot affect the value of X. In recursive models, the structural equations have a unique solution for a given *context* $\vec{u} \in (\times_{U \in \mathcal{U}} \mathcal{R}(U))$, that is, a value setting for the exogenous variables $\mathcal{U}$.

Def. context

To reason about causality, we consider a small language of causal formulas: For a signature $S = (\mathcal{U}, \mathcal{V}, \mathcal{R})$, *primitive events* of the form $X = x$ with $X \in \mathcal{V}$ and $x \in \mathcal{R}(X)$ are used to build *causal formulas* of the form $[\vec{Y} \leftarrow \vec{y}]\varphi$, whereby

Def. primitive event

Def. causal formula

- φ is a Boolean combination of primitive events,
- $\vec{Y} = (Y_1, \ldots, Y_k)^\top$ with distinct variables $Y_1, \ldots, Y_k$ in $\mathcal{V}$, and
- $\vec{y} = (y_1, \ldots, y_k)^\top$ with $y_i \in \mathcal{R}(Y_i)$ for all $i = 1, \ldots, k$.

In the case of $k = 0$, we simply write φ.

The satisfiability relation of a causal formula ψ is then defined in a causal model $M = (\mathcal{S}, \mathcal{F})$ with a given context $\vec{u}$ inductively on the formula:

- $(M, \vec{u}) \models X = x$ for a primitive event $X = x$, if the unique solution of F_X in the model M in context $\vec{u}$ is x,
- $(M, \vec{u}) \models \varphi$ for a Boolean combination of primitive events φ is defined in the for Boolean operations expected way, and
- $(M, \vec{u}) \models [\vec{Y} \leftarrow \vec{y}]\varphi$, if $(M_{\vec{Y}=\vec{y}}, \vec{u}) \models \varphi$ for a primitive event $X = x$.

Hence, a causal formula $[\vec{Y} \leftarrow \vec{y}]\varphi$ is true in a model M, if φ would be true after an external intervention that sets $\vec{Y}$ to $\vec{y}$.

[1] In fact, exogenous variables are in several examples from the literature even fully omitted.

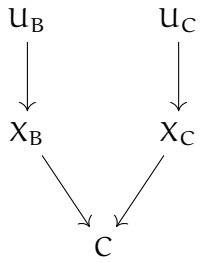
Example 2.1.1. We want to represent the process of a person using a coffee machine as a causal model. The machine requires to push both a button and to insert a coin in order to produce coffee. The order of pushing the button and inserting a coin does not matter. We choose the following variables:

- Exogenous variables U_B and U_C with $U_B = 1$ iff the person intends to push the button and $U_C = 1$ iff the person intends to insert a coin,
- endogenous variables X_B and X_C with $X_B = 1$ iff the button is pushed and $X_C = 1$ iff a coin is inserted, and
- endogenous variable C with $C = 1$ iff coffee is produced.

The intended causal behavior can then be modeled with the following structural equations for the endogenous variables:

- $F_{X_B} = U_X$,
- $F_{X_C} = U_C$,
- $F_C = X_B \wedge X_C$.

The above signature and structural equations give then the causal model M. The structural dependencies in this model can be visualized in the following way:



For a context $\vec{u}$, we can then, for instance, express that coffee is produced in this context by $(M, \vec{u}) \models C = 1$. △

2.1.2 But-For Causality

With the previous definitions in hand, we are ready to head to the different notions of causality. Informally, we always want to express what it means for a set of events to be a cause for some effect. For this being the case, we require three conditions, a satisfiability, a counterfactual, and a minimality condition, that informally state the following:

SAT The events and the effect indeed occurred (in the actual world).

CF In the counterfactual world in which the events did not occur, also the effect does not occur.

MIN The set of events is minimal, i.e., no strict subset satisfies **SAT** and **CF**.

The **CF**-condition forms thereby the heart of the definition – and the condition in that different notions of counterfactual causality vary. The most basic counterfactual concept we discuss is the so-called but-for causality, capturing exactly the basic intuitive idea behind counterfactual reasoning, namely to argue that if the causal event(s) had not happened, the effect would also not have happened. With respect to a certain set of events, we refer to events in this sets as "causal events"[2].

In our setting given a causal model M and context $\vec{u}$, we consider

Def. set of events
- as *set of events* the assignment of a set of endogenous variables, that might therefore also be represented in vector notation $\vec{X} = \vec{x}$ or as an conjunction of primitive events $X_1 = x_1 \wedge \cdots \wedge X_n = x_n$, and

Def. effects
- as *effects* a Boolean combination φ of primitive events.

This leads us to the formal definition of but-for causes, initially going back to the ideas of Lewis in 1973 [54]. Since but-for causes in their original flavor do not require minimality, we introduce the separate terminology of minimal but-for causes.

Definition 2.1 (But-For Causality)

Def. but-for cause
$\vec{X} = \vec{x}$ is a *but-for cause* of φ in $(M, \vec{u})$, if the following two conditions hold:

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

CF$_{BF}$ There is a setting $\vec{x}'$ of the variables in $\vec{X}$ such that $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}']\neg\varphi$.

Def. minimal but-for cause
We call the cause *minimal but-for cause* if, furthermore, the following condition holds:

MIN $\vec{X}$ is minimal, i.e., no strict subvector of $\vec{X}$ satisfies **SAT** and **CF$_{BF}$**.

As expressed in the **CF$_{BF}$**-condition, but-for causes ask in the counterfactual reasoning for an external intervention, i.e., an alternative variable setting $\vec{x}'$ of the causal variables $\vec{X}$, that prevents the effect to occur. We also call this alternative variable setting $\vec{x}$ a *witness* for the fact that $\vec{X} = \vec{x}$ is a cause of φ. The **SAT**- and **MIN**-condition are straightforward formalizations of the informal description above.

Def. witness

→ Example 2.1.1, p. 9

Example 2.1.2. Recall the above Example 2.1.1 of the coffee machine. We discuss potential causes for the effect $C = 1$ given a context $\vec{u} = (1, 1)$, i.e., potential causes for the producing of coffee when the person intends to push the button and to insert a coin.

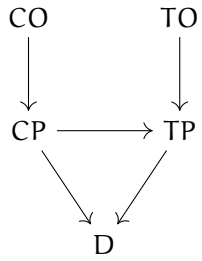
[2] Which might actually be a bit misleading, as we do so regardless of whether or not the set of events is indeed a cause.

- $X_B = 1$ and $X_C = 1$ are both minimal but-for causes for $C = 1$ in $(M, \vec{u})$: $(M, \vec{u}) \models X_B = 1$ and $(M, \vec{u}) \models C = 1$, hence, the **SAT**-condition holds. To reason for $\mathbf{CF_{BF}}$, we consider the alternative setting $x'_B = 0$ and have $(M, \vec{u}) \models [X_B \leftarrow 0] \neg C = 1$. The **MIN**-condition is always satisfied for singleton but-for causes. Showing $X_C = 1$ to be a minimal but-for cause works analogously.
- $X_B = 1 \wedge X_C = 1$ is a but-for cause, but no minimal but-for cause for $C = 1$: The **SAT**-reasoning works similar as above and the $\mathbf{CF_{BF}}$-condition is fulfilled, for instance, with the witness $X'_B = 0 \wedge X'_C = 0$. Minimality does not hold since there are as shown above strict subvectors satisfying **SAT** and $\mathbf{CF_{BF}}$. △

While but-for causality handles different kinds of real-world examples – provided they are modeled in a suitable way – as desired and in agreement to the intuition, this simple concept fails if the the causal event(s) preempt another potential cause. In those cases, the further potential cause then occurs in the counterfactual reasoning and, therefore, also the effect is still present such that the mere absence of the causal event(s) does not suffice to prevent the effect. Hence, the cause is not recognized as such, as it is demonstrated in the following example.

Example 2.1.3. We consider a drink machine, where both coffee and tea are ordered immediately after each other. If a drink is ordered, the machine starts to produce the respective drink. However, the machine cannot produce different drinks at the same time, hence, only the first order (in this case coffee) can be produced. We are interested whether the machine pours out a drink, which is the case after coffee or tea was produced. The scenario is modeled in the following way, where exogenous variables are fully omitted:

- CO stands for "Coffee is ordered", CP for "Coffee is produced", hence, $CP = CO$.
- TO and TP have the corresponding meaning for tea. However, tea is only produced if also coffee is not produced, i.e., $TP = TO \wedge \neg CP$.
- D stands for "A drink is poured out". Here, it does not matter which kind of drink, i.e., $D = CP \vee TP$.



$CO = 1$ is no but-for cause for $D = 1$ since the $\mathbf{CF_{BF}}$-condition is not fulfilled: Even with the potential witness $CO = 0$, we have $(M, \vec{u}) \models [CO \leftarrow 0] D = 1$. △

The example illustrates a deviation from our intuition as we would expect the ordering of coffee to be a cause for a drink to be poured out. This is exactly due to the preemption of the further potential cause of producing tea.

2.1.3 Actual Causality à la Halpern and Pearl

In order to capture also this broader range of examples, Halpern and Pearl refine the notion of but-for causality by the idea of so-called contingencies. Contingencies allow to fix parts of the counterfactual world to be as they had been in the actual world. More concretely, we allow further potential causes to not take effect in the counterfactual reasoning, such that now the absence of the actual cause indeed prevents the effect. This more involved concept of counterfactual causality is then called actual causality. We define it by only extending the $\mathbf{CF}_{\mathsf{BF}}$-condition with contingencies to obtain a more advanced **CF**-condition, now denoted $\mathbf{CF}_{\mathsf{Act}}$, the other two conditions remain unchanged.

Definition 2.2 (Actual Causality à la Halpern and Pearl)

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

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

$\mathbf{CF}_{\mathsf{Act}}$ There is a set $\vec{W} \subseteq \mathcal{V}$ of endogenous variables with setting $\vec{w}$, i.e., $(M, \vec{u}) \models \vec{W} = \vec{w}$, and there is a setting $\vec{x}'$ of the variables in $\vec{X}$ such that $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}]\neg\varphi$.

MIN $\vec{X}$ is minimal, i.e., no strict subvector of $\vec{X}$ satisfies **SAT** and $\mathbf{CF}_{\mathsf{Act}}$.

The set $\vec{W}$ in the $\mathbf{CF}_{\mathsf{Act}}$-condition formalizes exactly the aforementioned concept of contingencies. We fix its actual values $\vec{w}$ for the counterfactual reasoning by reasoning in the new causal model $M_{\vec{W}\leftarrow\vec{w}}$ such that the alternative setting x' can then no longer affect the values of $\vec{W}$. Similar to witnesses of but-for causes, we now say the tuple **Def.** witness $(\vec{W}, \vec{w}, \vec{x}')$ to be a *witness* for the fact that $\vec{X} = \vec{x}$ is an actual cause of φ.

Indeed, actual causality solves the problem of preemption.

→ Example 2.1.3, p. 11

Example 2.1.4. Consider again the scenario in Example 2.1.3. The ordering of coffee $\mathsf{CO} = 1$, that was no but-for cause, is an actual cause for the drink to be poured out: We choose the contingency $\mathsf{TP} = 0$, i.e., we fix the fact that in the actual world no tea is produced. Then, we have $(M, \vec{u}) \models [\mathsf{CO} \leftarrow 0, \mathsf{TP} \leftarrow 0]\ \neg\, \mathsf{CO} = 1$. Hence $(\{\mathsf{TP}\}, \mathsf{TP} = 0, \mathsf{CO} = 0\})$ is a witness for the fact that $\mathsf{CO} = 1$ is an actual cause of $\mathsf{D} = 1$. △

Note, that in the above example the larger set of events $\mathsf{CO} = 1 \wedge \mathsf{TO} = 1$ is a but-for cause. This shows, that actual causality analyzes causal connections more precisely than but-for causality, which can also be proven in general.

Proposition 1. *If $\vec{X} = \vec{x}$ is a (minimal) but-for cause of φ in $(M, \vec{u})$, then there is a subvector $\vec{Y} \subseteq \vec{X}$ and a subsetting $\vec{y} \subseteq \vec{x}$ such that $\vec{Y} = \vec{y}$ is an actual cause of φ in $(M, \vec{u})$. In particular, every singleton but-for cause is also an actual cause.*

Proof. Let $\vec{x}'$ be a witness for the (minimal) but-for cause $\vec{X} = \vec{x}$. Then, $(\emptyset, \emptyset, \vec{x}')$ is a witness for $\vec{X} = \vec{x}$ satisfying the **SAT**- and $\mathbf{CF}_{\mathsf{Act}}$-condition also in the definition of actual causality. If $\vec{X} = \vec{x}$ is not minimal, taking a minimal subvector $\vec{Y} \subseteq \vec{X}$ and subsetting $\vec{y} \subseteq \vec{x}$ that satisfies **SAT** and $\mathbf{CF}_{\mathsf{Act}}$ shows the first claim. The second claim simply follows, since the only strict subset of singleton events is $\emptyset$, that never fulfills both **SAT** and $\mathbf{CF}_{\mathsf{Act}}$. □

While we will not use causal models directly in the later definitions of counterfactual causality in real-time systems, we will argue that our definitions do widely agree with this foundational theory of causality in Section 3.3.

→ Section 3.3, p. 55

2.2 Real-Time Systems

Next, we introduce important definitions and results regarding real-time systems. As commonly done, we represent real-time systems by timed automata and orient ourselves at the underlying formalism of the model checker Uppaal [4, 17, 27]. This is mainly because our developed causality tool (c.f. Chapter 5) uses several functionalities and formats of Uppaal.

→ Chapter 5, p. 83

Timed automata use a set of real-valued clocks and so-called clock constraints over this set of clocks to model the real-time behavior. For a set of clocks C, a *clock constraint* is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ with $x, y \in C$, with $\sim \in \{<, \leqslant, =, \geqslant, >\}$, and with $n \in \mathbb{N}$. We denote the set of clock constraints over a clock set C as $\mathsf{CC}(C)$. In contrast to the usual definitions of timed automata, Uppaal allows not only to reset certain clocks when traversing a transition but to assign clocks also to non-zero values specified in terms of Uppaal's expression language. For our purpose, it suffices to define a *clock expression* over a set of clocks C to be a composition of assignments of the form $x := n$ for $x \in C$ and $n \in \mathbb{N}$ (i.e., an assignment of a clock to a fixed value) and denote the set of clock expressions as $\mathsf{CE}(C)$. We denote the "empty clock expression" (that is, the expression not changing any clock values) as ϵ.

Def. clock constraint

Def. clock expression

Definition 2.3 (Timed Automaton)

A *timed automaton* is a tuple $\mathsf{TA} = (Loc, l_0, C, Act, \rightarrow, I)$, where

Def. timed automaton

- Loc is a finite set of locations,
- $l_0 \in Loc$ is the initial location,
- C is a finite set of clocks,
- Act is a finite set of actions,
- $\rightarrow \subseteq (Loc \times \mathsf{CC}(C) \times Act \times \mathsf{CE}(C) \times Loc)$ is the finite transition relation, and
- $I : Loc \rightarrow \mathsf{CC}(C)$ is an invariant assignment.

For $(l, g, \alpha, E, l') \in \rightarrow$, we also write $l \xrightarrow{g:\alpha,E} l'$.

Def. labeled timed automaton

If there is furthermore a labeling function $L : Loc \rightarrow \mathcal{P}(AP)$ over a finite set of atomic propositions AP, we talk about a *labeled timed automaton*.

We introduce some syntactic sugar for timed automata:

- If there are multiple transitions only differing by their actions, we combine them to one transition annotated with a set of actions.
- For transitions $l \xrightarrow{g:\alpha,E} l'$ with $g = \top$, we might omit the guard and for transitions with $E = \epsilon$, we might omit the clock expression.
- We will assume all our timed automata to be labeled. If no explicit labeling function is given, we assume that the locations are labeled with their location names, i.e., $AP = Loc$ and $L(l) = \{l\}$.

All of this is also demonstrated in the following first example of a timed automaton.

Example 2.2.1. We model a coffee machine as a timed automaton depicted in Figure 2.1. To start producing coffee, the machine requires its button to be pushed and takes then 5 time units to produce the coffee. While coffee is produced, further button pushes have has no effect. Furthermore, the user of the machine can wait in front of the machine at any time.

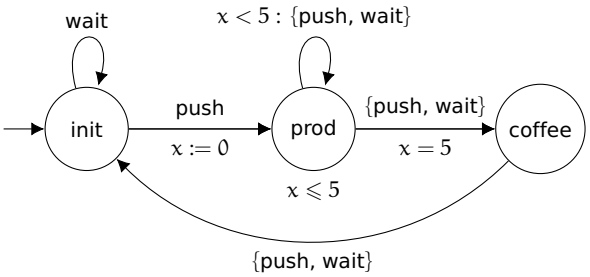


Figure 2.1: Timed automaton modeling a coffee machine

Invariants are annotated next to their locations (e.g., the invariant $x \leqslant 5$ below the location "prod"), while actions, guards, and clock expressions are arranged around the corresponding transition. △

Def. clock assignment

For an execution of a timed automaton, we use clock assignments to represent the current values of the clocks. Such a *clock assignment* over a set of clocks C is a function $u : C \rightarrow \mathbb{R}_{\geqslant 0}$, i.e., a function from the clock set to the non-negative reals. u_0 denotes the assignment where all clocks have value 0. We write $u \in g$, if the clock assignment u satisfies a clock constraint $g \in CC(C)$, write $u + \delta$ for the clock assignment that results out of u after $\delta \in \mathbb{R}_{\geqslant 0}$ time units have passed (that is, the clock assignment mapping all

$x \in C$ to $u(x) + \delta$), and write $E(u)$ for the clock assignment that results after applying the clock expression $E \in \mathsf{CE}(C)$ to u.

For a timed automaton $\mathsf{TA} = (Loc, l_0, C, Act, \to, I)$, we define the set of enabled transitions $Post(l, u, \alpha)$ of a given location $l \in Loc$ at a given clock assignment u for a given action α as

$$Post(l, u, \alpha) := \{\, l \xrightarrow{g:\alpha,E} l' \mid g \in \mathsf{CC}(C), E \in \mathsf{CE}(C), \text{ and } u \in g\}.$$

We say that the timed automaton is timewise *action-deterministic*, if $|Post(l, u, \alpha)| \leqslant 1$ for all $l \in Loc$, clock assignments u, and $\alpha \in Act$. **Def.** action-determinism

The operational semantics of timed automata is defined via a transition system with states $\langle l, u\rangle$ consisting of the current location l and the current clock assignment u. We shall also call the pair $\langle l, u\rangle$ a *configuration* of the timed automaton. We then allow two types of transition between states: *delay transitions*, where the automaton spends some time in its current location, and *action transitions*, where the automaton takes an enabled transition from the transition relation. **Def.** configuration **Def.** delay transition **Def.** action transition

Definition 2.4 (Semantics of Timed Automata)

Let $\mathsf{TA} = (Loc, l_0, C, Act, \to, I)$ be a timed automaton. The semantics is defined as a transition system whose states are pairs $\langle l, u\rangle$ of locations $l \in Loc$ and clock assignments u, whose initial state is $\langle l_0, u_0\rangle$, and whose transition relation is defined by

- $\langle l, u\rangle \xrightarrow{d} \langle l, u + \delta\rangle$, if $(u + d') \in I(l)$ for all $0 \leqslant \delta' \leqslant \delta$, $\delta \in \mathbb{R}_{\geqslant 0}$, (delay transition)
- $\langle l, u\rangle \xrightarrow{\alpha} \langle l', u'\rangle$, if $l \xrightarrow{g:\alpha,E} l'$ for some guard g and expression E such that $u \in g$, $u' = E(u)$, and $u' \in I(l')$. (action transition)

Even though this spans an infinite and even infinite branching transition system, the model-checking problem for certain logics like MITL (cf. Section 2.3) turn out to still be decidable. → Section 2.3, p. 20

For our discussion of causality in real-time systems, the traces and runs of timed automata will play a crucial role. We introduce three types of timed traces, all consisting out of timed actions.

Definition 2.5 (Runs, Timed Actions, and Timed Traces)

1. A *finite run* of a timed automaton is a sequence of executable delay and action transitions between the pairs of locations and clock assignments: **Def.** finite run
$$\rho = \langle l_0, u_0\rangle \xrightarrow{\delta_1}\xrightarrow{\alpha_1} \langle l_1, u_1\rangle \xrightarrow{\delta_2}\xrightarrow{\alpha_2} \ldots \xrightarrow{\delta_n}\xrightarrow{\alpha_n} \langle l_n, u_n\rangle.$$
2. An *infinite run* of a timed automaton is an infinite sequence of executable delay and action transitions between the pairs of locations and clock assignments: **Def.** infinite run
$$\rho = \langle l_0, u_0\rangle \xrightarrow{\delta_1}\xrightarrow{\alpha_1} \langle l_1, u_1\rangle \ldots \xrightarrow{\delta_i}\xrightarrow{\alpha_i} \langle l_i, u_i\rangle \ldots \qquad i = 2, 3, \ldots.$$

Def. lasso-shaped run

3. A *lasso-shaped run* of a timed automaton is an infinite run in lasso-form, i.e.,

$$\rho = \langle l_0, u_0 \rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \dots \xrightarrow{\delta_n} \xrightarrow{\alpha_n} \Big(\langle l_n, u_n \rangle \xrightarrow{\delta_{n+1}} \xrightarrow{\alpha_{n+1}} \dots \xrightarrow{\delta_{p-1}} \xrightarrow{\alpha_{p-1}} \langle l_{p-1}, u_{p-1} \rangle \xrightarrow{\delta_p} \xrightarrow{\alpha_p} \Big)^\omega.$$

Def. delay action

4. A *delay action* is a pair $\langle \delta, \alpha \rangle$, where $\alpha \in \mathrm{Act}$ is the action taken after a delay of $\delta \in \mathbb{R}_{\geqslant 0}$ since the last delay action (or the start of a timed automaton if it is the first delay action).

Def. timestamp action

5. A *timestamp action* is a pair $\langle t, \alpha \rangle$, where $\alpha \in \mathrm{Act}$ is the action taken at time-point $t \in \mathbb{R}_{\geqslant 0}$.

Def. finite delay trace

6. A finite *finite delay trace* is a sequence of delay actions $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle$.

Def. lasso-shaped delay trace

7. A *lasso-shaped delay trace* is a sequence of delay actions in lasso-form, i.e., $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle \Big(\langle \delta_{n+1}, \alpha_{n+1} \rangle \dots \langle \delta_p, \alpha_p \rangle \Big)^\omega$.

Def. timestamp trace

8. A finite *timestamp trace* is a sequence of timestamp actions $\xi = \langle t_1, \alpha_1 \rangle \dots \langle t_n, \alpha_n \rangle$ where $t_i \leqslant t_{i+1}$ for all $1 \leqslant i \leqslant n-1$.

Delay traces can be transformed into timestamp traces by setting $t_i := \sum_{j=1}^{i} d_j$ and vice versa timestamp traces into delay traces by setting $d_i := t_i - t_{i-1}$ (with $t_0 := 0$). Given a finite or lasso-shaped run ρ, the trace consisting of its delays/timestamps and actions is called *corresponding trace* of ρ. Conversely given a trace, we speak of *corresponding runs*. We say that a *trace is in a timed automaton* TA if there exists a corresponding run of TA and denote the set of all traces in TA as Traces(TA). Already by definition, a lasso-shaped run is an infinite run, however, there are also infinite runs that are not lasso-shaped (and hence, not necessarily computably treatable). If only the locations of a run are of interest we might omit the clock assignments and denote a run as $l_0 \xrightarrow{\delta_1} \xrightarrow{\alpha_1} l_1 \xrightarrow{\delta_2} \xrightarrow{\alpha_2} \dots \xrightarrow{\delta_n} \xrightarrow{\alpha_n} l_n$.

Def. corresponding trace

Def. corresponding runs

Def. trace in TA

Certain behaviors might occur in timed automata only after time is spent in a location without performing a further action. Hence, we might also consider finite runs ending with a delay and no further action written as $l_0 \xrightarrow{\delta_1} \xrightarrow{\alpha_1} l_1 \xrightarrow{\delta_2} \xrightarrow{\alpha_2} \dots l_{n-1} \xrightarrow{\delta_n}$ and denote its corresponding trace as $\langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, - \rangle$. Furthermore such a run might also stay infinitely long in its last location, we represent this by allowing $\delta_n = \infty$.

Def. extension

If a finite run ρ is a prefix of another run ρ', we call ρ' an *extension* of ρ. We thereby define ρ to be no prefix and, therefore, also no extension of itself, a run $l_0 \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \dots l_{n-1} \xrightarrow{\delta'_n}$ however to be a prefix and, therefore, an extension of $l_0 \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \dots l_{n-1} \xrightarrow{\delta_n}$ if $\delta' > \delta$.

Almost everywhere, it should become clear from the context whether we talk about finite delay, lasso-shaped delay, or timestamp traces; if there might be ambiguity, we will explicitly state which kind of timed trace is meant.

Example 2.2.2. For the timed automaton given in Example 2.2.1, there is, for example, the finite run → Example 2.2.1, p. 14

$$\langle \text{init}, x \mapsto 0\rangle \xrightarrow{4} \xrightarrow{\text{push}} \langle \text{prod}, x \mapsto 0\rangle \xrightarrow{2} \xrightarrow{\text{wait}} \langle \text{prod}, x \mapsto 2\rangle \xrightarrow{3} \xrightarrow{\text{wait}} \langle \text{coffee}, x \mapsto 5\rangle$$

with corresponding finite delay trace

$$\langle 4, \text{push}\rangle\langle 2, \text{wait}\rangle\langle 3, \text{push}\rangle.$$

△

For timewise action-deterministic timed automata, the corresponding runs of a given trace in TA are uniquely determined.

Proposition 2. *Let* TA *be a time-wise action-deterministic timed automaton. Then each trace in* TA *has a unique corresponding run of* TA*.*

Proof. The result is only proven for delay traces, the proofs for the other kinds of timed traces work analogously. We proceed by induction on the length n of the trace.

$n = 0$: For the empty trace, the empty run is the unique corresponding run of TA.

$n + 1$: Let $\xi = \langle \delta_1, \alpha_1\rangle \dots \langle \delta_n, \alpha_n\rangle\langle \delta_{n+1}, \alpha_{n+1}\rangle$ be a trace in TA. By induction, there exists a unique corresponding run $\rho' = \langle l_0, u_0\rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \dots \xrightarrow{\delta_n} \xrightarrow{\alpha_n} \langle l_n, u_n\rangle$ of $\langle \delta_1, \alpha_1\rangle \dots \langle \delta_n, \alpha_n\rangle$. Since ξ is a trace in TA and ρ' the unique run of its prefix, we have $\text{Post}(l_n, u_n + \delta_{n+1}, \alpha_{n+1}) \geqslant 1$ and since TA is timewise action-deterministic, we have also have $\text{Post}(l_n, u_n + \delta_{n+1}, \alpha_{n+1}) \leqslant 1$. Hence, there is a unique element $l_n \xrightarrow{g:\alpha,R} l_{n+1} \in \text{Post}(l_n, u_n + \delta_{n+1}, \alpha_{n+1})$ and

$$\rho' := \rho \xrightarrow{\delta_{n+1}} \xrightarrow{\alpha_{n+1}} \langle l_{n+1}, R(u_n + \delta_{n+1})\rangle$$

is the desired unique run corresponding run of ξ. □

This unique corresponding run is for a timewise action-deterministic automaton, in addition, easy to compute. This allows us to easily switch back and forth between traces and their corresponding run, which will in the following be done oftentimes implicitly without further justification. As a rule of thumb, we will talk about runs in contexts, where the locations or clock assignments are of interest, and refer to traces in contexts, where (only) the delays and actions matter.

We continue by defining some of the for our purpose important properties of runs of timed automata. We thereby denote with $\Delta = (\delta_i)_{i \in I}$ the sequence of all delays in a run. Depending on the type of the run, Δ might be a finite or an infinite sequence and I correspondingly a finite or an infinite index set. In particular for a lasso-shaped run, we define Δ as an infinite sequence of delays resulting from infinite unrolling of the lasso-part.

Definition 2.6 (Run Properties)

Let ρ be a run of a timed automaton TA and $\Delta = (\delta_i)_{i \in I}$ the (possibly infinite) sequence of its delays.

Def. execution time

1. The *execution time* $\mathsf{ExecTime}(\rho)$ of ρ is defined as $\sum_{\delta_i \in \Delta} := \delta_i$, whereby it might happen that $\mathsf{ExecTime}(\rho) = \infty$.

Def. time divergence

2. ρ is called *time-divergent*, if $\mathsf{ExecTime}(\rho) = \infty$, otherwise ρ is called time-convergent.

Def. zenoness

3. ρ is called *zeno*, if it is an infinite but time-convergent run. Otherwise ρ is non-zeno.

For a finite run ρ, we furthermore say that

Def. timelock

4. ρ is *timelocking*, if there is no run of TA that is an extension of ρ.

Def. deadlock

5. ρ is *deadlocking*, if there is no run of TA that is an extension of ρ with further actions.

We call a timed automaton non-zeno/timelock-free/deadlock-free, if all of its runs are non-zeno/not timelocking/not deadlocking.

We denote the set of all runs of a timed automaton TA as $\mathsf{Runs}(\mathsf{TA})$ and the set of all time-divergent runs as $\mathsf{Runs}_{\mathsf{div}}(\mathsf{TA})$.

Remark 2.2.1. Both time-locking as well as zeno runs are considered as modeling flaws that should be avoided. Just for reasons of simplification, most of the timed automata considered in the remainder of the presentations will, however, include in particular zeno runs. △

A further key operation for our definitions of counterfactual causality in real-time systems will be the intersection of timed automata. In contrast to for instance negation, timed automata are closed under intersection by constructing their product automaton. We define a slightly modified intersection operator, that given two labeled timed automata returns the labeled automaton, whose traces are exactly the intersection of the traces of both given automata while using as labels only the one the first automaton. The construction works as expected by basically building the product automaton and using the labeling function of the first automaton.

Definition 2.7 (Intersection Automaton)

Let $\mathsf{TA} = (\mathrm{Loc}, l_0, C, \mathrm{Act}, \rightarrow, I, AP, L)$ and $\mathsf{TA}' = (\mathrm{Loc}', l_0', C', \mathrm{Act}, \rightarrow', I', AP', L')$ be two labeled timed automata over the same set of actions Act and with disjoint clock sets C and C'. The intersection automaton of TA and TA' is defined as

$$\mathsf{TA} \cap \mathsf{TA}' := \big(\mathrm{Loc} \times \mathrm{Loc}', \langle l_0, l_0' \rangle, C \dot{\cup} C', \mathrm{Act}, \rightarrow \cap \rightarrow', I \cap I', AP, L_\pi\big),$$

whereby

- $\to \cap \to' := \left\{ \left(\langle l, l' \rangle, g \wedge g', \alpha, E; E', \langle r, r' \rangle \right) \mid (l, g, \alpha, E, r) \in \to,\ (l', g', \alpha, E', r') \in \to' \right\}$, where $E; E'$ denotes the composition of the expressions E and E',
- $I \cap I'(\langle l, l' \rangle) := I(l) \wedge I(l')$, and
- $L_\pi(\langle l, l' \rangle) := L(l)$.

Note that the assumption of disjoint clock sets can always be ensured by renaming and the assumption of the same set of actions by taking the union of the two sets of actions.

As desired, the intersection automaton fulfills the following property.

Proposition 3. *Let* TA *and* TA' *be two automata with disjoint sets of clocks* C *and* C'*. Then:*

1. *Let* $\rho = \langle l_0, u_0 \rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \ \dots \ \xrightarrow{\delta_i} \xrightarrow{\alpha_i} \langle l_i, u_i \rangle \dots$ *be a run of the automaton* TA *and* $\rho' = \langle l'_0, u'_0 \rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \ \dots \ \xrightarrow{\delta_i} \xrightarrow{\alpha_i} \langle l'_i, u'_i \rangle \dots$ *a run of the automaton* TA' *with identical corresponding trace* ξ*. Then the run*

$$\rho \cap \rho' := \langle \langle l_0, l'_0 \rangle, u_0 \cup u'_0 \rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \ \dots \ \xrightarrow{\delta_i} \xrightarrow{\alpha_i} \langle \langle l_i, l'_i \rangle, u_i \cup u'_i \rangle \ \dots$$

   is a run of $\mathsf{TA} \cap \mathsf{TA}'$ *with corresponding trace* ξ*, where* $u \cup u' : C \cup C' \to \mathbb{R}_{\geqslant 0}$ *denotes the union of the assignments* $u : C \to \mathbb{R}_{\geqslant 0}$ *and* $u' : C' \to \mathbb{R}_{\geqslant 0}$*.*

2. *Let* $\rho := \langle \langle l_0, l'_0 \rangle, u_0 \rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \ \dots \ \xrightarrow{\delta_i} \xrightarrow{\alpha_i} \langle \langle l_i, l'_i \rangle, u_i \rangle \dots$ *be a run of* $\mathsf{TA} \cap \mathsf{TA}'$ *with corresponding trace* ξ*. Then*

$$\rho_1 := \langle l_0, u_0|_C \rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \ \dots \ , \xrightarrow{\delta_i} \xrightarrow{\alpha_i} \langle l_i, u_i|_C \rangle \ \dots$$

   is a run of TA *and*

$$\rho_2 := \langle l'_0, u_0|'_C \rangle \xrightarrow{\delta_1} \xrightarrow{\alpha_1} \ \dots \ \xrightarrow{\delta_i} \xrightarrow{\alpha_i} \langle l'_i, u'_i|'_C \rangle \ \dots$$

   is a run of TA'*, both with corresponding trace* ξ*.*

Proof. Both claims follow directly by the definition of the intersection automaton. □

The result holds for any kind of runs (both finite and infinite runs). We will call the run $\rho \cap \rho'$ *intersection run*, the runs ρ_1 and ρ_2 *projective runs*. Regarding the traces of intersection automata we derive the following corollary.

Def. intersection run
Def. projective runs

Corollary 4. *Let* TA *and* TA' *be two automata with disjoint sets of clocks* C *and* C'*. Then, we have* $\mathsf{Traces}(\mathsf{TA} \cap \mathsf{TA}') = \mathsf{Traces}(\mathsf{TA}) \cap \mathsf{Traces}(\mathsf{TA}')$.

Proof. By applying Proposition 3 to the corresponding runs. □

→ Proposition 3, p. 19

While the intersection of timelock-free/deadlock-free automata is not necessarily timelock-free/deadlock-free, this corollary implies action-determinism and non-zenoness to transport to the intersection.

2.3 Real-Time Logics

In order to express specifications, properties, and particularly for our purpose effects in real-time systems, we conclude our preliminary discussions with some notes about real-time logics. For discrete-time logics, one roughly distinguishes between trace-based logics like *Linear Temporal Logic* (LTL) and so-called branching time logics like, for instance, *Computation Tree Logic* (CTL). Loosely speaking, trace-based logics focus on the sequence of labels (or sometimes outputs) generated during a run of the system[3]. A similar distinction can also be retained for the respective real-time counterparts *Metric Temporal Logic* (MTL) of LTL and *Timed Computation Tree Logic* (TCTL) of CTL. For our application, the family of branching time logics is not suitable as we will discuss in more depth in Section 3.3, such that we will focus on MTL or more precisely on its decidable fragment *Metric Interval Temporal Logic* (MITL). These real-time logics are based on continuous-time and label-valued signals.

→ Section 3.3, p. 55

Definition 2.8 (Signal)

Def. signal

A *signal* over a set of atomic propositions AP is a function $M : \mathbb{R}_{\geqslant 0} \to \mathcal{P}(AP)$.

Def. corresponding signal

Given a labeled timed automaton $\mathsf{TA} = (Loc, l_0, C, Act, \to, I, AP, L)$, the *corresponding signal* M of a time-divergent run ρ of TA is defined as $M(t) := L(l_\rho(t))$, where $l_\rho(t)$ denotes the current location of ρ at absolute time t.

The run ρ is required to be time-divergence since we can otherwise not define a total corresponding signal $M : \mathbb{R}_{\geqslant 0} \to \mathcal{P}(AP)$ for ρ but only up to $\mathsf{ExecTime}(\rho)$. Note furthermore, that for absolute times t at which ρ performs an action taking the run from one location to another location, its current location $l_\rho(t)$ is not uniquely determined. For this cases, we define $M(t)$ as the union of the labels of the multiple current locations.

Besides atomic propositions and the basic Boolean connectives, temporal logics include temporal operators specifying temporal behavior. MTL lifts the until operator $\mathcal{U}$ – also known, for example, from LTL – to real time. The operator is now parameterized, in addition, with a time interval $I \subseteq \mathbb{R}_{\geqslant 0}$ and the formula $\phi\, \mathcal{U}_I\, \psi$ states that after some period of time included in I, we have that ψ holds and until then ϕ holds. In its full generality, MTL is, however, not decidable. Therefore, we will work as already mentioned above with its decidable fragment MITL restricting MTL to non-singleton and naturals-bounded intervals I. Formally, the syntax and semantics of MITL are defined as follows.

Definition 2.9 (MITL)

Def. MITL syntax

The *syntax* of MITL formulas over a set of atomic propositions AP is defined by

$$\phi := p \mid \neg\phi \mid \phi \wedge \phi \mid \phi\, \mathcal{U}_I\, \phi,$$

[3] Here, the use of the term "trace-based" may lead to some confusion: In discrete settings, one commonly calls the sequence of labels trace, while for real-time systems, traces describe the sequence of delays and actions (cf. Def. 2.5).

where $p \in AP$ and I is a non-singleton interval of the form $[a, b]$, $(a, b]$, $[a, b)$, (a, b), (a, ∞), or $[a, \infty)$ with $a, b \in \mathbb{N}$ and $a < b$.

The *semantics* is defined with respect to a signal $M : \mathbb{R}_{\geqslant 0} \to \mathcal{P}(AP)$ and a point of time $t \in \mathbb{R}_{\geqslant 0}$ inductively by

Def. MITL semantics

$$
\begin{array}{lll}
M, t \models p & \text{iff} & p \in M(t) \\
M, t \models \neg\phi & \text{iff} & M, t \not\models \phi \\
M, t \models \phi \wedge \psi & \text{iff} & M, t \models \phi \text{ and } M, t \models \psi \\
M, t \models \phi \,\mathcal{U}_I\, \psi & \text{iff} & \exists t' > t.\ t' - t \in I,\ M, t' \models \psi \text{ and } \forall t'' \in (t, t').\, M, t'' \models \phi
\end{array}
$$

We say that a signal M *satisfies* an MITL formula ϕ, if $M, 0 \models \phi$. Furthermore, a time-divergent run ρ *satisfies* a formula ϕ, if its corresponding signal does so and that a timed automaton TA *satisfies* ϕ, if all its time-divergent runs satisfy ϕ. In all those cases, we shall write $M \models \phi$, $\rho \models \phi$, and $\mathsf{TA} \models \phi$ respectively.

Def. formula satisfaction

Further Boolean connectives are defined in the classical way. We introduce some abbreviations to get further common temporal operators:

- $\Diamond_I \phi := \top \,\mathcal{U}_I\, \phi$ (expresses that ϕ holds eventually in the interval I)
- $\Box_I \phi := \neg\Diamond_I \neg\phi$ (expresses that ϕ holds always in the interval I)
- $\phi \,\mathcal{U}\, \psi := \phi \,\mathcal{U}_{[0,\infty)}\, \psi$, $\Diamond \phi := \Diamond_{[0,\infty)} \phi$, and $\Box \phi := \Box_{[0,\infty)} \phi$

MITL is defined as a logic that talks about continuous signals with infinite domain which are as mentioned in turn only defined for time-divergent runs. For finite runs, we use the concept of good and bad prefixes enabling us also to specify finite runs using MITL.

Definition 2.10 (Good Prefixes)

Let ρ be a finite run and ϕ an MITL formula. We call ρ a *good prefix* of ϕ, if all time-divergent extensions ρ' of ρ satisfy ϕ.

Def. good prefix

We overload the notation and shall also write $\rho \models \phi$, if ρ is a good prefix of ϕ, and write $\rho \not\models \phi$, if ρ is no good prefix of ϕ.

We conclude our preliminary discussion by stating the result that MITL model checking is decidable by computational manners, i.e., we can for a timed automaton algorithmically decide whether all its time divergent runs satisfy an MITL formula [9]. We will use this model checker in our work as a "black box" and do not focus on the details of this model-checking process. Consequently, we do also not present a proof of the result but refer to the literature.

Theorem 5 (MITL Model Checking)**.** *The decision problem "Given a timed automaton* TA *and an MITL formula* ϕ*, do we have* $\mathsf{TA} \models \phi$*?" is decidable by computation.*

Proof. See the proof by Alur et al. [9]. □

We abbreviate the MITL model-checking problem from the above theorem as $\mathrm{MC}_{\mathrm{MITL}}$. Relevant for the complexity analysis of our notions, Alur et al. classify the problem as well in terms of the computational effort it takes to solve it.

Theorem 6. MC_{MITL} *is* EXPSPACE*-complete.*

Proof. See again [9]. □

Thereby EXPSPACE denotes the complexity class of decision problems that are deterministically solvable in exponential space. It is a well-known result in complexity theory, that the class of decision problems that are deterministically solvable in exponential time EXPTIME is a subclass of EXPSPACE, i.e., $\mathsf{EXPTIME} \subseteq \mathsf{EXPSPACE}$.

Chapter 3

Formal Definitions of Counterfactual Causality in Real-Time Systems

We aim to define a formal notion of counterfactual causality in real-time systems in the flavor of Halpern and Pearl's but-for and actual causality as presented in Section 2.1, to which we will refer as the "foundational notions" in the "foundational setting". We take an approach that looks at events and effects on timed traces. It turns out that the delay perspective on traces is well-suited to fully adapt the concepts of counterfactual causality to real-time systems. This development is presented in Section 3.1. We discuss and compare a different approach in Section 3.2 that takes a timestamp perspective on traces in timed automata. For both perspectives, we give a number of examples that showcase the usefulness and the conformity with the intuition of our notions but also demonstrate the differences between both approaches. Finally, we add various further remarks in Section 3.3: We comment on the assumptions of our setting in more detail, report limitations of the developed notions, and discuss further theoretical aspects of our causality formalism. We recall that all presented examples are available under

→ Section 2.1, p. 7
→ Section 3.1, p. 23
→ Section 3.2, p. 46
→ Section 3.3, p. 55

https://github.com/FelixJahnFJ/Real-Time-Causality-Tool/tree/main/Thesis_Examples.

for own tests and experiments using the developed causality tool.

3.1 Delay Causality in Real-Time Systems

3.1.1 Causal Setting: Events and Effects in Real-Time Systems

To get to a notion of causality in real-time systems, we first have to discuss and choose carefully, in which causal setting we want to work. In all our approaches, we will

start by considering a given timed automaton together with a run of this automaton and base the counterfactual reasoning then on its corresponding trace. For now, recall
→ Definition 2.5, p. 15
Definition 2.5 of a finite delay trace

$$\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle \quad \text{consisting of timed actions} \quad \langle \delta_i, \alpha_i \rangle, i = 1, \dots, n,$$

and of a finite run

$$\rho = l_0 \xrightarrow{\delta_1} \xrightarrow{\alpha_1} l_1 \xrightarrow{\delta_2} \xrightarrow{\alpha_2} \dots \xrightarrow{\delta_n} \xrightarrow{\alpha_n} l_n.$$

Halpern and Pearl define causes as sets of events that lead to an effect. Therefore, it is also in our context of real-time systems a crucial – and in various ways not canonical – choice, what we want to understand as events and effects, and, hence, what we consider as possible causes. Take a look at the following (synthetic) example.

Example 3.1.1. We consider the timed automaton TA in Figure 3.1 over the action alphabet $\{\alpha, \beta\}$:

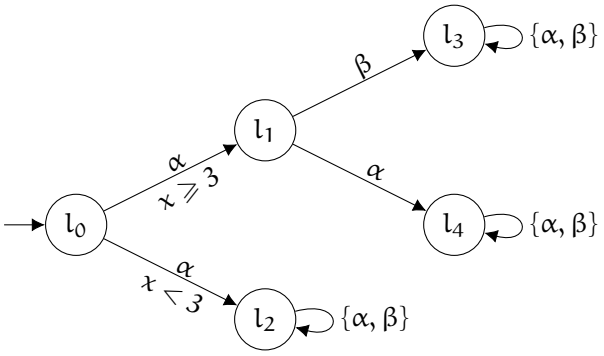


Figure 3.1: Timed Automaton TA

The run $l_0 \xrightarrow{3} \xrightarrow{\alpha} l_1 \xrightarrow{2} \xrightarrow{\alpha} l_4$ reaches location l_4. Both with a shorter delay in l_0 or with taking action β instead of α in l_1, the run would not have reached l_4. △

The above example illustrates, that in real-time systems both delays as well as actions seem to "cause" certain effects – in this case the effect, that location l_4 is reached. Hence, we want to consider at every point of the corresponding trace both delays and actions as events.

Definition 3.1 (Events – Delay Perspective)

Def. delay and action event

1. The set of *delay events* is defined as $\mathcal{DE} := \{(\delta, i) \in \mathbb{R}_{\geqslant 0} \times \mathbb{N}\}$, the set of *action events* over a set of actions Act as $\mathcal{AE} := \{(\alpha, i) \in \text{Act} \times \mathbb{N}\}$.

Def. set of events

2. The whole *set of events* $\mathcal{E}$ is then obtained by the disjoint union[1] of $\mathcal{DE}$ and $\mathcal{AE}$, i.e., $\mathcal{E} := \mathcal{DE} \dot\cup \mathcal{AE}$.

[1] To avoid confusion and ensure disjointeness of delay and action events, we assume that no $\alpha \in \text{Act}$ is a real value.

The integer values in the second component of the event pairs specify the index of the event in the trace, which allows to refer to events at different positions.

In line with the foundational definitions of counterfactual causality, we include not only the actual events, i.e., the delays and actions on the given trace in the set of events but define the set of events independently of a concrete trace. Later, sets of events will then be required to be satisfied by the given trace in order to qualify as potential cause. A trace satisfies a set of events if all delay and all action events indeed occur at the particular positions in the trace.

Definition 3.2 (Event Satisfaction – Delay Perspective)

We say that a finite delay trace $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle$ or a lasso-shaped delay trace $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle \big(\langle \delta_{n+1}, \alpha_{n+1} \rangle \dots \langle \delta_p, \alpha_p \rangle \big)^\omega$ *satisfies a set of events* $\mathcal{C} \subseteq \mathcal{E}$, if

Def. event satisfaction

- for every delay event $(\delta, i) \in \mathcal{C}$, we have $\delta = \delta_i$, and
- for every action event $(\alpha, i) \in \mathcal{C}$, we have $\alpha = \alpha_i$.

We denote this by $\xi \models \mathcal{C}$.

Given a trace $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle$, we also introduce a notation for the set of all events of this trace and denote

$$\mathcal{C}_\xi := \{(\delta_i, i) \mid i = 1, \dots, n\} \cup \{(\alpha_i, i) \mid i = 1, \dots, n\}.$$

The notation is adapted as expected also for lasso-shaped delay traces. It is easy to see that we have $\xi \models \mathcal{C}_\xi$ for every trace ξ.

Recall again, that the key principle in counterfactual reasoning is to simulate a counterfactual world, that is, the world that would have been if the potential cause had not been. In our case, we want to simulate how the timed automaton would have run if the potentially causal delay and action events had not happened. For this counterfactual simulation, we use in a first step a so-called counterfactual trace automaton, an idea adapted from the work of Coenen et al. [22]. Counterfactual trace automata represent exactly all the possible counterfactual traces, that is, those traces that allow exactly for the causal events alternatives to happen while all non-causal events remain fixed. To do so for a given trace

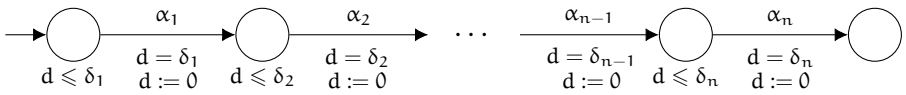
$$\xi = \langle \delta_1, \alpha_1 \rangle \langle \delta_2, \alpha_2 \rangle \dots \langle \delta_{n-1}, \alpha_{n-1} \rangle \langle \delta_n, \alpha_n \rangle$$

and a set of events like this demonstration example set

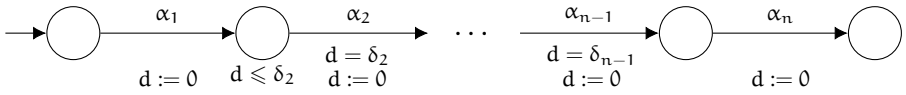
$$\mathcal{C} = \{(\delta_1, 1), (\delta_n, n), (\alpha_2, 2), (\alpha_n, n)\}$$

we proceed in the following way:

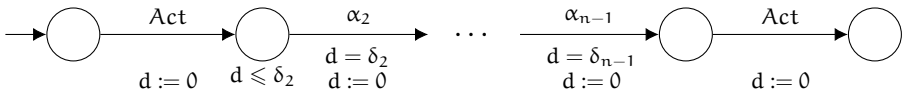
1. We enforce all delays and actions to be exactly as in the given actual trace using a clock d that measures the delays:



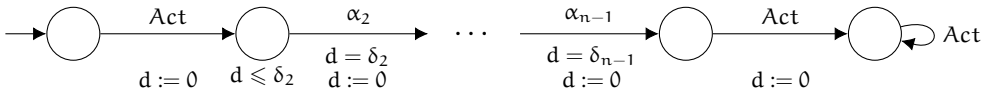
2. We relax the delay constraints at the points for which there is a delay event in $\mathcal{C}$ by omitting the corresponding guards and invariants:



3. We relax the action constraints at the points for which there is an action event in $\mathcal{C}$ by allowing all actions from Act to be taken:



4. We allow arbitrary continuations in the counterfactual simulation by adding a self-loop without any delay or action constraints to the last location:

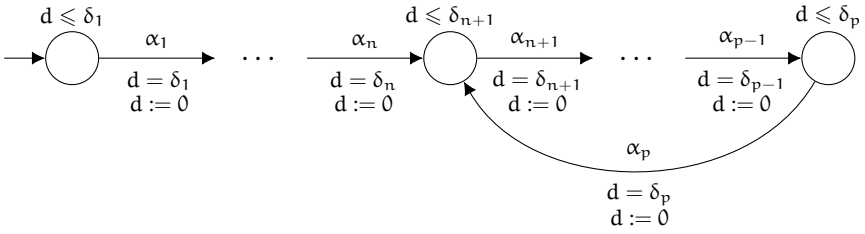


By this construction, we indeed enforce for traces in the counterfactual trace automaton exactly the non-causal events to be as they were in ξ, but allow alternatives for the causal events of ξ, that is, arbitrary delays at the points of causal delay events and arbitrary actions at the points of causal action events.

For the case of a trace ending with a delay, i.e., $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, - \rangle$, we proceed as if the non-existent action at the last position were a causal action event and allow all actions in the n-th transition. For the case of a trace ending with an "infinite delay", i.e., $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \infty, - \rangle$, we allow further actions only if the infinite delay is a causal one. For a lasso-shaped delay trace

$$\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle \Big(\langle \delta_{n+1}, \alpha_{n+1} \rangle \dots \langle \delta_p, \alpha_p \rangle \Big)^\omega$$

we adapt the above construction by mirroring the lasso-shape in the structure of the counterfactual trace automaton by starting in Step 1 with the following automaton:



That is, the automaton in which the transition belonging to the last delay action of the lasso-part leads to the location whose outgoing transition belongs to the first delay action of the lasso-part. Except for Step 4, which is omitted, we proceed for the remaining steps completely analogously.

This concept of a counterfactual trace automaton is formalized as follows.

Definition 3.3 (Counterfactual Trace Automaton – Delay Perspective)

Let $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle$ be a finite delay trace over a finite set of actions Act and let $\mathcal{C} \subseteq \mathcal{E}$ be a finite set of events. The *counterfactual trace automaton* of trace ξ for the set of events $\mathcal{C}$ is defined as $\mathrm{TA}_\xi^{\mathcal{C}} := (\mathrm{Loc}, l_0, C, \mathrm{Act}, \to, I)$ with

Def. counterfactual trace automaton

- $\mathrm{Loc} := \{0, \dots, n\}$
- $l_0 := 0$
- $C := \{d\}$
- the transition relation $\to$ is defined by the following rules:

$$\frac{(\delta_i, i) \notin \mathcal{C} \qquad (\alpha_i, i) \notin \mathcal{C}}{i-1 \xrightarrow{d=\delta_i:\, \alpha_i,\, d:=0} i} \qquad \frac{(\delta_i, i) \in \mathcal{C} \qquad (\alpha_i, i) \notin \mathcal{C}}{i-1 \xrightarrow{\top:\, \alpha_i,\, d:=0} i}$$

$$\frac{(\delta_i, i) \notin \mathcal{C} \qquad (\alpha_i, i) \in \mathcal{C} \qquad \beta \in \mathrm{Act}}{i-1 \xrightarrow{d=\delta_i:\, \beta,\, d:=0} i} \qquad \frac{(\delta_i, i) \in \mathcal{C} \qquad (\alpha_i, i) \in \mathcal{C} \qquad \beta \in \mathrm{Act}}{i-1 \xrightarrow{\top:\, \beta,\, d:=0} i}$$

$$\frac{\beta \in \mathrm{Act}}{n \xrightarrow{\beta} n}$$

- $I(i) := \begin{cases} d \leqslant \delta_{i+1}, & (\delta_{i+1}, i+1) \notin \mathcal{C} \\ \top, & \text{otherwise.} \end{cases}$

For a lasso-shaped delay trace $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle \big(\langle \delta_{n+1}, \alpha_{n+1} \rangle \dots \langle \delta_p, \alpha_p \rangle \big)^\omega$ the construction of the counterfactual trace automaton works as above but we use as set of locations $\mathrm{Loc} := \{0, \dots, n, n+1, \dots, p-1\}$, identify in the definition of the transition relation $\rightarrow$ index p with n, and omit the last defining rule of $\rightarrow$.

The first four defining rules of the transition relation cover exactly the four possible combinations

- there is *no causal delay event* and *no causal action event* at position i,
- there is a *causal delay event* but *no causal action event* at position i,
- there is *no causal delay event* but a *causal action event* at position i, or
- there is a *causal delay event* and a *causal action event* at position i.

The last rule creates the self-loop in the last state as sketched in Step 4 of the construction.

We remark two details regarding the invariants in the counterfactual trace automaton:

Remark 3.1.1. 1. Notice that the well-definedness of the invariant function I in the above definition is not that clear. In fact, the definition of I is ambiguous in cases where we have multiple causal delay events at a position i, i.e., if we have $(\delta, i) \in \mathcal{C}$ and $(\tilde{\delta}, i) \in \mathcal{C}$ with $\delta \neq \tilde{\delta}$. Such sets of events will, however, never be satisfiable by a trace, such that those sets already disqualify themselves beforehand as a cause.

2. Later, we will require the counterfactual run only to be time-divergent. The purpose of adding the invariants in the locations with non-causal delay events is, therefore, to exclude counterfactual runs that remain forever in locations with actually fixed delay times. On the other hand, it will indeed be possible for counterfactual runs to remain infinitely long in a location in which we have a causal delay event (as the invariant is omitted in those locations and such a run would be time-divergent). △

We give multiple examples for the construction of counterfactual trace automata in the upcoming subsections where we show how the concept is then finally used to define counterfactual causality in real-time systems.

For now, we will content with a property about the traces of the counterfactual trace automata for two particular kinds of event sets. The result basically states that in the counterfactual trace automaton for the empty set of events the only possible trace is exactly ξ, while in the counterfactual trace automaton for the set of all events $\mathcal{C}_\xi$ all traces over the given set of actions are possible.

Lemma 7. *Let ξ be a finite or lasso-shaped delay trace over a set of actions* Act.

1. *All time-divergent traces of $\mathsf{TA}_\xi^\emptyset$ agree with ξ in all their indices up to the length of ξ. In particular if ξ is time-divergent, ξ is the only time-divergent trace of $\mathsf{TA}_\xi^\emptyset$.*

2. *The time-divergent traces of $\mathsf{TA}_\xi^{\mathcal{C}_\xi}$ are all constructible time-divergent traces over* Act.

Proof. 1. Follows since all transitions of $\mathsf{TA}_\xi^\emptyset$ have (for finite ξ up to the last location) the form $i-1 \xrightarrow{d=\delta_i:\, \alpha_i,\ d:=0} i$, whereby $\langle \delta_i, \alpha_i \rangle$ are the timed actions of ξ.

2. Follows since all transitions of $\mathsf{TA}_\xi^{\mathcal{C}_\xi}$ have (for finite ξ up to the last location) now the form $i-1 \xrightarrow{\top:\, Act,\ d:=0} i$. □

The final aspect of the causal setting that remains to be discussed are effects. We specify effects in runs of timed automata by using the language of MITL:

Definition 3.4 (Run Effect)

A *run effect* is an MITL formula ϕ. We shall say that an effect ϕ *appears* in a run ρ, if we have $\rho \models \phi$, otherwise we say that the effect does *not appear*.

Def. run effect

Recall our remarks regarding MITL in Section 2.3: The semantics of MITL is defined with respect to time-divergent runs of timed automata by considering its corresponding signal. A time-divergent run ρ does either satisfy an MITL formula ϕ, written as $\rho \models \phi$, or does not satisfy the formula, written as $\rho \not\models \phi$. For our purposes also the notion of good prefixes (cf. Definition 2.10) will be important as we will oftentimes also consider finite runs in which certain effects appear. Recall in this regard in particular that MITL allows us to express properties of runs like reachability, which will be for simplicity the effect considered regularly in the upcoming examples.

→ Section 2.3, p. 20

→ Definition 2.10, p. 21

3.1.2 But-For Causality in Real-Time Systems

We are set to define the first notion of counterfactual causality in real-time systems. Therefore, recall again the crucial idea of counterfactual reasoning: We assume for the causal events alternatives to occur and consider then the counterfactual simulation of the world resulting from the alternative events. In the foundational setting, we simulate this counterfactual world based on given structural equations. By respecting those equations, we ensure that the simulation follows the rules of the causal model, i.e., the rules describing the behavior of the world.

Now given a run and a set of events, the counterfactual trace automaton of the corresponding trace represents as discussed above all the possible alternative traces. However, we do not yet ensure that those alternative runs indeed also respect the given rules of the world, which are in our setting specified by the given timed automaton. In fact, there might be lots of traces in the counterfactual trace automaton that are no traces

at all in the given automaton and have, therefore, no corresponding runs that should be considered as valid counterfactual simulations. Hence, we require for a counterfactual run that its corresponding trace is not only a trace in the counterfactual trace automaton but that it is also a trace in the given timed automaton. Technically, this is realized by using the intersection operator $\cap$ from Definition 2.7.

→ Definition 2.7, p. 18

Apart from this, the definition of but-for causes is a straightforward transfer of the foundational one (cf. Definition 2.1) to the real-time setting.

→ Definition 2.1, p. 10

Definition 3.5 (But-For Causality in Real-Time Systems – Delay Perspective)

Def. but-for cause

Let TA be a timed automaton, ρ a run of TA with corresponding delay trace ξ, and ϕ a run effect. A set of events $\mathcal{C} \subseteq \mathcal{E}$ is a *but-for cause* for ϕ in ρ of TA, if the following two conditions hold:

SAT $\xi \models \mathcal{C}$ and $\rho \models \phi$.

CF$_{\mathsf{BF}}$ There is a time-divergent run ρ' of $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ with $\rho' \not\models \phi$.

Def. minimal but-for cause

We call the cause *minimal but-for cause* if furthermore the following condition holds:

MIN $\mathcal{C}$ is minimal, i.e., no strict subset of $\mathcal{C}$ satisfies **SAT** and **CF$_{\mathsf{BF}}$**.

Def. witnessing run

We again call the counterfactual run ρ' in the **CF$_{\mathsf{BF}}$**-condition *witnessing run* or *witness* for the fact that $\mathcal{C}$ is a but-for cause for ϕ in ρ of TA.

Notice a small detail in the above definition: The effect ϕ refers in the **SAT**-condition to a run of the given automaton TA while in the **CF$_{\mathsf{BF}}$**-condition ϕ refers then to a run of the intersection automaton $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}}$. Exactly to handle this lifting of run effects we defined the labels of the locations of intersection automata to be only the label of the first component. Also conversely, run effects (not) appearing in runs in the intersection transport to the projective runs. Those transport properties are crucial to show first results regarding the notion of causality in real-time systems.

Proposition 8. *$\emptyset$ is never a but-for cause for a run effect ϕ in a run ρ of a timewise action-deterministic timed automaton* TA.

Proof. Let ρ be a run of a timewise action-deterministic timed automaton TA. We distinguish two cases:

$\rho \models \phi$: We show that the **CF$_{\mathsf{BF}}$**-condition is violated: Towards a contradiction, assume that there is a time-divergent run ρ' of $\mathsf{TA} \cap \mathsf{TA}_\xi^{\emptyset}$ with $\rho' \not\models \phi$, whereby ξ' is the corresponding trace of ρ'. By Lemma 7 we do however know that ξ' agrees with ξ up to the length of ξ. Hence the trace of the projection run ρ'_1 also agrees with ξ up to the length of ξ. Now Proposition 2 implies that the run ρ'_1 as a run of the timewise action-deterministic automaton TA agrees with the run ρ up to the

→ Lemma 7, p. 29

→ Proposition 2, p. 17

length of ρ. Now either if ρ is time-divergent we have $\rho = \rho'_1$, a contradiction since we have both $\rho \models \phi$ as well as $\rho'_1 \not\models \phi$ (since $\rho' \not\models \phi$), or if ρ is not time-divergent, ρ cannot be a good prefix of ϕ since there is the extension ρ'_1 with $\rho'_1 \not\models \phi$, again a contradiction.

$\rho \not\models \phi$: Now, the **SAT**-condition is violated already by definition and, hence, $\emptyset$ is again no cause for ϕ on ρ of TA. □

While this shows that an effect is never caused by "nothing", i.e., never by the empty set of events, we can on the other hand also show that in the case that the effect can be avoided in the timed automaton at all, there is indeed also a but-for cause in the sense of our definition for the effect.

Proposition 9. *Let ϕ be a run effect and TA a timed automaton. If there exists a time-divergent run ρ' of TA with $\rho' \not\models \phi$, then for every run ρ in which ϕ appears there is a (minimal) but-for cause for ϕ in ρ of TA.*

Proof. Let ρ' be a time-divergent run of TA with $\rho' \not\models \phi$ and ρ be a run of TA with $\rho \models \phi$. The set of events $\mathcal{C}_\xi$, i.e., the set of all events of the corresponding trace ξ of ρ, fulfills the **SAT**- as well as the $\mathbf{CF}_{\mathsf{BF}}$-condition:

By assumption, we have $\rho \models \phi$ and by construction of $\mathcal{C}_\xi$ we have $\xi \models \mathcal{C}_\xi$ such that the **SAT**-condition is fulfilled. By Lemma 7, we find the corresponding trace of ρ' also in $\mathsf{TA}_\xi^{\mathcal{C}_\xi}$. Hence, we find a correspondence to ρ' also in $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}_\xi}$ in which ϕ does not appear such that $\mathbf{CF}_{\mathsf{BF}}$ is fulfilled.

→ Lemma 7, p. 29

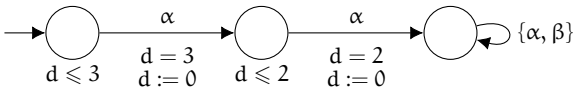
Therefore $\mathcal{C}_\xi$ is a but-for cause and a minimal subset of $\mathcal{C}_\xi$ that fulfills **SAT** and $\mathbf{CF}_{\mathsf{BF}}$ is a minimal but-for cause for ϕ in ρ of TA. □

The previous two results establish some basic properties we expect from a causality notion. In the following, we continue to argue for the meaningfulness of the notion by giving elementary examples that demonstrate the application of the above constructions and definitions, that illustrate how to validate whether a set of events fulfills the criteria of but-for causality, and that show that the notion is in many scenarios in accordance to what we would intuitively identify as a cause for certain events. We start with our entry example from the beginning of the chapter.

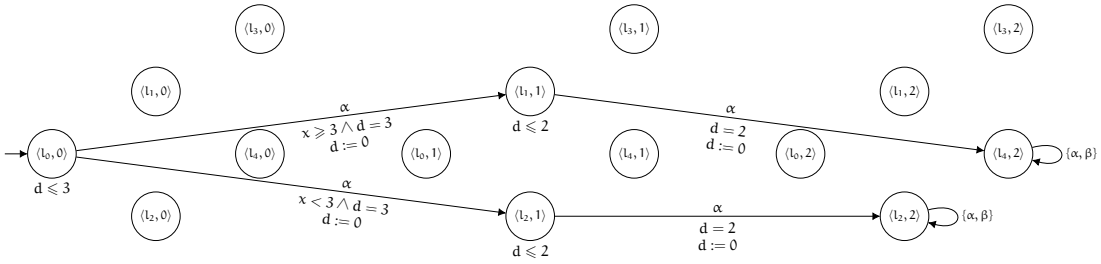
Example 3.1.2. Recall Example 3.1.1, where we considered the timed automaton TA in Figure 3.1 and its run $\rho := l_0 \xrightarrow{3} \xrightarrow{\alpha} l_1 \xrightarrow{2} \xrightarrow{\alpha} l_4$ with its effect that location l_4 is reached. We express the effect formally in MITL as $\phi := \Diamond\, l_4$. In line with our intuition, both $\mathcal{C}_1 := \{(3,1)\}$ as well as $\mathcal{C}_2 := \{(\alpha,2)\}$ are (minimal) but-for causes for ϕ in ρ of TA:

→ Figure 3.1, p. 24

Let ξ be the corresponding trace of ρ. We have both $\xi \models \mathcal{C}_1$ and $\xi \models \mathcal{C}_2$ as well as $\rho \models \phi$ such that the **SAT**-condition is fulfilled. For the $\mathbf{CF}_{\mathsf{BF}}$-condition, we start by constructing the counterfactual trace automata. For demonstration purposes, we first look at the one for the empty set of events $\mathsf{TA}_\xi^\emptyset$:

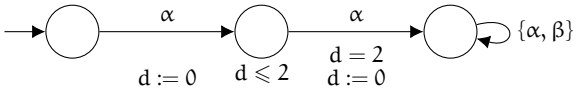


The counterfactual trace automaton is then used to construct the intersection $\mathsf{TA} \cap \mathsf{TA}_\xi^\emptyset$. We will generally depict all the locations of this product construction but only the transitions and invariants of interest:

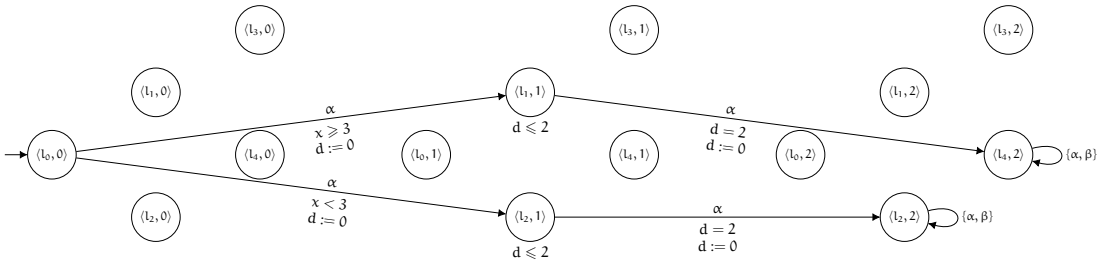


Due to the enforcement of the first delay, the transition from $\langle l_0, 0\rangle$ to $\langle l_2, 1\rangle$ cannot be taken. Hence, all runs in the intersection automaton lead again to location l_4 such that we cannot find a counterfactual run in which the effect ϕ does not appear.

This changes when turning to non-empty sets of events: For $\mathcal{C}_1 = \{(3, 1)\}$, the counterfactual trace automaton $\mathsf{TA}_\xi^{\mathcal{C}_1}$ looks as follows:

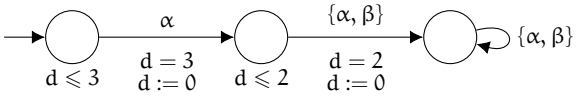


This then results in the following intersection automaton $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}_1}$:

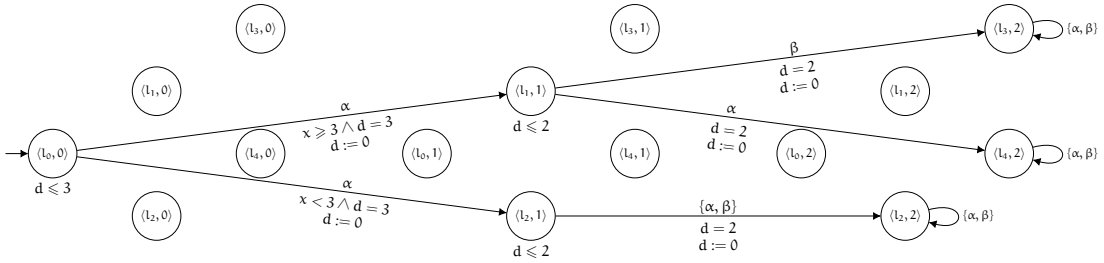


Since there is now no delay enforced in the first transition, we find, for example, the time-divergent counterfactual run $\rho' := \langle l_0, 0\rangle \xrightarrow{1}\xrightarrow{\alpha} \langle l_2, 1\rangle \xrightarrow{2}\xrightarrow{\alpha} \left(\langle l_2, 2\rangle \xrightarrow{1}\xrightarrow{\alpha}\right)^\omega$ with $\rho' \not\models \Diamond\, l_4$, i.e., in which ϕ does not appear. Hence, ρ' is a witness for $\mathcal{C}_1$ being a but-for cause for ϕ in ρ of TA.

For $\mathcal{C}_2 = \{(\alpha, 2)\}$, the counterfactual trace automaton $\mathsf{TA}_\xi^{\mathcal{C}_2}$ now allows the alternative action β for the second transition:



Therefore, also the intersection automaton $\mathsf{TA} \cap \mathsf{TA}_{\xi}^{\mathcal{C}_2}$ now allows an additional transition:



Again, we find with $\rho' := \langle l_0, 0\rangle \xrightarrow{3} \xrightarrow{\alpha} \langle l_1, 1\rangle \xrightarrow{2} \xrightarrow{\beta} \left(\langle l_3, 2\rangle \xrightarrow{1} \xrightarrow{\alpha}\right)^{\omega}$ a time-divergent counterfactual run for which we have $\rho' \not\models \Diamond\, l_4$ and which is, therefore, a witness for

The **MIN**-condition is, due to Proposition 8, always satisfied for singleton sets. △ → Proposition 8, p. 30

Notice that in the above examples we have in fact also the finite run $\rho' := \langle l_0, 0\rangle \xrightarrow{\infty}$ as a time-divergent run not reaching location l_4 and, therefore, as a counterfactual run witnessing $\mathcal{C}_1$ to be a but-for cause. This phenomenon of finite counterfactual runs is not limited to this example, but can be observed in general.

Remark 3.1.2. Recall Remark 3.1.1 in which we already mentioned that the counterfactual trace automaton includes runs staying infinitely long in locations with causal delay events. As those runs are time-divergent, they might qualify as finite counterfactual runs (we do not require counterfactual runs to be infinite). This, in turn, means that all delay events in locations without a bounding invariant will for many effects be identified as singleton causes. Allowing to stay infinitely long in a location as an alternative for a causal finite delay and thereby allowing in fact also all following delay and action events – no matter whether causal or non-causal – to not appear at all seems at first glance a bit odd. However, we would argue that it makes perfect sense to say that if the automaton could have stayed forever in a certain location and thereby avoid the effect, then the delay event causing the automaton to leave the location should count as a cause for the subsequent system behavior. → Remark 3.1.1, p. 28

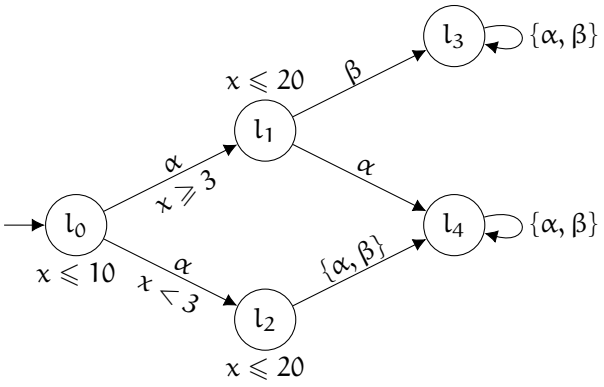
This has, however, implications for our following work as the causality analysis is significantly affected when we already have numerous delay events as singleton causes: firstly, all supersets of those singleton causes qualify as but-for causes and secondly also conversely, the analysis of what causes are minimal gets strongly affected by having those singleton sets already as minimal causes.

There are different ways to respond to this, for our purposes, oftentimes undesired behavior. In theory, it would be possible to require the counterfactual run to also

perform an infinite number of actions. However, preserving the decidability of this adapted definition seems to require an elaborate encoding of the additional property that would enlarge the automaton size and effect formula. Moreover, in practice, this property cannot be stated in UPPAAL's language, so we choose to avoid the behavior by manually excluding those singleton causes whenever necessary by adding upper bounds as invariants to the respective locations. △

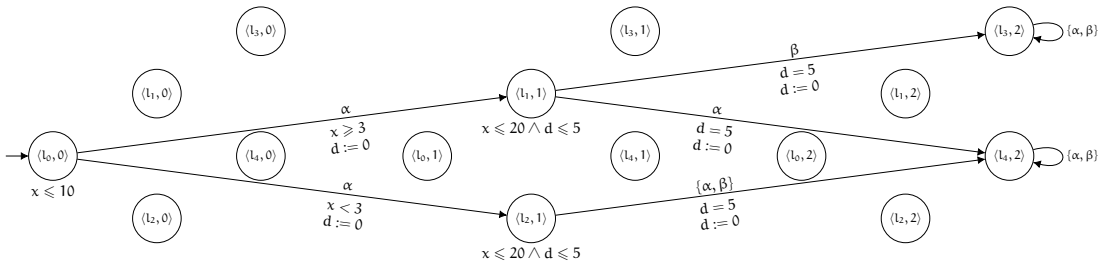
The following examples shows, that it might then indeed be the case that singleton events are not enough to explain a certain effect. Instead, causes with multiple events have to be considered.

Example 3.1.3. Consider a small variation of the timed automaton and let TA now be the following:



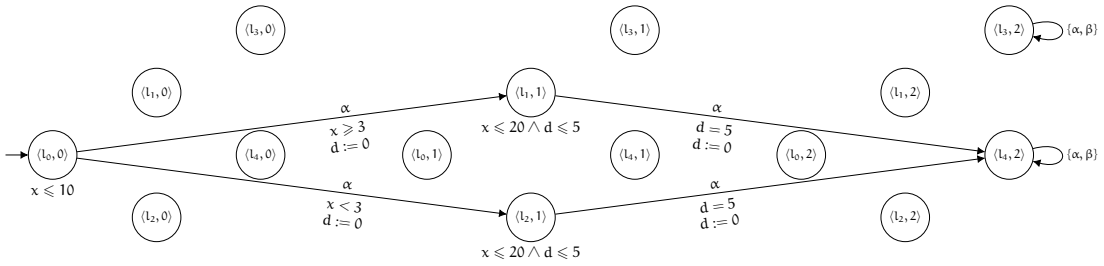
We have, for instance, the run $\mathrm{rho} := l_0 \xrightarrow{2} \xrightarrow{\alpha} l_2 \xrightarrow{5} \xrightarrow{\alpha} l_4$ with corresponding trace ξ. The set of events $\mathcal{C} := \{(2,1),(\alpha,2)\}$ is a but-for cause for the unchanged effect $\phi := \Diamond\, l_4$, that is now indeed also minimal:

The **SAT**-condition is easy to verify, for the $\mathbf{CF}_{\mathsf{BF}}$-condition we consider again the intersection automaton $\mathsf{TA} \cap \mathsf{TA}^{\mathcal{C}}_{\xi}$:

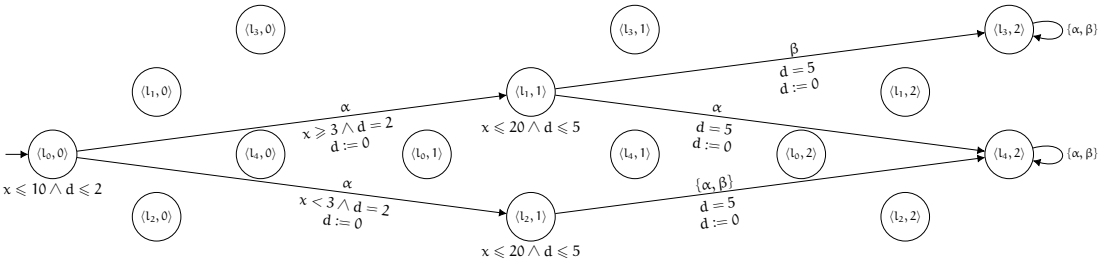


Here, we find, for example, $\rho' := \langle l_0,0\rangle \xrightarrow{3} \xrightarrow{\alpha} \langle l_1,1\rangle \xrightarrow{5} \xrightarrow{\beta} \left(\langle l_3,2\rangle \xrightarrow{1} \xrightarrow{\alpha}\right)^{\omega}$ as time-divergent witnessing run . For the **MIN**-condition, we have to check that neither

$\mathcal{C}_1 := \{(2,1)\}$ nor $\mathcal{C}_2 := \{(\alpha,2)\}$ is a but-for cause. When looking at the intersection automata, we notice however that in fact a singleton cause does not suffice to find a run not reaching l_4. For $\mathcal{C}_1$, $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}_1}$ has no β-transition leading to $\langle l_3, 2\rangle$:



For $\mathcal{C}_2$ in turn, $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}_2}$ enforces the first delay to be 2, preventing the transition to location $\langle l_2, 1\rangle$ to be traversable:

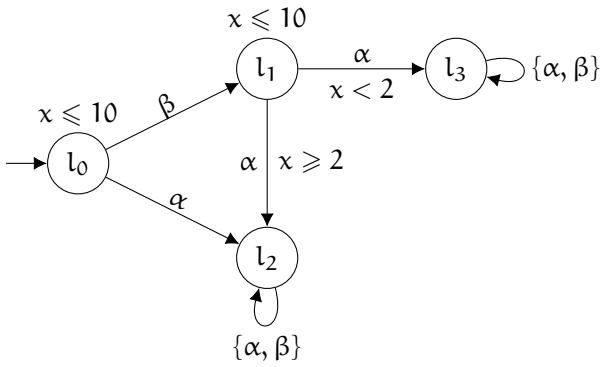


Hence, there are no counterfactual runs for the smaller sets of events $\mathcal{C}_1$ and $\mathcal{C}_2$. △

For this inference, the aforementioned manual exclusion of singleton delay cause sets was indeed necessary and realized by the invariants in the locations l_0, l_1, and l_2. Without those invariants, we had both the sets $\{(2,1)\}$ and $\{(5,2)\}$ as but-for causes and, therefore, in particular $\{(2,1),(\alpha,2)\}$ no longer as a minimal but-for cause.

In the next example, we show that it is also possible to have a causal delay and a causal action event at the same position. Furthermore, we demonstrate how our concept can identify a delay to be causal for the behavior of the timed automaton later in the run, which will also explain why it is at all important to require counterfactual runs to be time-divergent.

Example 3.1.4. We consider the following timed automaton TA:

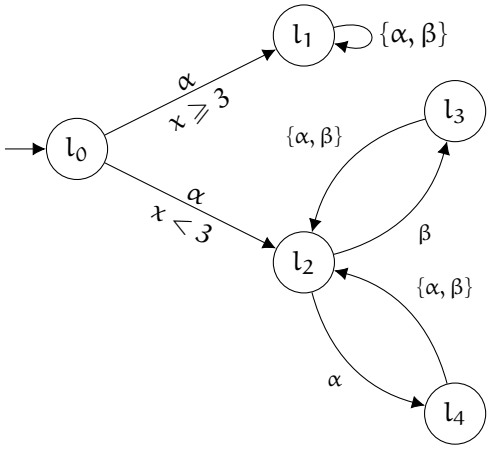


The effect $\phi := \Diamond\, l_2$, i.e., reaching location l_2, appears in the run $\rho := l_0 \xrightarrow{2} \xrightarrow{\alpha} l_2$ and $\mathcal{C} := \{(2,1),(\alpha,1)\}$ is a minimal but-for cause for ϕ on ρ in TA. The reasoning works as above, we merely take a closer look at why the strictly smaller set $\{(\alpha,1)\}$ is no but-for cause: the causal action allows a counterfactual run via l_1. However, the fixed first delay results in the clock assignment $\{x \mapsto 2\}$ when entering l_1 for which the transition guard $x < 2$ leading to l_3 is not enabled. Hence, the only time-divergent continuation leads to location l_2 such that no time-divergent run without appearing effect exists. △

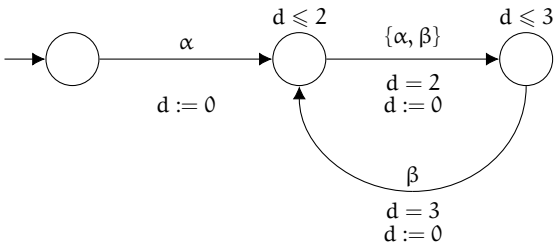
Notice, that in the above example indeed the delay of 2 in the first location is identified to be causal for a disabled transition and therefrom resulting behavior later in the run. Furthermore, the example shows, as already indicated, the necessity of the requirement for the counterfactual run to be time-divergent: If we would only analyze counterfactual runs up to the length of the actual run, we would not recognize that in every possible continuation of the potential counterfactual run $l_0 \xrightarrow{2} \xrightarrow{\alpha} l_1$ the effect still appears. As a consequence, we would mistakenly identify $\{(\alpha,1)\}$ as a singleton cause for the effect.

Our last synthetic example demonstrates the explanation of further and more involved effects, for which we also deal for the first time with actual runs in lasso-shaped from.

Example 3.1.5. Consider the following timed automaton TA:



The lasso-shaped run $\rho := l_0 \xrightarrow{4} \xrightarrow{\alpha} (l_1 \xrightarrow{2} \xrightarrow{\beta} l_1 \xrightarrow{3} \xrightarrow{\beta})^\omega$ does not reach location l_2, i.e., the effect $\phi_1 := \rho \models \neg\Diamond\, l_4$ appears. We identify $C_1 := \{(4,1)\}$ as minimal but-for cause for ϕ_1 in ρ of TA. Furthermore, also the effect $\phi_2 := \models \neg\Box\Diamond\, l_4$ appears in ρ, that is, ρ does not visit l_4 infinitely often[2]. Here, we identify $C_1 := \{(4,1),(\beta,2)\}$ as minimal but-for cause for ϕ_2 in ρ of TA. The reasoning thereof works analogously as above, now using – for example for C_2 – the following counterfactual trace automaton $\mathsf{TA}^{\mathcal{C}_2}_{\xi}$, whereby ξ is again the corresponding trace of ρ:



△

Lastly, we revisit the real-time model of a coffee machine from Section 2.2 as an example for a simplified real-world scenario in which we can explain different system behaviors.

Example 3.1.6. Recall the automaton TA given in Figure 2.1 modeling a coffee machine. In Example 2.2.2, we then considered its run

→ Figure 2.1, p. 14
→ Example 2.2.2, p. 17

$$\rho := \mathsf{init} \xrightarrow{4} \xrightarrow{\mathsf{push}} \mathsf{prod} \xrightarrow{2} \xrightarrow{\mathsf{wait}} \mathsf{prod} \xrightarrow{3} \xrightarrow{\mathsf{wait}} \mathsf{coffee}.$$

Now, we can analyze the cause behind the effect that coffee is produced: $\{(\mathsf{push},1)\}$ is a minimal but-for cause for $\phi_1 := \rho \models \Diamond\, \mathsf{coffee}$ in ρ of TA.

Furthermore, we can also reason about time related effects and analyze, for instance, the cause why coffee is produced in a maximum of 10 time units, i.e., for the effect $\phi_2 := \rho \models \Diamond_{[0,10]}\, \mathsf{coffee}$. For ϕ_2, we now also have $\{(4,1)\}$ as minimal but-for cause. △

Note how our notion of but-for causality identifies in the above example causes for real-time behaviors in perfect accordance with our intuition: For the supply of coffee in a particular time interval, we would regard both the performed action leading to the production of coffee as well as the timing of the start of the production as singleton causes.

3.1.3 Actual Causality in Real-Time Systems

As discussed in Section 2.1, the concept of but-for causality comes with the issue of

→ Section 2.1, p. 7

[2] As ϕ_2 is not expressible in the restricted language of UPPAAL, this is example is as the only one in the course of the thesis not fully handled by the developed causality tool (cf. Chapter 5).

preemption: if the causal events preempt a further potential cause, but-for causality will not be able to identify the events as a cause. This issue appears in fact not only in the foundational setting, it can also be observed for our notion of but-for causality in real-time systems.

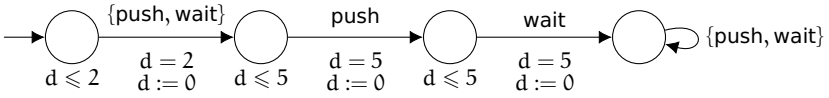
→ Figure 2.1, p. 14

Example 3.1.7. Consider again the automaton TA in Figure 2.1 modeling a coffee machine and now its run

$$\rho := \mathsf{init} \xrightarrow{2,\,\mathsf{push}} \mathsf{prod} \xrightarrow{5,\,\mathsf{push}} \mathsf{coffee} \xrightarrow{5,\,\mathsf{wait}} \mathsf{init}$$

in which the effect $\phi := \Diamond\,\mathsf{coffee}$ appears. The set of events $\mathcal{C} := \{(\mathsf{push}, 1)\}$ is *no* but-for cause for ϕ on ρ of TA:

With corresponding trace ξ, the counterfactual trace automaton $\mathsf{TA}_\xi^{\mathcal{C}}$ allows an alternative action for the first transition but enforces the action push for the second transition:



Hence, the only candidate for a prefix of a time-divergent counterfactual run is the one with corresponding trace $\langle 2, \mathsf{wait}\rangle\langle 5, \mathsf{push}\rangle\langle 5, \mathsf{wait}\rangle$. The corresponding run in the intersection $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ of this trace is however

$$\langle \mathsf{init}, 0\rangle \xrightarrow{2,\,\mathsf{wait}} \langle \mathsf{init}, 1\rangle \xrightarrow{5,\,\mathsf{push}} \langle \mathsf{prod}, 2\rangle \xrightarrow{5,\,\mathsf{wait}} \langle \mathsf{coffee}, 3\rangle$$

such that there cannot exist a counterfactual run in which ϕ does not appear. Hence, the set of events $\mathcal{C}$ does not fulfill the $\mathbf{CF}_{\mathsf{BF}}$-condition. △

→ Example 2.1.3, p. 11

Similar to Example 2.1.3 in the foundational setting, also in this situation the first push of the coffee machine button is not identified as but-for cause due to the preempted second push. Only both push events together are identified as but-for cause. This is, however, again in contradiction to our intuition: intuitively we would argue that the first push alone caused the coffee to be made. In particular, the second push actually results only in traversing the self-loop in the "prod"-location but has no influence on reaching the effect and should therefore not be identified as cause.

→ Definition 2.2, p. 12

Following the ideas of Halpern and Pearl in the foundational setting (cf. Definition 2.2), we extend the basic notion of but-for causality with the concept of contingencies to receive the more advanced notion of actual causality now also for our real-time context. Recall that contingencies allowed to fix parts of the counterfactual world to be as they had been in the actual world. This means for our setting that contingencies should allow to let parts of the counterfactual run to be as they had been in the actual run. Concretely, we want to allow in every step the configuration of the counterfactual run, i.e., its location and clock assignment, to be as it had been in the corresponding step of

the actual run. As already done to represent the possible counterfactual traces and again inspired by Coenen et al. [22], we use also for modeling contingencies an automata-based approach. More precisely, we extend the initially given timed automaton to a so called contingency automaton that preserves the structure of the initial automaton but enables contingencies with respect to the given run.

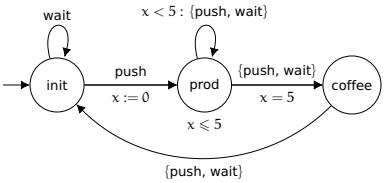
We demonstrate the construction of this contingency automaton exemplary for the situation in Example 3.1.7, so consider the coffee automaton from Figure 2.1 with its run

→ Example 3.1.7, p. 38

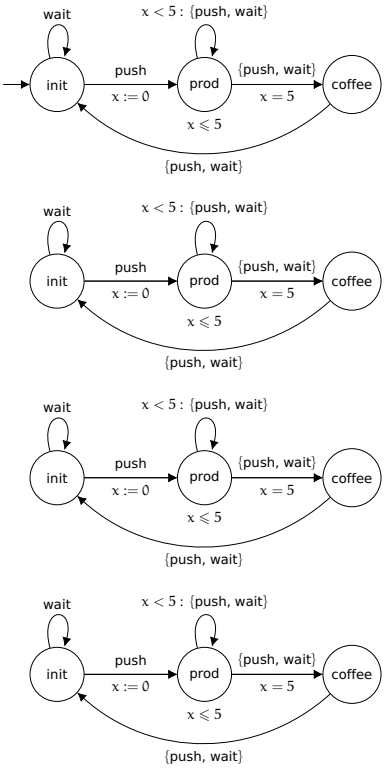
$$\rho := \langle \mathsf{init}, \{x \mapsto 0\} \rangle \xrightarrow{2} \xrightarrow{\mathsf{push}} \langle \mathsf{prod}, \{x \mapsto 0\} \rangle \xrightarrow{5} \xrightarrow{\mathsf{push}} \langle \mathsf{coffee}, \{x \mapsto 5\} \rangle \xrightarrow{5} \xrightarrow{\mathsf{wait}} \langle \mathsf{init}, \{x \mapsto 10\} \rangle.$$

We proceed as follows:

1. We start with the given initial timed automaton, so in our case:



2. We duplicate the automaton by the number of transitions in the given run, so in our case we copy the automaton three times to obtain four copies overall:



3. We redirect the transitions in all copies but the last to now lead to its target location in the respective next copy. The transitions in the last copy remain unchanged. The automaton of this step is given in Figure 3.2 depicted below.

4. We add the transitions for allowing contingencies: With every step, the automaton should be able to reset exactly to the configuration the given run had in this step. To do so, we add transitions to the respective location and enforce the clock assignment using a suitable clock expression. The final contingency automaton is depicted in Figure 3.3. To the right, we specify the actions and expressions of the contingency transitions distinguished by different colors.

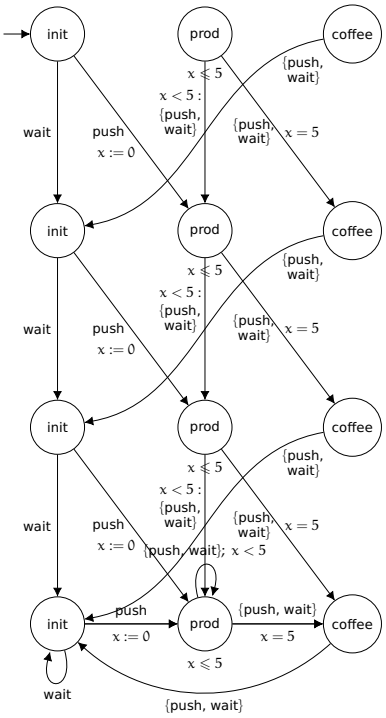


Figure 3.2: Step 3

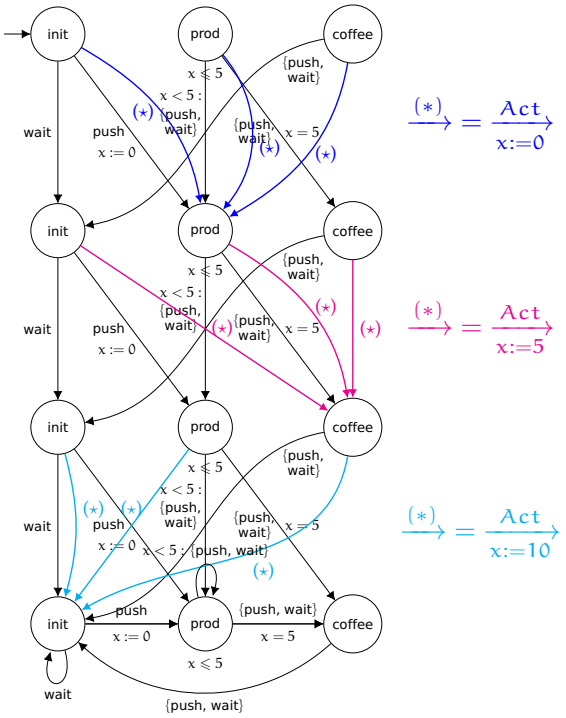


Figure 3.3: Step 4 (Final Automaton)

The construction described in Steps 1-3 is a well-known technique to count and store the number of transition the automaton has already taken: By always proceeding in the next copy, we know that exactly i transitions must have been taken when the automaton is in copy i, for $i = 1, \ldots, n-1$. Only this knowledge then allows us to add contingency transitions, whose target location dependents on the current step, correctly. This yields the for our purpose desired result as we allow the automaton to reset its location and clock setting exactly to what they actually were in the respective step. Note that we want this reset to be always possible, hence, we do have outgoing transitions from every location of the previous copy, allow every action for this transition and do not restrict

the transition with a guard. Finally, the clocks are reset by using the clock expression of composed assignments that sets all clocks exactly to the value they had in the respective configuration.

Formally, the contingency automaton is defined as follows.

Definition 3.6 (Contingency Automaton)

Let $\rho = \langle l_0, u_0\rangle \xrightarrow{\delta_1}\xrightarrow{\alpha_1} \langle l_1, u_1\rangle \xrightarrow{\delta_2}\xrightarrow{\alpha_2} \ \dots \ \xrightarrow{\delta_n}\xrightarrow{\alpha_n} \langle l_n, u_n\rangle$ be a finite run of a timed automaton $\mathsf{TA} = (Loc, l_0, C, Act, \rightarrow, I)$. The *contingency automaton* of timed automaton TA and run ρ is defined as $\mathsf{TA}_\rho^{con} := (Loc', l_0', C', Act, \rightarrow', I')$ with

Def. contingency automaton

- $Loc' := Loc \times \{0, \dots, n\}$
- $l_0' := \langle l_0, 0\rangle$
- $C' := C$
- the transition relation $\rightarrow'$ is defined by the following rules:

$$\frac{l \xrightarrow{g:\,\alpha,\,E} l' \qquad i = 0, \dots, n-1}{\langle l, i\rangle \xrightarrow{g:\,\alpha,\,E} \langle l', i+1\rangle} \qquad \frac{l \xrightarrow{g:\,\alpha,\,E} l'}{\langle l, n\rangle \xrightarrow{g:\,\alpha,\,E} \langle l', n\rangle}$$

$$\frac{i = 0, \dots, n-1 \qquad \alpha \in Act}{\langle l, i\rangle \xrightarrow{\top:\,\alpha,\,E(u_{i+1})} \langle l_{i+1}, i+1\rangle}$$

  where in the last rule, l_{i+1} is the respective location of ρ, u_{i+1} is the respective clock assignment of ρ, and $E(u)$ is the expression enforcing the clock assignment to be exactly u, i.e., $E(\{c_1 \mapsto v_1, \dots, c_n \mapsto v_n\}) := \ c_1 := v_1; \ \dots; \ c_n := v_n$

- $I(\langle l, i\rangle) := I(l)$
- If TA has a labeling function L, the labeling function of TA_ρ^{con} is defined as $L'(\langle l, i\rangle) := L(l)$.

For a lasso-shaped run

$$\rho = \langle l_0, u_0\rangle \xrightarrow{\delta_1}\xrightarrow{\alpha_1} \dots \xrightarrow{\delta_n}\xrightarrow{\alpha_n} \Big(\langle l_n, u_n\rangle \xrightarrow{\delta_{n+1}}\xrightarrow{\alpha_{n+1}} \dots \xrightarrow{\delta_{p-1}}\xrightarrow{\alpha_{p-1}} \langle l_{p-1}, u_{p-1}\rangle \xrightarrow{\delta_p}\xrightarrow{\alpha_p}\Big)^\omega$$

the construction of the contingency automaton TA_ρ^{con} works as above only with using as set of locations $Loc' := Loc \times \{0, \dots, n, n+1, \dots, p-1\}$, with identifying in the definition of the transition relation $\rightarrow$ index p with n, and with omitting the second rule in the definition of the transition relation.

Again the three rules defining the transition relation mirror the three types of transitions that we want to have in the contingency automaton:

- In the first $n-1$ copies, we redirect the transitions from the initial automaton to lead to the next copy.
- In the n-th copy, we leave the transitions from the initial automaton unchanged.
- The third rule takes care of the contingency transitions and allows to reset the system when entering copy $i+1$ to the configuration $\langle l_{i+1}, u_{i+1} \rangle$ of the given run.

Note that the contingency automaton is no longer necessarily timewise action-deterministic – in fact the automaton is almost always and also in the above example of the coffee machine not action-deterministic. Either a contingency transition can be traversed, that is, that the contingency is taken, or the "normal" transitions from the initial automaton can be traversed, that is, that the contingency is not taken. This non-determinism will, however, not cause any major formal problems. For more details of this aspect, we refer to the proof of Proposition 12 and the discussion in Section 3.3. Like in the construction of the intersection automata, we also define the labeling function of the contingency automaton exactly as the projection to the label of the initially given automaton such that run effects transport again as desired.

→ Proposition 12, p. 46
→ Section 3.3, p. 55

Using the contingency automaton, we can now define a notion of actual causality in our real-time setting. The definition reads exactly as the one for but-for causality, except for the detail that we intersect in the **CF**-condition no longer the given initial automaton with the counterfactual trace automaton but that we now consider the intersection $\mathsf{TA}_\rho^{\mathsf{con}} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ of the contingency automaton and the counterfactual trace automaton as model for the counterfactual simulation.

Definition 3.7 (Actual Causality in Real-Time Systems)

Let TA be a timed automaton, ρ a run of TA with corresponding delay trace ξ, and ϕ a run effect. A set of events $\mathcal{C} \subseteq \mathcal{E}$ is an *actual cause* for ϕ in ρ of TA, if the following three conditions hold:

Def. actual cause

SAT $\xi \models \mathcal{C}$ and $\rho \models \phi$.

CF$_{\mathsf{Act}}$ There is a time-divergent run ρ' in $\mathsf{TA}_\rho^{\mathsf{con}} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ with $\rho' \not\models \phi$.

MIN $\mathcal{C}$ is minimal, i.e., no strict subset of $\mathcal{C}$ satisfies **SAT** and **CF**$_{\mathsf{Act}}$.

The non-determinism of the contingency automaton and, therefore, also the non-determinism of the intersection is now in fact a crucial detail ensuring that we obtain a notion of actual causality that is in accordance with the foundational concept. We will discuss this relationship and the differences to Coenen et al.'s techniques [22] for obtaining a faithful modeling of contingencies in more depth in Section 3.3.

→ Section 3.3, p. 55

Recall that, in contrast to our definition, the $\mathbf{CF}_{\mathsf{Act}}$-condition in the foundational setting asked explicitly for the existence of a set of contingencies that are then fixed in the count

As desired, the notion of actual causality does indeed solve the problem of preemption also now in the real-time setting.

Example 3.1.8. We recall the situation from Example 3.1.7 with timed automaton TA from Figure 2.1 and run

→ Example 3.1.7, p. 38
→ Figure 2.1, p. 14

$$\rho := \mathsf{init} \xrightarrow{2} \xrightarrow{\mathsf{push}} \mathsf{prod} \xrightarrow{5} \xrightarrow{\mathsf{push}} \mathsf{coffee} \xrightarrow{5} \xrightarrow{\mathsf{wait}} \mathsf{init}.$$

We discussed that the set of events $\mathcal{C} := \{(\mathsf{push}, 1)\}$ is no but-for cause for the effect $\phi := \Diamond\,\mathsf{coffee}$, $\mathcal{C}$ is, however, an actual cause for ϕ in ρ of TA:

When intersecting the contingency automaton $\mathsf{TA}^{\mathsf{con}}_{\rho}$ (cf. Figure 3.3) with the counterfactual trace automaton, we can find the time-divergent counterfactual run

→ Figure 3.3, p. 40

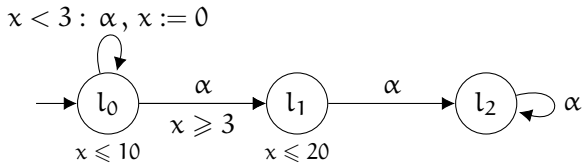
$$\rho' := \mathsf{init} \xrightarrow{2} \xrightarrow{\mathsf{wait}} \mathsf{init} \xrightarrow{5} \xrightarrow{\mathsf{push}} \mathsf{prod} \xrightarrow{5} \xrightarrow{\mathsf{wait}} \left(\mathsf{init} \xrightarrow{5} \xrightarrow{\mathsf{wait}}\right)^{\omega}$$

in which the effect ϕ does not appear[3]. Hence the $\mathbf{CF}_{\mathsf{Act}}$ condition is fulfilled. △

The counterfactual run ρ' in the above example was indeed only enabled due to the additional contingency transitions as it takes the contingency in the last step to traverse from location "prod" to location "init", a transition that does not exist in the initial automaton.

We present a further example for the improved capability of actual causality in which a delay event now preempts another potentially causal delay event.

Example 3.1.9. We consider the following timed automaton TA:



In the run

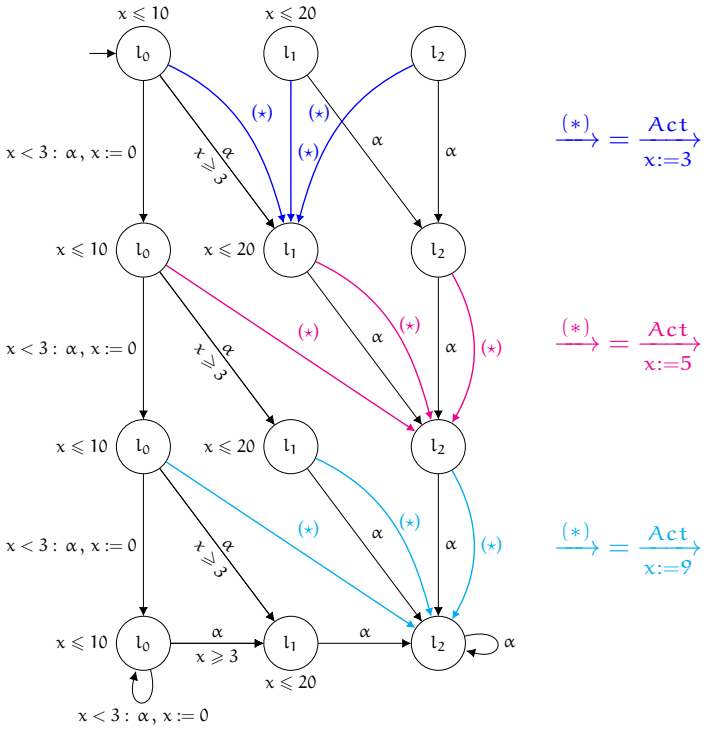
$$\rho := l_0 \xrightarrow{3} \xrightarrow{\alpha} l_1 \xrightarrow{2} \xrightarrow{\alpha} l_2 \xrightarrow{4} \xrightarrow{\alpha} l_2$$

the effect $\phi := \Diamond\, l_1$ appears. The set of events $\mathcal{C} := \{(3, 1)\}$ is no but-for cause, it is, however, an actual cause for ϕ in ρ of TA:

[3] Like here, we might also in the future abuse notation in contingency automata as well as in the counterfactual runs and simply write for the locations only the component with the name of the location in the initial timed automaton and omit the further components of the tuple-locations generated by composition.

$\mathcal{C}$ violates the $\mathbf{CF}_{\mathsf{BF}}$-condition, since the delay of 4 at the third position remains enforced, so every potential counterfactual run will again reach location l_1 after the third transition such that the effect still appears.

The $\mathbf{CF}_{\mathsf{Act}}$-condition is, however, fulfilled: To see this, we consider the contingency automaton $\mathsf{TA}_\rho^{\mathsf{con}}$:



Taking the contingency in the third step allows to take a transition from l_0 to l_2 and thereby to avoid traversing location l_1 with the counterfactual run

$$\rho' := l_0 \xrightarrow{2} \xrightarrow{\alpha} l_0 \xrightarrow{2} \xrightarrow{\alpha} l_0 \xrightarrow{4} \xrightarrow{\alpha} (l_2 \xrightarrow{1} \xrightarrow{\alpha})^\omega.$$

Checking the **SAT**- and **MIN**-condition is again straightforward. △

Also in this example we would argue that the concept of actual causality matches more precisely with the intuition on what event(s) should be identified as cause: In the above situation, we have the minimal but-for cause $\{(3,1),(4,3)\}$ – it seems however unreasonable to say that the third delay in run ρ is causal for traversing l_1 since the specific delay of 4 has firstly no impact at all on the course of the run and appears secondly temporally even after the considered effect appears. Again, the event $(4,3)$ must be included in the but-for cause only because it is preempted by the first delay event $(3,1)$. And again, only by taking a contingency, we can solve this issue since the contingency automaton allows transitions that have not existed in the originally given automaton. With this, we can find a counterfactual run even though we still have a

delay of 4 in the third step.

Lastly, we want to discuss some theoretical results regarding our notion of actual causality and also its connection to but-for causality. We start by lifting the statement from Proposition 1 to our real-time setting. This needs, however, first an auxiliary result regarding the runs of the contingency automaton.

→ Proposition 1, p. 12

Lemma 10. *Let* TA *be a labeled timed automaton and* ρ *a finite or lasso-shaped run of* TA*. Then there is for every time-divergent run* ρ' *of* TA *a time-divergent run* ρ'_{con} *of* $\mathsf{TA}_\rho^{\mathsf{con}}$ *with the same corresponding signal.*

Proof. Let ρ be a finite run of TA with length n and

$$\rho' = l_0 \xrightarrow{\delta_1}\xrightarrow{\alpha_1} l_1 \ \dots \ \xrightarrow{\delta_i}\xrightarrow{\alpha_i} l_i \dots \qquad\qquad i = 2, 3, \dots$$

a time-divergent run of TA. By the definition of the transition function of $\mathsf{TA}_\rho^{\mathsf{con}}$, there is then also the time-divergent run

$$\rho'_{\mathsf{con}} = \langle l_0, 0\rangle \xrightarrow{\delta_1}\xrightarrow{\alpha_1} \langle l_1, 1\rangle \ \dots \ \xrightarrow{\delta_n}\xrightarrow{\alpha_n} \langle l_n, n\rangle \xrightarrow{\delta_{n+1}}\xrightarrow{\alpha_{n+1}} \langle l_{n+1}, n\rangle \ \dots \ \xrightarrow{\delta_i}\xrightarrow{\alpha_i} \langle l_i, n\rangle \dots$$
$$i = n+2, n+3, \dots$$

of $\mathsf{TA}_\rho^{\mathsf{con}}$ that has by definition of the labeling function of $\mathsf{TA}_\rho^{\mathsf{con}}$ the same corresponding signal as ρ. For lasso-shaped ρ, there is analogously a lasso-shaped run ρ'_{con} in $\mathsf{TA}_\rho^{\mathsf{con}}$. □

This allows to lift the foundational result stating the connection between but-for and actual causality:.

Proposition 11. *If a set of events* $\mathcal{C}$ *is a (minimal) but-for cause for a run effect* ϕ *in a run* ρ *of a timed automaton* TA*, then there is a subset* $\mathcal{C}' \subseteq \mathcal{C}$ *that is an actual cause for* ϕ *in* ρ *of* TA*. In particular, every singleton but-for cause is also an actual cause.*

Proof. Analogously to the proof of Proposition 1 it suffices to show that if $\mathcal{C}$ fulfills the $\mathbf{CF}_{\mathsf{BF}}$-condition it does also fulfill the $\mathbf{CF}_{\mathsf{Act}}$-condition. So let $\mathcal{C}$ fulfill $\mathbf{CF}_{\mathsf{BF}}$, that is, that there is a time-divergent run ρ' of $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ with $\rho' \not\models \phi$. As shown in the proof of Corollary. 4, the projection runs ρ'_1 and ρ'_2 are, consequently, runs of TA and $\mathsf{TA}_\xi^{\mathcal{C}}$ respectively. By Lemma 10, there is then also the run $\rho'_{1_{\mathsf{con}}}$ in $\mathsf{TA}_\rho^{\mathsf{con}}$ and again as in the proof of Corollary. 4, we have, therefore, a run of $\mathsf{TA}_\rho^{\mathsf{con}} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ (namely the run pointwise composed from $\widehat{\rho}'_1$ and ρ'_2). Since the corresponding signal is by construction of the labeling functions not changed by all those run operations (projection, pairing, building of the contingency automaton), ϕ does still not appear in the time-divergent run of $\mathsf{TA}_\rho^{\mathsf{con}} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ such that $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{Act}}$. □

→ Proposition 1, p. 12
→ Corollary. 4, p. 19
→ Lemma 10, p. 45
→ Corollary. 4, p. 19

For but-for causality, we established two sanity results, namely that the empty set is never a cause and an existence statement of causes. We can show both results also for the notion of actual causality.

Proposition 12. *$\emptyset$ is never an actual cause for a run effect ϕ in a run ρ of a time-wise action-deterministic timed automaton* TA*.*

Proof. The proof works similar to the proof of Proposition 8. In the case of $\rho \models \phi$, we however used that TA is action-deterministic to conclude that a potential counterfactual run agrees with the actual run ρ. Even though $\mathsf{TA}_\rho^{\mathsf{con}}$ is no longer action-deterministic in general and, therefore, Proposition 2 no longer applicable, we can still draw the necessary inference for ρ: The non-determinism in the contingency automaton results from the additional contingency transitions

→ Proposition 2, p. 17

$$\langle l, i\rangle \xrightarrow{\top:\, \alpha,\, E(u_{i+1})} \langle l_{i+1}, i+1\rangle,$$

where $\langle l_{i+1}, u_{i+1}\rangle$ are the configurations of ρ. Therefore, the contingency transitions only allow to traverse exactly the locations of ρ such that we can still use that all runs of the corresponding trace of ρ must agree with ρ. Hence, the overall inference still works as in the proof of Proposition 8 and the result follows. □

Proposition 13. *Let ϕ be a run effect and* TA *a timed automaton. If there exists a time-divergent run ρ' of* TA *with $\rho' \not\models \phi$, then for every run ρ in which ϕ appears, there is an actual cause for ϕ in ρ of* TA*.*

→ Proposition 9, p. 31
→ Proposition 11, p. 45

Proof. Follows directly with Proposition 9 and Proposition 11. □

Recall again, that we do not know how to formally prove causality definitions to be "correct definitions"[4]. Moreover, we will report on some limitations and problems our definitions struggle with in following sections. We still think that the presented theoretical results together with the numerous considered examples, in which our notions of but-for and actual causality work as desired, provide solid reasons for the meaningfulness and usefulness of our definitions.

3.2 Timestamp Causality

In the last section, we introduced an approach towards counterfactual causality in real-time systems that looks at a given run and its corresponding trace from a delay perspective, that is, the delay values in the run were considered as causal events. This choice is by far not canonical: In Chapter 6, we discuss multiple other concepts in the related work of others, where we will see even settings in which one does – in contrast to our work – not even starts with a given run or trace. In this section now, we want to discuss an alternative to the delay perspective and look instead at the absolute timestamps of actions in timed traces. It turns out that the intuition is not sweeping in support of either one of the two perspectives alone. Instead, we will see both examples,

→ Chapter 6, p. 105

[4] See in this regard also again the remark by Halpern quoted in Chapter 1.

in which we intuitively prefer a delay perspective, as well as examples, in which a timestamp perspective seems to match our intuition way better.

We want to start with an everyday example that motivated us in the first place to take a different perspective than that of delay events:

Example 3.2.1. Consider the following (very simplified) description of an agricultural field in the course of the year.

- The agricultural field is cultivated and is therefore sown, fertilized, watered and finally harvested throughout the year.
- In order to have a successful harvest it is necessary that the sowing, fertilizing, watering and harvesting is done exactly in the right months of the year:
  - The field must be sown in March,
  - the field must be fertilized in April,
  - the field must be watered in July, and
  - the field must be harvested in September.
- If just one of the actions is not done in the right month, the harvest will fail.

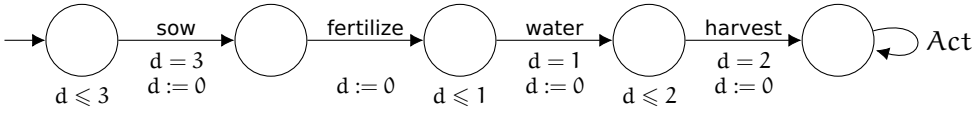
Now imagine a scenario in which a farmer mistakenly fertilizes the field not in April but only in June while he sows the field correctly in March, waters it correctly in July, and harvests it correctly in September. Intuitively, we would suggest that the wrong point of time at which the farmer fertilized the field and nothing else caused the harvest to fail.

We want to model the agricultural field as a real-time system. We do so with a timed automaton TA, using the set of actions $\mathrm{Act} := \{\mathsf{sow}, \mathsf{fertilize}, \mathsf{water}, \mathsf{harvest}\}$, and a clock m whose value represents the current month of the year. The automaton TA is depicted in Figure 3.4 on the next page. The above scenario of a faulty fertilizing in June can be rediscovered in the timed automaton as the run

→ Figure 3.4, p. 48

$$\rho := l_0 \xrightarrow{3} \xrightarrow{\mathsf{sow}} l_1 \xrightarrow{3} \xrightarrow{\mathsf{fertilize}} \mathsf{fail} \xrightarrow{1} \xrightarrow{\mathsf{water}} \mathsf{fail} \xrightarrow{2} \xrightarrow{\mathsf{harvest}} \mathsf{fail},$$

in which the effect $\phi := \Diamond\,\mathsf{fail}$ appears. Applying now however our notion of causality from the previous section yields the set of events $\{(3,2),(1,3)\}$, i.e., the second and third delays of the run as minimal but-for cause (and as well as actual cause) for ϕ in ρ of TA. In particular, the singleton set $\mathcal{C} := \{(3,2)\}$ of only the delay after which the field is fertilized is not identified as a cause on its own. The counterfactual trace automaton $\mathsf{TA}_\xi^{\mathcal{C}}$ for this singleton event set looks as follows:



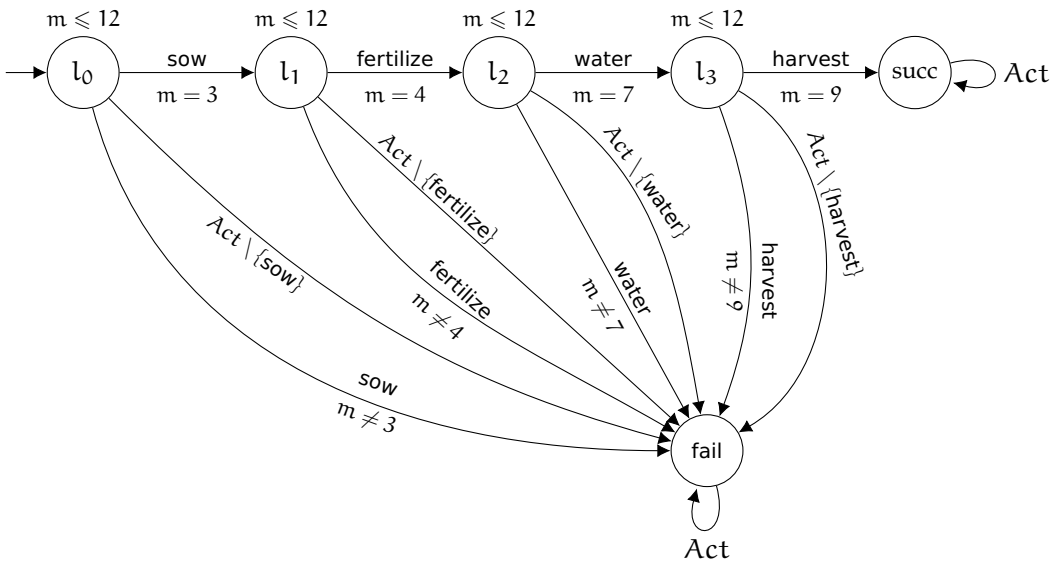


Figure 3.4: Timed automaton modeling an agricultural field

Although $\mathsf{TA}_{\xi}^{\mathfrak{C}}$ allows an alternative to 1 for the second delay – which means it allows to "correctly" fertilize in April – it then enforces a delay of 1 before the third action water. This then results in a faulty watering already in the month May, therefore, again in a failed harvest, and hence, there is no counterfactual run without effect ϕ. △

This example gives rise to a whole class of scenarios in which the notion of causality that takes a delay perspective fails to match with our intuition: for cases in which the total time of certain actions since the start of the timed automaton is of interest and not the time between certain actions, the delay perspective on causality might fail to precisely analyze the causal connections. This is due to the propagation of time: changing the time of a certain action by changing its preceding delay also shifts the absolute times since the start of the timed automaton of all following actions.

If now, however, the absolute time is of importance for how the run continues (as it is in our example, as the absolute time of when the field is watered and harvested is of importance) and if, in the actual run, the absolute times of the following actions were already "correct"[5] (as it was in our example, as the time of watering and harvesting was "correct" in July and September respectively), then a shift of the absolute times of those actions leads to "incorrect" times of the actions and, therefore, again to an appearance of the effect. In the delay perspective, the effect can therefore only be avoided by making also the successor delay causal, which then allows to shift the absolute times of the following actions back to how they were in the actual run (that is why in the above example the second and its successor delay, the third delay, are only together identified

[5]"Correct" in the sense that the absolute times were such that they could lead to an avoidance of the effect.

as a cause).

This observation suggests that there are scenarios in which it might be better for the causal analysis to allow changing the absolute time – or in the following also called timestamp – of certain actions in a run without shifting the timestamp of following actions. We can obtain such a notion of causality by taking a timestamp perspective to runs and its corresponding traces: Instead of considering delay traces, we now look at timestamp traces (cf. Definition 2.5) that have the form

→ Definition 2.5, p. 15

$$\xi = \langle t_1, \alpha_1 \rangle \dots \langle t_n, \alpha_n \rangle,$$

where $t_i \leqslant t_{i+1}$ for all $1 \leqslant i \leqslant n-1$. Instead of taking the delay spent in a location as an event, we now consider the absolute time that has passed since the start of the automaton as an event. Consequently, Definition 3.1 of sets of events is adapted.

→ Definition 3.1, p. 24

Definition 3.8 (Events – Timestamp Perspective)

1. The set of *timestamp events* is defined as $\mathcal{TE} := \{(t, i) \in \mathbb{R}_{\geqslant 0} \times \mathbb{N}\}$, the set of *action events* as $\mathcal{AE} := \{(\alpha, i) \in \mathtt{Act} \times \mathbb{N}\}$. **Def.** timestamp and action event
2. The whole *set of events* $\mathcal{E}$ is then obtained by the disjoint union of $\mathcal{TE}$ and $\mathcal{AE}$, i.e., $\mathcal{E} := \mathcal{TE} \dot\cup \mathcal{AE}$. **Def.** set of events

The principle ideas of our notion of causality in real-time systems remain, however, the same also now for the timestamp perspective and we proceed very similar as we did in the previous section for the delay perspective to obtain a definition of (minimal) but-for causality. The development must only be slightly adapted accordingly to the changed setting of considering timestamp traces and timestamp events instead of delay traces and delay events. This starts with Definition 3.2 of event satisfaction.

→ Definition 3.2, p. 25

Definition 3.9 (Event Satisfaction – Timestamp Perspective)

A finite timestamp trace $\xi = \langle t_1, \alpha_1 \rangle \dots \langle t_n, \alpha_n \rangle$ *satisfies a set of events* $\mathcal{C} \subseteq \mathcal{E}$, if **Def.** event satisfaction

- for every timestamp event $(t, i) \in \mathcal{C}$, we have $t = t_i$, and
- for every action event $(\alpha, i) \in \mathcal{C}$, we have $\alpha = \alpha_i$.

We denote this by $\xi \models \mathcal{C}$.

While the changes in the definitions of events and event satisfaction are minor and straightforward, the construction of counterfactual trace automata for timestamp traces and events requires as the only step in the adaption a little more discussion. Recall that given a trace and set of events the idea of counterfactual trace automata was to represent possible counterfactual traces, which was done by enforcing the non-causal

events of the given trace while allowing alternatives for the causal events. In advance
→ Definition 3.3, p. 27
of Definition 3.3, we explained in detail how to fix delay and action events and how the resulting constraints are then again relaxed for the causal events. The construction now in the timestamp perspective works similar: for action events, we can proceed exactly as already in the delay perspective. For the enforcement of timestamp events we need a slightly differing idea and use no longer a clock d measuring the delay spend in the current location but a clock t that measures the absolute time since the start of the timed automaton.

So consider again the demonstration example as already in the delay perspective with given trace

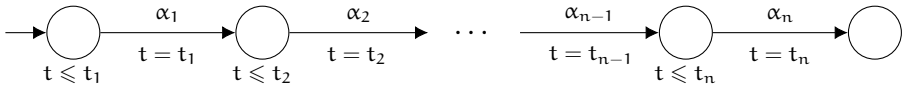
$$\xi = \langle t_1, \alpha_1\rangle\langle t_2, \alpha_2\rangle \dots \langle t_{n-1}, \alpha_{n-1}\rangle\langle t_n, \alpha_n\rangle$$

and a given set of events

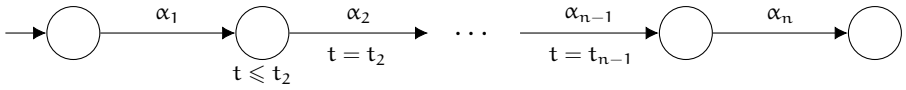
$$\mathcal{C} = \{(t_1, 1), (t_n, n), (\alpha_2, 2), (\alpha_n, n)\}.$$

We proceed in the following way:

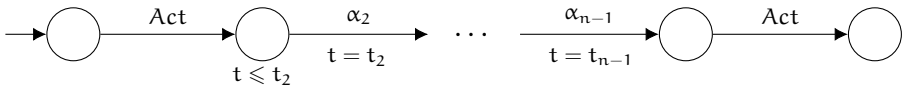
1. We enforce all timestamps and actions to be exactly as in the given actual trace using a clock t measuring the time since the start of the automaton:



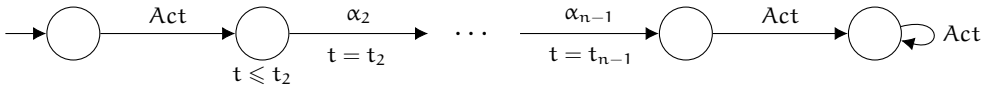
2. We relax the timestamp constraints as we did it for the delay constraints in the delay perspective by omitting guards and invariants:



3. We relax the action constraints exactly as we did already in the delay perspective:



4. We allow arbitrary continuations again as in the delay perspective:



The described construction results in the following adaptions of the formal definition of counterfactual trace automata.

Definition 3.10 (Counterfactual Trace Automaton – Timestamp Perspective)

Let $\xi = \langle t_1, \alpha_1 \rangle \dots \langle t_1, \alpha_n \rangle$ be a finite timestamp trace over a finite set of actions Act and let $\mathcal{C} \subseteq \mathcal{E}$ be a finite set of events. The *counterfactual trace automaton* of trace ξ for the set of events $\mathcal{C}$ is defined as $TA_\xi^{\mathcal{C}} := (Loc, l_0, C, Act, \to, I)$ with

Def. counterfactual trace automaton

- $Loc := \{0, \dots, n\}$,
- $l_0 := 0$,
- $C := \{t\}$,
- the transition relation $\to$ is defined by the following rules:

$$\frac{(t_i, i) \notin \mathcal{C} \qquad (\alpha_i, i) \notin \mathcal{C}}{i-1 \xrightarrow{t = t_i : \alpha_i} i} \qquad \frac{(t_i, i) \in \mathcal{C} \qquad (\alpha_i, i) \notin \mathcal{C}}{i-1 \xrightarrow{\top : \alpha_i} i}$$

$$\frac{(t_i, i) \notin \mathcal{C} \qquad (\alpha_i, i) \in \mathcal{C} \qquad \beta \in Act}{i-1 \xrightarrow{t = t_i : \beta} i} \qquad \frac{(t_i, i) \in \mathcal{C} \qquad (\alpha_i, i) \in \mathcal{C} \qquad \beta \in Act}{i-1 \xrightarrow{\top : \beta} i}$$

$$\frac{\beta \in Act}{n \xrightarrow{\beta} n}$$

- $I(i) := \begin{cases} t \leqslant t_{i+1}, & (t_{i+1}, i+1) \notin \mathcal{C}, \\ \top, & \text{otherwise.} \end{cases}$

Therewith, we can then again define (minimal) but-for causality in real-time systems now from a timestamp perspective.

Definition 3.11 (But-For Causality – Timestamp Perspective)

Let TA be a timed automaton, ρ a run of TA with corresponding timestamp trace ξ, and ϕ a run effect. A set of events $\mathcal{C} \subseteq \mathcal{E}$ is a *but-for cause* for ϕ in ρ of TA, if the following two conditions hold:

Def. but-for cause

SAT $\xi \models \mathcal{C}$ and $\rho \models \phi$.

CF$_{BF}$ There is a time-divergent run ρ' of $TA \cap TA_\xi^{\mathcal{C}}$ with $\rho' \not\models \phi$.

We call the cause *minimal but-for cause* if furthermore the following condition holds:

Def. minimal but-for cause

MIN $\mathcal{C}$ is minimal, i.e., no strict subset of $\mathcal{C}$ satisfies **SAT** and **CF$_{BF}$**.

➔ Definition 3.5, p. 30

In fact, we have thereby only changed a single word in comparison to Definition 3.5 as we now consider the corresponding *timestamp* trace instead of the corresponding *delay* trace. All further adaptions happen then implicitly as we now refer with $\mathcal{C} \subseteq \mathcal{E}$ to a subset $\mathcal{C}$ of a differing whole set of events $\mathcal{E}$, the satisfaction predicate $\xi \models \mathcal{C}$ is defined differently for this differing set of events $\mathcal{C}$ and trace ξ, and also the counterfactual trace automaton $\mathsf{TA}_\xi^{\mathcal{C}}$ is now defined differently for the timestamp trace ξ.

➔ Example 3.2.1, p. 47

As desired, this perspective on causality now yields a different causal analysis for scenarios like the agricultural field from Example 3.2.1, that is now in accordance with our intuition.

➔ Figure 3.4, p. 48

Example 3.2.2. Consider again the timed automaton TA given in Figure 3.4 modeling an agricultural field and its run

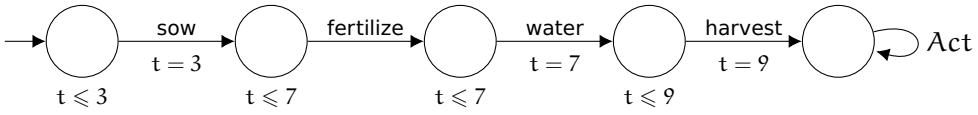
$$\rho := l_0 \xrightarrow{3} \xrightarrow{\mathsf{sow}} l_1 \xrightarrow{3} \xrightarrow{\mathsf{fertilize}} \mathsf{fail} \xrightarrow{1} \xrightarrow{\mathsf{water}} \mathsf{fail} \xrightarrow{2} \xrightarrow{\mathsf{harvest}} \mathsf{fail}$$

in which the effect $\phi := \Diamond\,\mathsf{fail}$ appears. From the timestamp perspective, the singleton set of events $\mathcal{C} := \{(6, 2)\}$ is now a but-for cause for ϕ on ρ of TA:

We consider the corresponding timestamp trace

$$\xi = \langle 3, \mathsf{sow}\rangle\langle 6, \mathsf{fertilize}\rangle\langle 7, \mathsf{water}\rangle\langle 9, \mathsf{harvest}\rangle$$

of run ρ. At first, we have $\xi \models \mathcal{C}$ such that the **SAT**-condition is fulfilled. Next, we construct the counterfactual trace automaton $\mathsf{TA}_\xi^{\mathcal{C}}$ that looks for the timestamp trace ξ and set of timestamp events $\mathcal{C}$ as follows:



This counterfactual trace automaton now allows an alternative for the time of fertilizing but fixes the absolute timestamp of all other actions, in particular the timestamp of the following watering and harvesting. Hence, the alternative of fertilizing at absolute time 4 without changing the other timestamps is now possible and we can find in the intersection $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ the counterfactual run

$$\rho' := l_0 \xrightarrow{3} \xrightarrow{\mathsf{sow}} l_1 \xrightarrow{1} \xrightarrow{\mathsf{fertilize}} l_2 \xrightarrow{3} \xrightarrow{\mathsf{water}} l_3 \xrightarrow{2} \xrightarrow{\mathsf{harvest}} \mathrm{succ} \xrightarrow{\infty}$$

with $\rho' \not\models \Diamond\,\mathsf{fail}$. Therefore, ϕ does not appear in ρ' and the $\mathbf{CF}_{\mathsf{BF}}$-condition is fulfilled. △

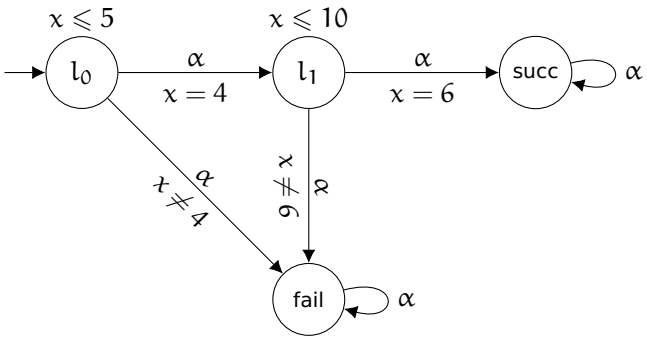
This shows that in the timestamp perspective, it is now possible to shift the time of the fertilizing from the "faulty" month of June to the "correct" month of April while the

absolute timestamps of the other actions are not shifted. Hence, we can now identify the "faulty" time of the fertilizing alone as but-for cause for the failed harvest. There are numerous other real-world examples where this timestamp perspective to causality seems to be the perspective of choice: when modeling a school bell that should ring at specified times, when simulating a supermarket in the course of the week in which product deliveries take place on fixed days, or streetlights that are to be switched on at sunset and switched off at sunrise.

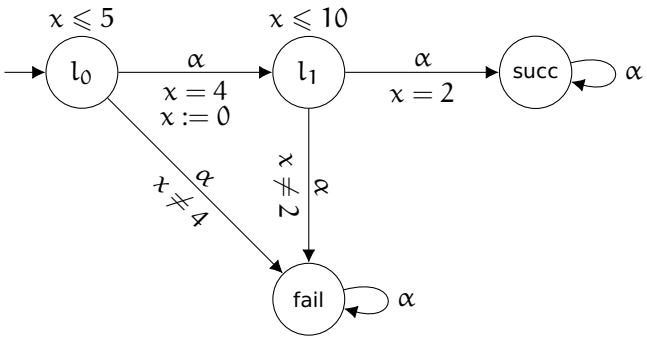
However, there are conversely also just as many examples in which the timestamp perspective is not suitable but in which the delay perspective results in an analysis that corresponds better to intuition: a cooking process in which it is important that the ingredients should be mixed in with correct temporal delays between each other, a chemical process to which energy, heat or chemicals must be added at the right intervals, or a railway crossing where traffic lights and barriers should behave in the correct time delay with respect to the crossing train.

We demonstrate how the two perspectives apply differently well in different scenarios with the following synthetic minimal example.

Example 3.2.3. We consider two timed automata: Firstly the timed automaton TA_1:



Secondly the very similar timed automaton TA_2 in which the basic difference is simply that the clock x is reset after the first transition:



We look again at the effect $\phi := \Diamond\,\mathsf{fail}$. In both automata, we have the same "successful" run $\rho' := l_0 \xrightarrow{4} \xrightarrow{\alpha} l_1 \xrightarrow{2} \xrightarrow{\alpha} \mathsf{succ}$ in which ϕ does not appear. Consider now, however,

runs in which the effect appears, so, for instance, the run

$$\rho_1 := l_0 \xrightarrow{3} \xrightarrow{\alpha} \mathsf{fail} \xrightarrow{3} \xrightarrow{\alpha} \mathsf{fail}$$

of TA_1 as well as the run

$$\rho_2 := l_0 \xrightarrow{3} \xrightarrow{\alpha} \mathsf{fail} \xrightarrow{2} \xrightarrow{\alpha} \mathsf{fail}$$

of TA_2. We observe the following: Using the delay perspective, we obtain as minimal but-for cause for ϕ in ρ_1 of TA_1 the set of events $\{(3,1),(3,2)\}$, while the timestamp perspective yields, however, the minimal but-for cause $\{(3,1)\}$. For TA_2 and ρ_2 the perspectives behave exactly conversely: With the delay perspective we get $\{(3,1)\}$ as minimal but-for cause for ϕ in ρ_2 of TA_2, with the timestamp perspective we, however, need the larger set $\{(3,1),(5,2)\}$ to obtain a minimal but-for cause. △

The difference between the two scenarios lies exactly in the question whether we are interested in the time since the start of the automaton (as in TA_1, where the clock x is not reset) or whether we are interested in the time spent in the current location (as in TA_2, where x is reset).

Remark 3.2.1. Also allowing contingencies does not solve the discussed issue in Example 3.2.1 and Example 3.2.3, that in the delay perspective the singleton sets with only one delay event are not identified as but-for cause, as they are neither actual causes. This is since by taking a contingency in a certain step, we can only fix both the clock values together with the location in which the actual run was in this step. Therefore, even though the contingency allows to set the clocks to the desired values, the effect is not avoided since the contingency sets the run to the undesired "fail" location. △

→ Example 3.2.1, p. 47
→ Example 3.2.3, p. 53

Therefore, it really depends on the particular application scenario which perspective yields the causal analysis closer to our intuition. Actually, it is sometimes already in the first place a philosophical question of the perspective whether one considers in a real-time process certain delays of actions or their timestamps to be causal for a certain effect[6].

Besides the question to what extent the two presented perspectives are in accordance with intuition, there are however further differing aspects between the perspectives that all speak in fact rather in favor of the delay perspective. First of all, the attentive reader might have noticed that we considered always only finite timestamp traces. This is due to the fact, that it is difficult (or basically even unclear at all) how to represent infinite timestamp traces suitably since defining lasso-shaped timestamp traces seems not to be possible in a proper way. Therefore, also the complete development in the timestamp perspective as well as the final notion of causality can only handle finite traces and runs.

[6] Maybe, philosophers or lawyers could come up at this point with thrilling examples in which the moral responsibility or legal debt depends on whether one takes a delay or a timestamp perspective to the scenario.

Consequently, effects that talk about the infinite behavior of runs cannot be analyzed from a timestamp perspective.

Secondly, we also have, in contrast to the delay perspective, not defined a notion of actual causality for the timestamp perspective. While it seems formally well-defined to copy the definition of actual causality, that is, to intersect the contingency automaton constructed exactly as stated in Definition 3.6 with the counterfactual trace automaton for timestamp traces, this approach leads to undesired and counter-intuitive results: In particular, this notion identifies sets of events as actual causes that should not be causes at all. This is because taking a contingency can manipulate clocks and thereby adjust the absolute time that is needed to reach for instance certain locations. We simultaneously measure, however, exactly this absolute time with clock t in the counterfactual trace automaton and seem to have no possibility to properly adjust this clock accordingly such that we obtain unfaithful runs in the intersection automaton.

→ Definition 3.6, p. 41

Lastly, we would also argue from a philosophical point of view that a delay perspective is way more in the flavor of real-time systems in general. Real-time systems are often used for reactive and non-terminating systems in which it is not really of importance how much time has passed since the system was activated but how much time has passed since certain real-time events.

All those advantages of the delay perspective in comparison to the timestamp perspective together prompted us to present the delay perspective in the previous section as our notion of choice for counterfactual causality in real-time. Nonetheless, we think that the discussion of the differing timestamp perspective yielded several theoretical as well as philosophical insights. Furthermore, it raises some interesting questions and possible future lines of research, for instance, on whether and how it might be possible to combine both perspectives in one notion. Those further ideas will be discussed in more detail in Chapter 7.

→ Chapter 7, p. 111

3.3 Remarks, Limitations, and Discussion

We want to give some further remarks on theoretical aspects surrounding our formalization and report in this course also on shortcomings of the developed notions. To this end, we start by discussing the assumptions that we have (not) to make on the real-time setting we work in.

We assume the given timed automaton to be timewise action-deterministic. Assuming timewise action-determinism is in various regards crucial for our development. Notice firstly, that Propositions 8 and 12, which state the empty set of events to never be a cause, would become false: If the automaton has a run ρ' differing from the given actual run ρ, however, with the same corresponding trace, it could well happen that $\rho \models \phi$ while $\rho' \not\models \phi$, in which case ρ' would a witnessing run for the empty set to be a cause. As also generally for causal models, we want instead that the occurrence of

→ Propositions 8, 12, p. 30 and 46

effects is fully determined by the occurring events. For our setting, which considers only events of timed traces, this consequently means, that the occurrence of effects should be determinable from a given trace. Due to Proposition 2, this can be enforced by assuming timewise action-determinism of the automaton we are working in.

→ Proposition 2, p. 17

We assume the actual run to be given in a concrete form. Having a concrete run with concrete real-time delays at hand is the starting point in our causal setting. This differs in fact from other approaches to causality in real-time systems as we will discuss in more depth in Chapter 6. Our choice was in particular guided by the targeted application to obtain very concrete explanations of an observed system behavior.

→ Chapter 6, p. 105

We assume the effect to be described in MITL. Evidently necessary for the decidability of meaningful causality notions is in general to consider decidable effects. We do so in our real-time setting by specifying effects in MITL, a decidable real-time logic. Moreover, MITL is a trace-based logic (or in the terminology of the real-time setting, a signal-based logic), that is, MITL specifies a system by describing properties of the corresponding signals of its runs. This signal-based nature of MITL turned out to be well-suitable for our causal reasoning, as they firstly transport easily through different automata operations, and matched secondly with our causal setting that considers effects occurring in certain runs of the system. On the contrary, it does not seem that our approach can be easily adapted to apply also to branching-time logics, as they specify a system by rather specifying its locations with regard to all the possible runs originating from a location.

We cover both finite as well as lasso-shaped runs. The flexibility in our formalizations and constructions to handle both finite as well as lasso-shaped runs makes our development applicable to different scenarios. Finite runs enable us, for instance, to gain explanations for (finite) bad prefixes of safety properties, supporting lasso-shaped runs extends our development to provide also explanations for infinite violations of liveness properties.

We do not require the transition relation of the timed automaton to be total. Not necessary is to require some kind of totality property for the transition relation, that is, we do not assume actions to always be or become enabled. Going further, we do not even require the automaton to be deadlock-free, and hence, allow locations without any outgoing transition at all. In practice however, the causal analysis of such systems seems to be oftentimes inappropriate as disabled actions result quickly in timelocks or deadlocks in the constructed intersections. A phenomenon also related to the in Section 3.2 discussed the difficulty of time propagation, i.e., that changes in one delay induce timing changes also in the whole remainder of the run. Consequently, guards might become unsatisfiable resulting also in a blocking behavior of the counterfactual simulation.

→ Section 3.2, p. 46

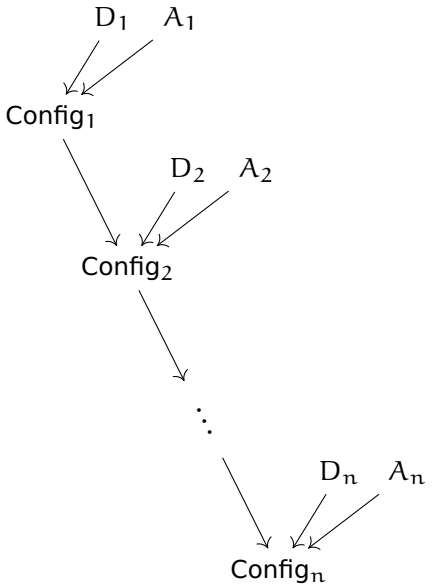
Next, we want to discuss in more detail how our developed real-time notions are related to the foundational causal work of Halpern and Pearl and justify more concretely

that our definitions are – as claimed several times – indeed in the flavor of Halpern and Pearl's causality concept [37]. We therefore argue, how our setting can be described in a causal model and that the foundational and real-time notions of causality do then widely agree. However, we do therefore not claim to provide complete formal arguments but rather focus on the intuitive connections.

Recall that Halpern and Pearl used causal models, that is, different variables and structural equations over those variables, for describing the considered parts of the world (cf. Section 2.1.1). Our considered "world" is given by the timed automaton. A causal model describing the execution of a timed automaton TA for n steps could be designed in the following way:

→ Section 2.1.1, p. 7

- Endogenous variables D_i and A_i for $i = 1, \ldots, n$ represent the delays and actions in the course of the execution. Hence, D_i range over $\mathbb{R}_{\geqslant 0}$ and A_i over Act, whereby Act is the set of actions of TA.
- Endogenous variables Config_i for $i = 0, \ldots, n$ describe the configuration of the system in step i, that is, its location and clock assignment. Hence, Config_i ranges over values of the form $\langle l, u \rangle$, whereby l are locations of TA and u a clock assignment.
- For $i = 1, \ldots, n$, the variables D_i, A_i, and Config_{i-1} determine the value of configuration Config_i. The concrete structural equation of Config_i is induced by the concrete structure of TA, that is, that the formula encodes the transitions of TA.



In the causal formulas used in the foundational setting to specify effects, we could as Boolean combinations over the primitive events $\mathsf{Config}_1 = \langle l_i, u_i \rangle$ then express certain run effects as well.

If we now restrict the endogenous variables, that are considered as events of potential causes only to the variables A_i and D_i, then the causal analysis induced by foundational but-for causality does indeed coincide with the developed real-time notion: delay and actions events $(\vec{D}, \vec{A}) = (\vec{\delta}, \vec{\alpha})$ are identified as cause for an effect, if there exists an alternative setting $(\vec{D}, \vec{A}) = (\vec{\delta}', \vec{\alpha}')$ such that the effect does not occur in the counterfactual simulation – exactly in accordance to the real-time definition of but-for causality asking for the existence of a counterfactual run if alternative delays and actions are allowed.

Even more, if we restrict the endogenous variables considered for contingencies conversely to the variables Config_i, also the notions of actual causality show large commonalities as they both allow to fix the configurations of the actual run. Our real-time notion encodes the foundational propositional definition asking for the existence of a set of contingency variables Config_i in the contingency automaton.

→ Section 3.1.3, p. 37

Notice how the in Section 3.1.3 already mentioned non-determinism of the contingency automaton is for this encoding in fact essential: If there exists a set of contingencies $\vec{W} = \{\mathsf{Config}_i \mid i \in \{i = 1, \ldots, n\}\}$ witnessing a set of events to be an actual cause, we know that it is found in the runs of the contingency automaton as they can choose in each step non-deterministically either to take or not to take a contingency. If we know conversely that there is a counterfactual run in the contingency automaton, the steps in which the run non-deterministically chose to take a contingency induce again the set $\vec{W}$ of contingencies that witness in the foundational notion the cause. In contrast, Coenen et al. [22] capture the question of the existence of contingencies not by using a non-deterministic transition relation, but by extending the set of possible inputs (of Moore machines that are considered in their setting) by auxiliary contingency variables and choose contingencies accordingly to the "non-deterministic" occurrence of those inputs.

We want to remark on an important detail in the formal modeling above: for the delays as well as for the configurations of automata (as they contain clock assignments) it is essential to allow them to have an infinite range of possible values as there are infinitely many possible alternatives to the real-time behavior of the automata. Allowing, for instance, only finitely many alternatives $\{\delta'_1, \ldots, \delta'_k\}$ to a causal delay δ would result in an incomplete causal theory in the sense that there might be causes that are not identified as such since the witnessing counterfactual relies on an alternative delay $\delta' \notin \{\delta'_1, \ldots, \delta'_k\}$. Moreover, as discussed in the previous sections, it is, also crucial for a faithful causal analysis to analyze the counterfactual runs not only up to the length of the given actual run but to look at their infinite behavior. And also in the first place, we describe the effects in timed-automata based on MITL for time-divergent, and hence, mainly infinite runs. Therefore, the above model should actually be extended by further variables Config_i with $i = n+1, n+2, \ldots$, resulting in a model even with infinitely many variables[7].

[7] Tempted by considerations of dynamical systems, Halpern and Peters extend in a recently published work in fact (finite) causal models to generalized structural equation models [42] that can handle infinitely many variables over infinite ranges.

Known decidability and complexity results of causality notions in the sense of Halpern and Pearl do, however, only consider causal models with finitely many variables ranging over a finite set of possible values [30, 7]. As further discussed in the next chapter, we could in our setting tackle both issues that endanger decidability – the necessary consideration of infinitely many counterfactual alternatives as well as the infinite simulation of the counterfactual world – through the automata-based approach to the causal reasoning: automata yield a finite representation of infinitely many traces and model-checking can analyze the infinite behavior of automata. This will yield new decidability and complexity results for deciding actual causality in a theory with possibly infinite many variables that partially range over an infinite domain.

The introduced causal model describing our real-time setting helps us also to illustrate a last aspect that we want to discuss in more detail, namely, that the contingencies, that our notion of actual causality allows to be taken, are relatively coarse-grained. Coarse-grainedness – in contrast to fine-grainedness – means in the context of contingencies, that choosing to take a single contingency (i.e., choosing to add a single variable to the set of contingencies) already fixes a relatively large part of the world. In our case, choosing to take a contingency $\mathsf{Config}_i = \langle l_i, u_i \rangle$ in step i already fixes the complete configuration of the automaton in this step. A more fine-grained notion of contingencies would in our setting, for instance, be induced by a notion that allows to fix also only the location l_i or only the clock assignment u_i without fixing the other component as well.

There might, however, be applications of actual causality in which we want to fix exactly a particular part of the world to its actual value (in our setting, for instance, to fix the clock assignment u_i) without fixing at the same time a particular other part of the world (in our setting, for instance, without fixing location l_i). Such a situation occurred in fact exactly in Example 3.2.3 as already indicated in Remark 3.2.1. We think that a more fine-grained notion of contingencies could, hence, allow to partly solve the issue of time propagation.

→ Example 3.2.3, p. 53
→ Remark 3.2.1, p. 54

Going even further, there might also occur examples in which we want to fix only a certain aspect of a clock assignment (for instance only the value of a single clock) or to fix only a certain aspect of a location (for instance in product automata with product locations only the first component of the location). Since we do not allow this in our notion of actual causality, we can in fact only solve the preemption problem in the particular cases, in which the considered effect occurs "temporarily" in the actual run (in Example 3.1.7, the actual run stays only temporarily in the "coffee"-location; in Example 3.1.9, the run stays only temporarily in the crucial location l_1). In Section 7.1, we remark on a future approach to extend our work to networks of timed automata which might also induce a more fine-grained notion of contingencies. There, also various further ideas for refining or adapting the in this chapter presented definitions of counterfactual causality in real-time systems will be discussed.

→ Example 3.1.7, p. 38
→ Example 3.1.9, p. 43
→ Section 7.1, p. 111

Chapter 4

Algorithms for Cause Checking and Computation

While we have already discussed several theoretical results regarding our developed notions of causality in the last chapter, we have not yet looked at the aspects of decidability and computability of causes. The numerous presented examples in which we checked whether a given set of events is indeed a but-for or actual cause suggest that our definitions of causality should be decidable by computational manners. In fact, we can substantiate this conjecture and will in this chapter now present algorithms for cause checking as well as for the computation of causes and prove the correctness of those algorithms. Furthermore, we take a look at the computational complexity of those tasks. We present the development for the setting of the delay perspective, all of the work applies, however, exactly analogously for the timestamp perspective.

When looking again at our definitions of causality, it is evident that algorithms checking those definitions need to apply model checking with respect to the considered run effects. As we specified our effects using MITL, Theorem 5 yields that such model-checking procedures do indeed exist. Therefore, we can assume for the whole chapter a computable model-checking function M that takes a timed automaton TA and an MITL formula ϕ, and checks whether the automaton satisfies the formula, that is,

→ Theorem 5, p. 22

$$M(\mathsf{TA}, \phi) := \begin{cases} \textbf{true}, & \text{if } \mathsf{TA} \models \phi, \text{ i.e., } \forall \rho \in \mathsf{Runs}_{\mathsf{div}}.\, \rho \models \phi, \\ \textbf{false}, & \text{otherwise.} \end{cases}$$

We will not look into the details of this model-checking process but simply apply the assumed function without further discussion in the presented algorithms.

Before heading to the algorithms themselves, we have to remark an important result regarding the conditions in our causality definitions, as the algorithms will crucially rely on this observation.

Lemma 14 (Monotonicity)**.** *For every timed automaton* TA*, run* ρ*, and effect* ϕ*, we have that*

1. *if a set of events* $\mathcal{C}$ *fulfills* ***SAT*** *also every subset* $\mathcal{C}' \subseteq \mathcal{C}$ *fulfills* ***SAT****.*
2. *if a set of events* $\mathcal{C}$ *fulfills* $\boldsymbol{CF}_{\mathsf{BF}}$ *(fulfills* $\boldsymbol{CF}_{\mathsf{Act}}$*) also every superset* $\mathcal{C}' \supseteq \mathcal{C}$ *fulfills* $\boldsymbol{CF}_{\mathsf{BF}}$ *(fulfills* $\boldsymbol{CF}_{\mathsf{Act}}$*).*

Proof. Let ξ be the corresponding trace of ρ.

1. If $\mathcal{C}$ fulfills **SAT**, we have that $\xi \models \mathcal{C}$ and $\rho \models \mathsf{E}$. Directly by the definition of event satisfaction (cf. Definition 3.2) it follows that also for every subset $\mathcal{C}' \subseteq \mathcal{C}$ we have $\xi \models \mathcal{C}$ such that also $\mathcal{C}'$ fulfills **SAT**.

→ Definition 3.2, p. 25

2. Let $\mathcal{C}$ fulfill $\mathbf{CF}_{\mathsf{BF}}$, that is, that we have a time-divergent run ρ' of $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ with $\rho' \not\models \mathsf{E}$ and let $\mathcal{C}' \supseteq \mathcal{C}$. Furthermore let $\rightarrow$ be the transition relation of $\mathsf{TA}_\xi^{\mathcal{C}}$ and let $\rightarrow'$ be the transition relation of $\mathsf{TA}_\xi^{\mathcal{C}'}$. From the defining rules of the transition relation it follows that $\rightarrow \subseteq \rightarrow'$, such that every run of $\mathsf{TA}_\xi^{\mathcal{C}}$ is also a run of $\mathsf{TA}_\xi^{\mathcal{C}'}$. Therefore, also $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}'}$ has with ρ' a time-divergent run in which ϕ does not appear such that $\mathcal{C}'$ fulfills $\mathbf{CF}_{\mathsf{BF}}$. The proof for $\mathbf{CF}_{\mathsf{Act}}$ works analogously for the time-divergent run in intersection of the contingency and counterfactual trace automata. □

Those monotonicity properties of the **SAT**-, $\mathbf{CF}_{\mathsf{BF}}$-, and $\mathbf{CF}_{\mathsf{Act}}$-condition will be of high advantage in particular with regard to the necessary computational work.

4.1 Cause Checking

We start by discussing the algorithms for checking whether a given set of events is a cause for an effect in a run of a timed automaton. As suspected and also already practiced in all the examples in the previous chapter, we will proceed with checking this cause property by checking all the respective defining conditions one after another. For the **SAT**-condition, we have to verify that firstly the corresponding trace of the given run indeed satisfies the tested events and secondly whether the effect did indeed appear in the run. While checking the event satisfaction is a straightforward task, checking whether in the particular given run the effect appears requires more effort.

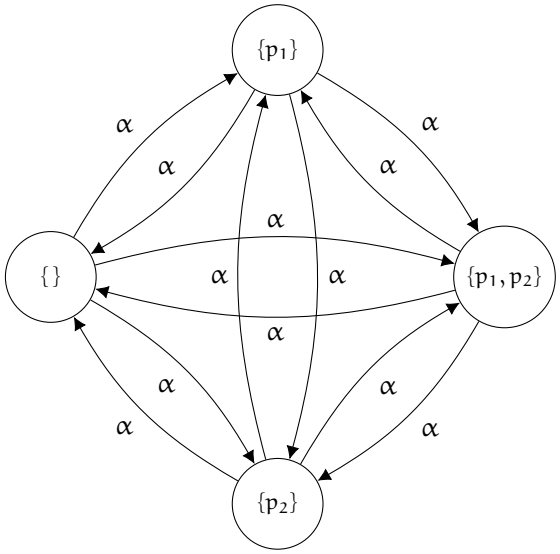
The first issue occurs as we assumed that the model-checking function M decides whether a whole system satisfies an effect (i.e., whether all its time-divergent runs satisfy the effect), and not whether a given single run does so. Secondly, for non time-divergent runs, the satisfaction of effects specified as MITL formulas is defined via the notion of good prefixes (cf. Definition 2.10). According to this, we have to check whether all time-divergent extensions of a given prefix satisfy the effect.

→ Definition 2.10, p. 21

We solve both issues together in one carefully composed construction: The key idea is to apply the model checker to an automaton in which the run on which we want to check the effect is enforced. Fortunately, this corresponds exactly to a construction we have already access to, namely the intersection of our given automaton with the counterfactual trace automaton for the empty set of events. Similar to the reasoning already in Proposition 8, we know due to the properties of the counterfactual trace automaton for empty event sets as well as due to the assumed action-determinism that the runs in the intersection automaton $\mathsf{TA} \cap \mathsf{TA}^{\emptyset}_{\xi}$ must agree with ρ (whereby ξ denotes again the corresponding trace).

→ Proposition 8, p. 30

To handle now also prefixes, we adapt the usual intersection construction particular for our purpose and allow after the enforcement of the prefix arbitrary time-divergent extensions, that is, that we concatenate the intersection with an automaton that allows for a given set of atomic propositions the signal prefix to be extended in all possible ways. For a two elementary set of atomic propositions $AP = \{p_1, p_2\}$, this extension automaton looks, for instance, as follows:



Here, α is an arbitrary element from Act of the initially given automaton. Generally speaking, this extension automaton is constructed by first taking for each element of the powerset of AP, that is, for each possible label, a unique location. All those locations are then pairwise connected via transitions. The sets written inside the locations specify the label of the location.

Formally, this whole construction described above will be called "satisfiability intersection" and is formally defined as an extension of the intersection operator (cf. Definition 2.7).

→ Definition 2.7, p. 18

Definition 4.1 (Satisfiability Intersection)

Def. satisfiability intersection automaton

Let TA be a timed automaton, ρ a finite or lasso-shaped run with corresponding trace ξ. The *satisfiability intersection automaton* $\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}_\xi^\emptyset$ of TA and ξ is defined for lasso-shaped ξ simply as the intersection

$$\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}_\xi^\emptyset := \mathsf{TA} \cap \mathsf{TA}_\xi^\emptyset.$$

For finite ξ, we let $\mathsf{TA} \cap \mathsf{TA}_\xi^\emptyset = (Loc, l_0, C, Act, \to, I, AP, L)$ and define the satisfiability intersection automaton of TA and ξ in the following way as an extension of the intersection:

$$\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}_\xi^\emptyset := (Loc \dot\cup Loc_{AP}, l_0, C, Act, \to \dot\cup \to', I', AP, L'),$$

whereby

- $Loc_{AP} := \{l_P \mid P \in \mathcal{P}(AP)\}$, where we assume those locations to not already occur in Loc,
- $\to' := \{(\langle l_n, n\rangle, \top, \alpha, \epsilon, l_P) \mid l \in Loc, l_P \in Loc_{AP}\} \cup \{(l_P, \top, \alpha, \epsilon, l_{P'}) \mid l_P, l_{P'} \in Loc_{AP}\}$, where l_n is the last location of ρ, n the length of ρ, and α some action in Act,
- $I'(l) := \begin{cases} I(l), & l \in Loc \\ \top, & l \in Loc_{AP}, \end{cases}$ and
- $L'(l) := \begin{cases} L(l), & l \in Loc \\ P, & l = l_P \in Loc_{AP}. \end{cases}$

The additional set of locations Loc_{AP} contains exactly the locations corresponding to possible labels $P \in \mathcal{P}(AP)$ allowing as desired arbitrary extensions. The additional transitions in $\to'$ handle the concatenation of the intersection automaton and the extension automaton (first set of the union) and the arbitrary traveling through the extension automaton (second set of the union).

→ Algorithm 1, p. 65

With this operation in hand we are set to define Algorithm 1 for checking the **SAT**-condition in our definitions of causality, which is depicted on the next page.

As mentioned, the algorithm simply checks one after another all the necessary properties that must be fulfilled in order for the **SAT**-condition to be fulfilled. This rigorous proceeding justifies then also its correctness.

→ Algorithm 1, p. 65

Lemma 15. *Algorithm 1 works as stated, that is, that we have*

$$\textit{Check\_SAT}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{\textit{true}} \textit{ if and only if } \mathcal{C} \textit{ fulfills } \textbf{\textit{SAT}}$$

for every timed automaton TA*, run* ρ*, effect* ϕ*, and set of events* $\mathcal{C}$*.*

Algorithm 1: Check_SAT

Input: timed automaton TA, run ρ with corresponding trace $\xi = \langle \delta_1, \alpha_1 \rangle \dots \langle \delta_n, \alpha_n \rangle$, effect ϕ, set of events $\mathcal{C}$

Output: **true**, if **SAT** is fulfilled, i.e., if $\xi \models \mathcal{C}$ and $\rho \models \phi$; **false**, otherwise

```
1  for delay event (δ, i) ∈ C do          /* checks delay event satisfaction */
2      if δ ≠ δ_i then
3          return false
4      end
5  end
6
7  for action event (α, i) ∈ C do         /* checks action event satisfaction */
8      if α ≠ α_i then
9          return false
10     end
11 end
12
13 return M(TA ∩_SAT TA^∅_ξ, φ)           /* checks effect satisfaction */
```

Proof. The for-loop from Line 1 to Line 5 checks whether we have for every delay event $(\delta, i) \in \mathcal{C}$ that $\delta = \delta_i$, the for-loop from Line 7 to Line 11 checks whether we have for every action event $(\alpha, i) \in \mathcal{C}$ that $\alpha = \alpha_i$, and Line 13 checks whether $\rho \models \phi$.

For the later property, we know again by Lemma 7 and Proposition 2 that the corresponding intersection run of ρ is the only run in $\mathsf{TA} \cap \mathsf{TA}^{\emptyset}_{\xi}$. For ρ lasso-shaped, we know the intersection run of ρ to be the only time-divergent run in $\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}^{\emptyset}_{\xi} = \mathsf{TA} \cap \mathsf{TA}^{\emptyset}_{\xi}$ and therefore

→ Lemma 7, p. 29
→ Proposition 2, p. 17

$$\rho \models \phi \Leftrightarrow \mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}^{\emptyset}_{\xi} \models \phi$$

$$\Leftrightarrow M(\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}^{\emptyset}_{\xi}, \phi) = \textbf{true}.$$

For a finite ρ, we know again by Lemma 7 and Proposition 2 together with the chosen construction of the satisfiability intersection automaton that the time-divergent runs in $\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}^{\emptyset}_{\xi} \models \phi$ are exactly all possible extensions of the intersection run of ρ. Therefore, we have again

$$\rho \models \phi \Leftrightarrow \forall\, \rho' \text{ time-divergent extension of } \rho.\ \rho' \models \phi$$

$$\Leftrightarrow \mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}^{\emptyset}_{\xi} \models \phi$$

$$\Leftrightarrow M(\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}^{\emptyset}_{\xi}, \phi) = \textbf{true}.$$

Hence, **SAT** is fulfilled if and only if Check_SAT($\mathsf{TA}, \rho, \phi, \mathcal{C}$) = **true**. □

The $\mathbf{CF}_{\mathsf{BF}}$-condition is checked simply by applying the model checker M to the appropriate automaton. Notice that all our automata constructions are computable such that they can indeed be constructed and represented by computational manners:

Algorithm 2: Check_CF_BF

Input: timed automaton TA, run ρ with corresponding trace ξ, effect ϕ, set of events $\mathcal{C}$

Output:
- **true**, if $\mathbf{CF}_{\mathsf{BF}}$ is fulfilled, i.e., if there is an time-divergent run ρ' of $\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ with $\rho' \not\models \phi$
- **false**, otherwise

1 **return** $\neg\, M(\mathsf{TA} \cap \mathsf{TA}_\xi^{\mathcal{C}},\, \phi)$

The algorithm for checking the $\mathbf{CF}_{\mathsf{Act}}$-condition works exactly analogously, except for the detail that we check now the intersection of the contingency and the counterfactual trace automaton for the existence of a time-divergent run ρ'.

Algorithm 3: Check_CF_Act

Input: timed automaton TA, run ρ with corresponding trace ξ, effect ϕ, set of events $\mathcal{C}$

Output:
- **true**, if $\mathbf{CF}_{\mathsf{Act}}$ is fulfilled, i.e., if there is an time-divergent run ρ' of $\mathsf{TA}_\rho^{\mathsf{con}} \cap \mathsf{TA}_\xi^{\mathcal{C}}$ with $\rho' \not\models \phi$
- **false**, otherwise

1 **return** $\neg\, M(\mathsf{TA}_\rho^{\mathsf{con}} \cap \mathsf{TA}_\xi^{\mathcal{C}},\, \phi)$

The correctness of those algorithms follows simply by the defining property of the model-checking function M.

Lemma 16. *Algorithms 2 and 3 work as stated, that is, that we have*

$$\textit{Check\_CF\_BF}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{\textit{true}} \textit{ if and only if } \mathcal{C} \textit{ fulfills } \boldsymbol{CF}_{\mathsf{BF}}$$

and

$$\textit{Check\_CF\_Act}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{\textit{true}} \textit{ if and only if } \mathcal{C} \textit{ fulfills } \boldsymbol{CF}_{\mathsf{Act}}$$

for every timed automaton TA*, run* ρ*, effect* ϕ*, and set of events* $\mathcal{C}$*.*

Proof. Follows with the equivalence chain

$$
\begin{aligned}
&\text{Check\_CF\_BF}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{true} \\
\Leftrightarrow\ & M(\mathsf{TA} \cap \mathsf{TA}^{\mathcal{C}}_{\xi},\ \phi) = \textbf{false} \\
\Leftrightarrow\ & \neg\forall \text{ time-divergent runs } \rho' \text{ of } \mathsf{TA} \cap \mathsf{TA}^{\mathcal{C}}_{\xi}.\ \rho' \models \phi \\
\Leftrightarrow\ & \exists \text{ time-divergent run } \rho' \text{ of } \mathsf{TA} \cap \mathsf{TA}^{\mathcal{C}}_{\xi}.\ \rho' \not\models \phi \\
\Leftrightarrow\ & \mathcal{C} \text{ fulfills } \mathbf{CF}_{\mathsf{BF}}
\end{aligned}
$$

The result for $\mathbf{CF}_{\mathsf{Act}}$ follows analogue with $\mathsf{TA}^{\mathsf{con}}_{\rho} \cap \mathsf{TA}^{\mathcal{C}}_{\xi}$ instead of $\mathsf{TA} \cap \mathsf{TA}^{\mathcal{C}}_{\xi}$. □

The last condition that needs to be checked is the minimality of a set of events $\mathcal{C}$. Thereby, we take for the first time advantage of the monotonicity observation from Lemma 14: Under the invariant that the algorithm is only used for causes fulfilling the **SAT**-condition, we know that also all its subsets will fulfill **SAT**. Furthermore, we know again by monotonicity that if there is a strict subset $\mathcal{C}' \subsetneq \mathcal{C}$ fulfilling $\mathbf{CF}_{\mathsf{BF}}$, that then also all sets between $\mathcal{C}'$ and $\mathcal{C}$ (i.e., all sets $\mathcal{C}''$ with $\mathcal{C}' \subseteq \mathcal{C}'' \subsetneq \mathcal{C}$) do so. Therefore, it suffices to check whether none of the sets with one element less than $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$:

→ Lemma 14, p. 62

Algorithm 4: Check_MIN

Input: timed automaton TA, run ρ, effect ϕ, set of events $\mathcal{C}$ fulfilling **SAT**

Output:
- **true**, if **MIN** is fulfilled, i.e., if there is no strict subset of $\mathcal{C}$ satisfying **SAT** and $\mathbf{CF}_{\mathsf{BF}}$
- **false**, otherwise

```
1 for event e ∈ C do
2     if Check_CF_BF(TA, ρ, φ, C \ {e}) then      /* checks smaller event set */
3         return false
4     end
5 end
6 return true
```

The algorithm for checking the minimality condition for actual causality, i.e., whether there is no strict subset of $\mathcal{C}$ satisfying **SAT** and $\mathbf{CF}_{\mathsf{Act}}$ works analogously, except that we call Check_CF_Act instead of Check_CF_BF in Line 2. Also the following correctness proof works exactly analogously.

Lemma 17. *Algorithm 4 works as stated, that is, that we have*

$$\textit{Check\_MIN}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{\textit{true}} \text{ if and only if } \mathcal{C} \text{ fulfills } \textbf{\textit{MIN}}$$

for every timed automaton TA*, run* ρ*, effect* ϕ*, and set of events* $\mathcal{C}$ *that fulfills* ***SAT****.*

Proof. Let $\mathcal{C}$ fulfill **SAT**, we show the result by proving the contradictory equivalence

$$\text{Check\_MIN}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{false} \Leftrightarrow \exists \mathcal{C}' \subsetneq \mathcal{C}.\, \mathcal{C}' \text{ fulfills } \textbf{SAT} \text{ and } \textbf{CF}_{\mathsf{BF}}.$$

" $\Rightarrow$ ": Let $\text{Check\_MIN}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{false}$, then we must have due to Line 2 and 3 an event $e \in \mathcal{C}$ such that $\text{Check\_CF\_BF}(\mathsf{TA}, \rho, \phi, \mathcal{C}\setminus\{e\}) = \textbf{false}$. Hence, $\mathcal{C}\setminus\{e\}$ is a strict subset of $\mathcal{C}$ that fulfilling **SAT** by Lemma 14 and $\textbf{CF}_{\mathsf{BF}}$ by Lemma 16.

→ Lemma 14, p. 62

→ Lemma 16, p. 66

" $\Leftarrow$ ": Let there be $\mathcal{C}' \subsetneq \mathcal{C}$ fulfilling **SAT** and $\textbf{CF}_{\mathsf{BF}}$. Then, there is an event $e \in \mathcal{C} \setminus \mathcal{C}'$ such that $\mathcal{C}' \subseteq \mathcal{C}\setminus\{e\} \subsetneq \mathcal{C}$. Therefore, Lemma 14 implies that $\mathcal{C}\setminus\{e\}$ also fulfills $\textbf{CF}_{\mathsf{BF}}$ which then again implies by Lemma 16 that $\text{Check\_CF\_BF}(\mathsf{TA}, \rho, \phi, \mathcal{C}\setminus\{e\}) = \textbf{false}$. Line 2 and 3 result therefore in $\text{Check\_MIN}(\mathsf{TA}, \rho, \phi, \mathcal{C} \setminus \{e\}) = \textbf{false}$. □

From those correctness results we can finally derive the decidability of all our notions of causality in real-time systems.

Theorem 18. *Given a set of events $\mathcal{C}$ it is decidable whether $\mathcal{C}$ is a but-for cause, whether $\mathcal{C}$ is a minimal but-for cause, as well as whether $\mathcal{C}$ is an actual cause for an effect ϕ in a run ρ of a timed automaton* TA*.*

Proof. Follows directly with the respective conjunctions of Algorithms 1 to 4 together with their correctness statements in Lemmas 15 to 17. □

We denote the decision problems addressed in this theorem as CC_{BF}, $\text{CC}_{\text{BF-Min}}$, and CC_{Act} respectively.

Lastly, we want to discuss the computational effort that it takes to decide instances of these decision problems. As our notions crucially rely on solving certain model-checking problems, it appears evident that cause checking is at least as hard as model checking. In fact, we will establish that cause checking is, exactly as model checking, EXPSPACE-complete (cf. Theorem 6).

→ Theorem 6, p. 22

To do so, we first show the cause-checking problems to be decidable in exponential space by analyzing the complexity of our above presented algorithms. Thereby, the by far most expensive parts in the computations are the respective model-checking problems that need to be solved.

Lemma 19. CC_{BF}, $\text{CC}_{BF\text{-}Min}$, *and* CC_{Act} *are in* EXPSPACE.

Proof. Recall Theorem 6 stating the MITL model-checking problem to be in EXPSPACE. The algorithms used to decide the cause-checking problems have to solve the following model-checking problems:

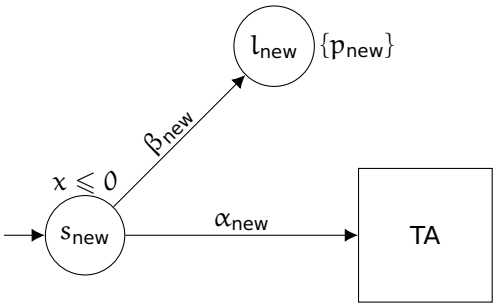
- Algorithm 1: $\text{Check\_SAT}(\mathsf{TA}, \rho, \phi, \mathcal{C})$ has to decide a model-checking problem for formula ϕ and the automaton $\mathsf{TA} \cap_{\mathsf{SAT}} \mathsf{TA}^{\emptyset}_{\xi}$, that is, an automaton with the size of TA times the length of ρ.

→ Algorithm 1, p. 65

- Algorithm 2: Check_CF_BF($\mathsf{TA}, \rho, \phi, \mathcal{C}$) has to decide a model-checking problem for formula ϕ and the automaton $\mathsf{TA} \cap \mathsf{TA}^{\mathcal{C}}_{\xi}$, that is, again an automaton with the size of TA times the length of ρ. → Algorithm 2, p. 66
- Algorithm 3: Check_CF_Act($\mathsf{TA}, \rho, \phi, \mathcal{C}$) has to decide a model-checking problem for formula ϕ and the automaton $\mathsf{TA}^{\mathsf{con}}_{\rho} \cap \mathsf{TA}^{\mathcal{C}}_{\xi}$, that is, an automaton with the size of TA times the length of ρ squared. → Algorithm 3, p. 66
- Algorithm 4: Check_MIN($\mathsf{TA}, \rho, \phi, \mathcal{C}$) in the but-for causality case calls in the worst case $|\mathcal{C}|$ times Check_CF_BF and has therefore in the worst case to decide $|\mathcal{C}|$ times a model-checking problem for formula ϕ and automata with the size of TA times the length of ρ. Analogously in the case of actual causality, $|\mathcal{C}|$ model-checking problems for formula ϕ and automata with the size of TA times the length of ρ squared need to be decided in the worst case. → Algorithm 4, p. 67

All remaining computations in the algorithms are clearly dominated by the complexity of solving the model-checking problem. Hence, this analysis shows all the cause-checking problems $\mathrm{CC}_{\mathrm{BF}}$, $\mathrm{CC}_{\mathrm{BF\text{-}Min}}$, and $\mathrm{CC}_{\mathrm{Act}}$ to be in EXPSPACE. □

For showing EXPSPACE-hardness of cause checking, we will establish a reduction from the model checking to the cause-checking problem, or more precisely a reduction from the complement of the model-checking problem[1]. For this end, we need a further automaton construction: Given a timed automaton TA, we add a new dummy initial location s_{new} and a further fresh location l_{new} with unique fresh label $\{p_{\mathsf{new}}\}$. l_{new} and the original initial location l_0 of TA are reachable from s_{new} via two fresh actions α_{new} and β_{new} respectively, furthermore, we do not allow the automaton to spend time in its new dummy location:



Formally, we define for a timed automaton $\mathsf{TA} = (\mathrm{Loc}, l_0, C, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$ its reduction automaton as

$$\mathsf{TA}_{\mathsf{red}} := (\mathrm{Loc} \dot\cup \{s_{\mathsf{new}}, l_{\mathsf{new}}\},\ s_{\mathsf{new}},\ C \dot\cup \{x_{\mathsf{new}}\},\ \mathrm{Act} \dot\cup \{\alpha_{\mathsf{new}}, \beta_{\mathsf{new}}\},\ \rightarrow',\ I',\ \mathrm{AP} \dot\cup \{p_{\mathsf{new}}\},\ L'),$$

[1] Deriving hardness from this reduction will still be a valid inference as EXPSPACE is closed under complementation.

whereby s_{new} and l_{new}/α_{new} and β_{new} are fresh locations/actions, x_{new}/p_{new} is a fresh clock/proposition, and

- $\rightarrow' := \rightarrow \cup \{(s_{new}, \top, \alpha_{new}, \epsilon, l_{new}), (s_{new}, \top, \beta_{new}, \epsilon, l_0)\}$,
- $I'(l) := \begin{cases} I(l), & l \in Loc, \\ x_{new} \leqslant 0, & l = s_{new}, \\ \top, & l = l_{new}, \end{cases}$
- $L'(l) := \begin{cases} L(l), & l \in Loc, \\ \{\}, & l = s_{new}, \\ \{p_{new}\}, & l = l_{new}. \end{cases}$

The idea of the reduction is now, that the original automaton TA does not satisfy a specification ϕ, that is, that it has a run without appearing effect ϕ if and only if taking α_{new} as first action in the reduction automaton is a cause for a slightly modified effect.

Lemma 20. CC_{BF}, $CC_{BF\text{-}Min}$, *and* CC_{Act} *are* EXPSPACE*-hard.*

Proof. Theorem 6 also stated the MITL model-checking problem to be EXPSPACE-hard. EXPSPACE-hardness of cause checking follows now by a reduction from the complement of the model-checking problem $\overline{MC_{MITL}}$ via the following reduction function r mapping instances (TA, ϕ) of the model-checking problem to instances of the cause-checking problems:

$$r : (\mathsf{TA}, \phi) \mapsto (\mathsf{TA}_{red},\ \rho_{red},\ \phi_{red},\ \mathcal{C}_{red})$$

with $\rho_{red} := s_{new} \xrightarrow{0} \xrightarrow{\alpha_{new}} l_{new}$, $\phi_{red} := \phi \vee \Diamond\, p_{new}$, and $\mathcal{C}_{red} := \{((\alpha_{new}, 1)\}$.

In fact, the reduction via r works as desired and reduces $\overline{MC_{MITL}}$ to CC_{BF}: Let $(\mathsf{TA}, \phi) \in \overline{MC_{MITL}}$. Since we have $\xi \models \mathcal{C}_{red}$, whereby ξ is the corresponding trace of ρ_{red}, and $\rho_{red} \models \phi_{red}$ the instance $r(\mathsf{TA}, \phi)$ fulfills the **SAT**-condition. From $\mathsf{TA} \not\models \phi$, we furthermore obtain a time-divergent run ρ' of TA with $\rho' \not\models \phi$ such that $s_{new} \xrightarrow{0} \xrightarrow{\beta_{new}} \rho'$ is a run of TA_{red} in which ϕ does also not appear. Hence, there is as well a time-divergent run in $\mathsf{TA}_{red} \cap \mathsf{TA}_{\xi}^{\mathcal{C}_{red}}$ without appearing effect ρ_{red} such that the $\mathbf{CF}_{BF}$-condition is fulfilled and $r(\mathsf{TA}, \phi) \in CC_{BF}$. Since $\mathcal{C}_{red}$ is singleton, we have $r(\mathsf{TA}, \phi) \in CC_{BF\text{-}Min}$ and $r(\mathsf{TA}, \phi) \in CC_{Act}$ as well. For $(\mathsf{TA}, \phi) \notin \overline{MC_{MITL}}$, we have $\mathsf{TA} \models \phi$ and, hence, $\mathsf{TA}_{red} \models \phi_{red}$ such that $\mathbf{CF}_{BF}$ is not fulfilled as there is no counterfactual run without appearing effect ρ_{red}. Therefore, $r(\mathsf{TA}, \phi) \notin CC_{BF}$ and $r(\mathsf{TA}, \phi) \notin CC_{BF\text{-}Min}$. Since the only contingency transition added in $\mathsf{TA}_{red_{\rho_{red}}}^{con}$ leads to l_{new}, we also have $\mathsf{TA}_{red_{\rho_{red}}}^{con} \models \phi_{red}$ and the same argument yields $r(\mathsf{TA}, \phi) \notin CC_{Act}$. □

This lets us conclude the EXPSPACE-completeness of the cause-checking problem.

Theorem 21. CC_{BF}, $\mathrm{CC}_{BF\text{-}Min}$, *and* CC_{Act} *are* EXPSPACE*-complete.*

Proof. By Lemmas 19 and 20. □

4.2 Cause Computation

We head to the question of how to compute causes for a given effect in a run of a timed automaton. Thereby we aim in most cases at computing all minimal but-for causes or respectively all actual causes (that are already by definition minimal). We will first present briefly a naive approach to this task but will improve on this in various ways to end up with a more involved algorithmic procedure, that we prove to be correct and analyze again its complexity. We thereby concentrate on but-for causality, actual causes are again computed exactly analogously by checking $\mathbf{CF}_{\mathsf{Act}}$ instead of $\mathbf{CF}_{\mathsf{BF}}$.

As only the events that really occur in a run might contribute to the occurrence of an effect it suffices to focus for the computation of causes only on those events. Also other events than the ones occurring in the given run cannot be part of a cause as this would violate the **SAT**-condition. As there are in a finite or lasso-shaped run only finitely many events, also the number of potential causes that can be built out of these events is finite – an observation already suggesting that the sets of events that form causes should be computable.

And indeed, a naive approach (basically a brute-forcing approach) that enumerates all the possible sets of events, checks whether they are a cause, and computes then in a second step the minimal elements from all identified causes has the desired output. For the notion of but-for causality, the algorithm computing all causes (also non-minimal) looks as follows:

Algorithm 5: Compute_But-For_Causes

Input: timed automaton TA, run ρ with corresponding trace ξ, effect ϕ
Output: set of all but-for causes for ϕ in ρ of TA

```
1  if ¬ M(TA ∩_SAT TA^∅_ξ, φ) then          /* checks effect sattisfaction */
2  |  return {}
3  end
4
5  Res := {}
6  for set of events C ∈ P(C_ξ) do          /* enumerates the set of events */
7  |  if Check_CF_BF(TA, ρ, φ, C) then
8  |  |  Res := Res ∪ {C}
9  |  end
10 end
11 return Res
```

→ Algorithm 1, p. 65

As it is not a priori clear, that the considered effect does indeed appear in the given run, the algorithm checks as done in Algorithm 1 firstly this prerequisite for having causes at all. If this is the case, the algorithm considers all possible subsets of the set of all events $\mathcal{C}_\xi$, checks whether they fulfill the $\mathbf{CF}_{\mathsf{BF}}$-condition, which then suffices to know that the set of events is a but-for cause.

To obtain the minimal causes, we can then simply filter for the minimal elements:

Algorithm 6: Compute_Minimal_But-For_Causes (Naive)

Input: timed automaton TA, run ρ, effect ϕ
Output: set of all minimal but-for causes for ϕ in ρ of TA

```
1 BF_Causes := Compute_But-For_Causes(TA, ρ, φ)
2 Res := {}
3 for set of events C ∈ BF_Causes do          /* enumerates but-for causes */
4 |   if ¬∃ C' ∈ BF_Causes. C' ⊊ C then
5 |   |   Res := Res ∪ {C}
6 |   end
7 end
8 return Res
```

While it is easy to see that this method works as intended, the computational effort to obtain all the minimal causes is for this naive approach enormously large: already for the computations in Algorithm 5 one has to consider all elements of the powerset $\mathcal{P}(\mathcal{C}_\xi)$. For each of those 2^{2n} many sets (whereby n is the length of the given run), one has then to do the expensive check of the $\mathbf{CF}_{\mathsf{BF}}$-condition. And also Algorithm 6 iterates once again over a potentially large part of the powerset.

We want to improve on this time-consuming naive algorithm by taking use of two key ideas:

- Combine the computation sets fulfilling the $\mathbf{CF}_{\mathsf{BF}}$-condition ($\mathbf{CF}_{\mathsf{Act}}$-condition) directly with the filtering for minimal causes in order to iterate only once over the powerset.
- Start with checking small causes in order to exclude in the case of a found cause already all its supersets by exploiting monotonicity.

We realize these ideas in the following way: Instead of computing the full powerset $\mathcal{P}(\mathcal{C}_\xi)$ right at the start, we dynamically build the set in stages and interweave the computation of causes in this process. For this step-by-step construction of the powerset, we use an algorithmic idea from functional programming as powersets can be constructed using the following recursive pattern:

$$\mathcal{P}(\emptyset) := \{\emptyset\}$$

$$\mathcal{P}(\{e\} \dot\cup \mathcal{C}) := \mathcal{P}(\mathcal{C}) \cup \{\{e\} \dot\cup P \mid P \in \mathcal{P}(\mathcal{C})\}$$

Thereby, the idea in the step case $\mathcal{P}(\{e\} \dot{\cup} S)$ is to first compute the powerset $\mathcal{P}(S)$ of the one element smaller set S, that is, all sets in the powerset not containing e, and add then the fresh element e to the sets in $\mathcal{P}(S)$ to obtain all sets in the powerset containing e. Accordingly, the powerset construction for an exemplary set $S = \{e_1, e_2, e_3\}$ with three elements proceeds like this:

Step 0: $\{\emptyset\}$

Step 1: $\{\emptyset, \{e_1\}\}$

Step 2: $\{\emptyset, \{e_1\}, \{e_2\}, \{e_1, e_2\}\}$

Step 3: $\{\emptyset, \{e_1\}, \{e_2\}, \{e_1, e_2\}, \{e_3\}, \{e_1, e_3\}, \{e_2, e_3\}, \{e_1, e_2, e_3\}\}$

We observe how the sets of the powerset from the previous step are used to build larger sets, i.e., supersets, belonging to the powerset. If we think now again about the occurring elements e as events in the sense of our causal setting and, consequently, about the sets $\mathcal{C}$ of the powerset as sets of events for which we want to check whether they are minimal causes for some effect, we notice how to take advantage of this observation: if a set of events is identified as cause, we do not have to consider its supersets anymore since the supersets cannot be a minimal cause. Hence, it suffices to continue the construction of the powerset only with the elements that are not identified as cause. That is, that it suffices to construct a partial powerset, which will save us in practice a significantly amount of computational work.

And also a closely related further observation regarding this construction technique will be of large use for us: we see that if a set $\mathcal{C}$ is added to the partial powerset, all the sets added afterwards will be no subsets of $\mathcal{C}$. Or conversely stated: when considering a set of events $\mathcal{C}$ its subsets were already considered previously in the computation. Heading back to our goal of computing causes, this means that if we identify a set of events $\mathcal{C}$ to fulfill **SAT** and $\mathbf{CF}_{\mathsf{BF}}$, we know that it suffices to ensure that none of the sets considered up to this point in the computation is a strict subset fulfilling **SAT** and $\mathbf{CF}_{\mathsf{BF}}$in order to conclude that $\mathcal{C}$ is a minimal cause.

According to the first observation, we know, however, also that sets from previous steps that have been identified as a cause are no longer used to build supersets. Hence sets from previous steps will be no subsets fulfilling **SAT** and $\mathbf{CF}_{\mathsf{BF}}$. All in all, it therefore even suffices to check only the sets that were considered so far in the current step of the partial powerset construction.

As in those considerations the order of the sets in which they are processed throughout the computation plays a crucial role and will be part of essential invariants ensuring the correctness of the computation, we will collect the sets from the partial powerset construction not as a set but as a list. This enables us also to refer in the correctness proof to the positions of certain sets in the partial powerset construction. The described

procedure finally results in the following improved algorithm for the computation of minimal causes:

Algorithm 7: Compute_Minimal_But-For_Causes

Input: timed automaton TA, run ρ with corresponding trace ξ, effect ϕ
Output: set of all minimal but-for causes for ϕ in ρ of TA

```
1  if ¬ M(TA ∩_SAT TA^∅_ξ, φ) then                    /* checks effect sattisfaction */
2  |  return {}
3  end
4
5  Res := {}
6  Power := [∅]
7
8  for event e ∈ C_ξ do                              /* enumerates eligible events */
9  |  Cur_Res := {}
10 |  for cause C ∈ [{e} ∪̇ P | P ∈ Power] do        /* next step of the powerset */
11 |  |  if ∃ C' ∈ Cur_Res. C' ⊊ C then             /* smaller minimal cause? */
12 |  |  |  continue
13 |  |  end
14 |  |  if Check_CF_BF(TA, ρ, φ, C) then                    /* cause found? */
15 |  |  |  Cur_Res := Cur_Res ∪ {C}
16 |  |  else
17 |  |  |  Power := Power @ [C]
18 |  |  end
19 |  end
20 |  Res := Res ∪ Cur_Res ;      /* this step's causes to overall result */
21 end
22
23 return Res
```

The algorithm again starts in Line 1 to check whether the given run fulfills the effect. After initializing the set $\mathtt{Res}$ storing the result, i.e., the identified minimal causes, and the list $\mathtt{Power}$ in which we will build the partial power set, we start by processing the events $e \in \mathcal{C}_\xi$ step by step. We will refer to this loop from Line 8 to Line 21 as outer loop and to the events successively processed in this loop as outer loop events. As mentioned, we will only have to check for strict subsets from the current step, which we will store for this purpose temporarily in the separate set $\mathtt{Cur\_Res}$. At the end of each step in Line 20, those identified causes are then added to the overall result.

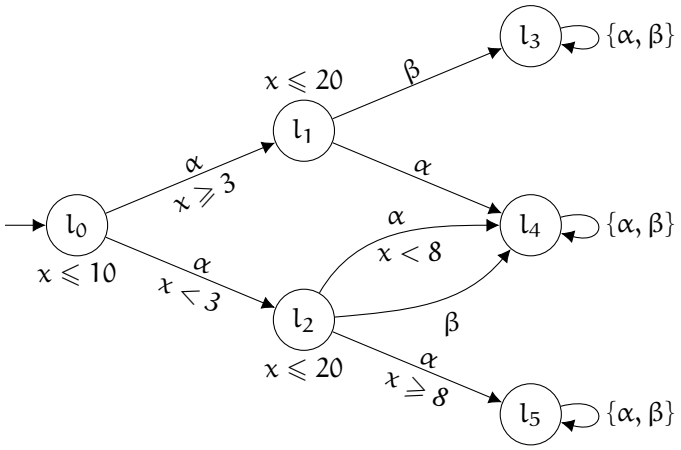
Following the above discussed method to construct powersets, we build in each step the new sets $[\{e\} \dot\cup P \mid P \in \mathtt{Power}]$ containing the fresh event e and check in the inner loop from Line 10 to Line 19 whether they are a minimal cause: Therefore, we first check

for such a set of events $\mathcal{C}$ whether there exists a strict subset from the current step that was already identified as cause (cf. Line 11). If so, $\mathcal{C}$ is no minimal cause and also its supersets can be excluded to be a minimal cause, such that we simply continue.

If there is no such strict subset, we check whether $\mathcal{C}$ fulfills the $\mathbf{CF}_{\mathsf{BF}}$-condition using Algorithm 2. If this is the case, we can directly conclude by all our previous considerations, that $\mathcal{C}$ is indeed a minimal cause. Otherwise, $\mathcal{C}$ is added to partial powerset Power such that its supersets will be considered in the following steps of the outer loop.

We demonstrate the computation procedure of the algorithm on an exemplary execution study.

Example 4.2.1. We consider the following timed automaton TA:



For the effect $\phi := \Diamond\, l_4$ in run $\rho := l_0 \xrightarrow{2} \xrightarrow{\alpha} l_1 \xrightarrow{5} \xrightarrow{\alpha} l_4$, there are (from the delay perspective) two minimal but-for causes, namely $\{(5,2)\}$ and $\{(2,1),(\alpha,2)\}$. Algorithm 7 proceeds as follows to compute those causes:

As we have $\rho \models \phi$, the events of $\mathcal{C}_\xi = \{(2,1),(5,2),(\alpha,1),(\alpha,2)\}$, with ξ the corresponding trace of ρ, are started to be processed step by step, where we let the order in which the events are processed be the order in which they are listed in the set $\mathcal{C}_\xi$:

Step 0: Res and Power are initialized:
$\Rightarrow$ Res $= \{\,\}$
Power $= [\emptyset]$

Step 1: Event $(2,1)$ is processed, the following new set of events constructed:
– $\{(2,1)\}$: violates $\mathbf{CF}_{\mathsf{BF}} \Rightarrow \{(2,1)\}$ is added to Power
$\Rightarrow$ Res $= \{\,\}$
Power $= [\emptyset, \{(2,1)\}]$

Step 2: Event $(5,2)$ is processed, the following new sets of events constructed:

- $\{(5,2)\}$: fulfills $\mathbf{CF}_{\mathsf{BF}} \Rightarrow \{(5,2)\}$ is added to $\mathtt{Cur\_Res}$
- $\{(2,1),(5,2)\}$: has a subset in $\mathtt{Cur\_Res} \Rightarrow$ continue

$\Rightarrow \mathtt{Res} = \{\{(5,2)\}\}$
$\mathtt{Power} = \big[\emptyset, \{(2,1)\}\big]$

Step 3: Event $(\alpha,1)$ is processed, the following new sets of events constructed:

- $\{(\alpha,1)\}$: violates $\mathbf{CF}_{\mathsf{BF}} \Rightarrow \{(\alpha,1)\}$ is added to $\mathtt{Power}$
- $\{(2,1),(\alpha,1)\}$: violates $\mathbf{CF}_{\mathsf{BF}} \Rightarrow \{(2,1),(\alpha,1)\}$ is added to $\mathtt{Power}$

$\Rightarrow \mathtt{Res} = \{\{(5,2)\}\}$
$\mathtt{Power} = \big[\emptyset, \{(2,1)\}, \{(\alpha,1)\}, \{(2,1),(\alpha,1)\}\big]$

Step 4: Event $(\alpha,2)$ is processed, the following new sets of events constructed:

- $\{(\alpha,2)\}$: violates $\mathbf{CF}_{\mathsf{BF}} \Rightarrow \{(\alpha,2)\}$ is added to $\mathtt{Power}$
- $\{(2,1),(\alpha,2)\}$: fulfills $\mathbf{CF}_{\mathsf{BF}} \Rightarrow \{(2,1),(\alpha,2)\}$ is added to $\mathtt{Cur\_Res}$
- $\{(\alpha,1),(\alpha,2)\}$: violates $\mathbf{CF}_{\mathsf{BF}} \Rightarrow \{(\alpha,1),(\alpha,2)\}$ is added to $\mathtt{Power}$
- $\{(2,1),(\alpha,1),(\alpha,2)\}$: has a subset in $\mathtt{Cur\_Res} \Rightarrow$ continue

$\Rightarrow \mathtt{Res} = \{\{(5,2)\},\{(2,1),(\alpha,2)\}\}$
$\mathtt{Power} = \big[\emptyset, \{(2,1)\}, \{(\alpha,1)\}, \{(2,1),(\alpha,1)\}, \{(\alpha,2)\}, \{(\alpha,1),(\alpha,2)\}\big]$

The final result $\mathtt{Res} = \{\{(5,2)\},\{(2,1),(\alpha,2)\}\}$ is returned. △

The example shows how all our considerations from above play out: In Step 2, we identify $\{(5,2)\}$ as a cause and, hence, do not consider its supersets in the following steps anymore. Conversely, we know without further computations at the points were we identify $\{(5,2)\}$ and $\{(2,1),(\alpha,2)\}$ as causes that they are minimal. Notice, however, the necessity to check for the occurring sets of events whether there was a strict subset identified as cause previously in the current step: Omitting this check would lead to classifying $\{(2,1),(5,2)\}$ in Step 2 and $\{(2,1),(\alpha,1),(\alpha,2)\}$ in Step 4 mistakenly as minimal causes.

→ Algorithm 7, p. 74

While we have explained and justified in detail the correctness of the algorithm intuitively, we now also want to prove formally that Algorithm 7 computes indeed all minimal causes. This will require a certain technical effort.

Therefore, we fix for now the given run ρ with its corresponding trace ξ and let $e_1, e_2, \ldots, e_l \in \mathcal{C}_\xi$ be the order in which the outer loop events are processed. Regarding this order, we then denote for a set of events $\mathcal{C} \subseteq \mathcal{C}_\xi$ the index of its largest event as $\max(\mathcal{C})$, i.e.,

$$\max(\mathcal{C}) := \begin{cases} \max\{k \in \mathbb{N} \mid e_k \in \mathcal{C}\}, & \text{if } \mathcal{C} \neq \emptyset \\ 0, & \text{if } \mathcal{C} = \emptyset. \end{cases}$$

With regard to this order, we furthermore introduce notations for the sets Cur_Res and Res as well as for the list Power at certain points in the execution of the algorithm, more precisely for the values of those sets and lists at the end of each event loop pass after executing Line 20. Hence, we denote

$$\begin{aligned} \text{Cur\_Res}_k &:= \text{Cur\_Res} \quad \text{after Line 20 with current outer loop event } e_k, \quad k = 1, \ldots, l \\ \\ \text{Res}_0 &:= \emptyset \\ \text{Res}_k &:= \text{Res} \quad \text{after Line 20 with current outer loop event } e_k, \quad k = 1, \ldots, l \\ \\ \text{Power}_0 &:= [\emptyset] \\ \text{Power}_k &:= \text{Power} \quad \text{after Line 20 with current outer loop event } e_k, \quad k = 1, \ldots, l \end{aligned}$$

In order to prove the correctness of the algorithm computing minimal but-for causes, we will have to show various and strong invariants of those sets and lists. In particular, we state and prove formally that Res, Cur_Res, and Power contain indeed exactly those sets of events we intuitively described above. We start the formal proof, however, with two simple properties directly following from the construction of the algorithm.

Lemma 22. *1. For each* $k = 1, \ldots, l$, *we have* $\text{Res}_k = \bigcup_{i=1}^{k} \text{Cur\_Res}_i$.

2. For each $k = 1, \ldots, l-1$, *there is a sublist* $L \subseteq [\{e_{k+1}\} \dot\cup P \mid P \in \text{Power}_k]$ *such that* $\text{Power}_{k+1} = \text{Power}_k \mathbin{@} L$.

Proof. 1. Follows easily by induction on k with regard to Line 20, the only point at which Res is changed.

2. Follows with the sublist L of $[\{e\} \dot\cup P \mid P \in \text{Power}_k]$ that contains exactly the sets of events $\mathcal{C}$ not filtered out by Line 11 and 14 and that are, hence, added in Line 17 to Power. □

As discussed, the order in which the sets of events are processed plays a crucial role for the correctness of the algorithm. We state formally the above mentioned property of the partial powerset that subsets always occur only at lower positions.

Lemma 23. *For* $k = 1, \ldots, l$, *we have* $\text{Power}_k[j] \subsetneq \text{Power}_k[i] \Rightarrow j < i$.

Proof. We prove the contraposition

$$j \geqslant i \Rightarrow \neg(\text{Power}_k[j] \subsetneq \text{Power}_k[i])$$

again by induction on k. For $k = 0$ we have due to $\mathtt{Power}_0 = [\emptyset]$ nothing to prove. In the induction step $k \rightsquigarrow k+1$, let $j \geqslant i$ and we have to show $\neg\mathtt{Power}_{k+1}[j] \subsetneq \mathtt{Power}_{k+1}[i]$. For $j = i$ the claim follows easily, such that we can even assume $j > i$. By using the equation
→ Lemma 22, p. 77
$\mathtt{Power}_{k+1} = \mathtt{Power}_k \mathbin{@} L$ with $L \subseteq [\{e_{k+1}\} \dot\cup P \mid P \in \mathtt{Power}_k]$ from Lemma 22, we obtain three possible cases:

1. $\mathtt{Power}_{k+1}[j], \mathtt{Power}_{k+1}[i] \in \mathtt{Power}_k$: The claim follows by the induction hypothesis.

2. $\mathtt{Power}_{k+1}[j] \in L, \mathtt{Power}_{k+1}[i] \in \mathtt{Power}_k$: Since $L \subseteq [\{e_{k+1}\} \dot\cup P \mid P \in \mathtt{Power}_k]$, we have $e_{k+1} \in \mathtt{Power}_{k+1}[j]$ by induction, however, $e_{k+1} \notin \mathtt{Power}_k$, such that $\neg\mathtt{Power}_{k+1}[j] \subsetneq \mathtt{Power}_{k+1}[i]$.

3. $\mathtt{Power}_{k+1}[j], \mathtt{Power}_{k+1}[i] \in L$: Then we can write the sets as $\mathtt{Power}_{k+1}[j] = \mathtt{Power}_k[j'] \cup \{e_{k+1}\}$ as well as $\mathtt{Power}_{k+1}[i] = \mathtt{Power}_k[i'] \cup \{e_{k+1}\}$ with indices $j' > i'$. By induction we then obtain $\neg\mathtt{Power}_k[j'] \subsetneq \mathtt{Power}_k[i']$ such that also $\neg\mathtt{Power}_{k+1}[j] \subsetneq \mathtt{Power}_{k+1}[i]$ holds. □

Those results allow us to prove the necessary strong invariants regarding the sets of events in $\mathtt{Cur\_Res}$ and $\mathtt{Power}$: $\mathtt{Cur\_Res}$ contains at the end of each step of the outer loop with loop event e_k exactly those sets of events that are a minimal cause and have as maximal element e_k. $\mathtt{Power}_k$ contains exactly those sets of events with maximal element at most e_k that have no minimal cause as a subset and that have hence no subset in $\mathtt{Res}_k$.

Lemma 24. *For* $k = 1, \ldots, l$*, we have*

$$\mathtt{Cur\_Res}_k = \{\mathcal{C} \mid \max(\mathcal{C}) = k \wedge \mathcal{C} \textit{ is minimal but-for cause}\}$$

and

$$\mathtt{Power}_k = \{\mathcal{C} \mid \max(\mathcal{C}) \leqslant k \wedge \neg\exists \mathcal{C}' \subseteq \mathcal{C}.\ \mathcal{C}' \in \mathtt{Res}_k\}.$$

Proof. We prove the two equations together by complete induction on k:

$k = 0$: Since we have $\max(\mathcal{C}) = 0 \Leftrightarrow \mathcal{C} = \emptyset$ and know $\emptyset$ to be never a but-for cause by
→ Proposition 8, p. 30
Proposition 8, we have

$$\mathtt{Cur\_Res}_0 \overset{\text{Def.}}{=} \emptyset = \{\mathcal{C} \mid \max(\mathcal{C}) = 0 \wedge \mathcal{C} \text{ is minimal but-for cause}\}$$

as well as

$$\mathtt{Power}_0 \overset{\text{Def.}}{=} [\emptyset] = \{\mathcal{C} \mid \max(\mathcal{C}) \leqslant 0 \wedge \neg\exists \mathcal{C}' \subseteq \mathcal{C}.\ \mathcal{C}' \in \mathtt{Res}_0\}.$$

$k+1$: By the inductive hypothesis, we can assume the claim to hold for all $r < k+1$ and start to prove the set equality

$$\mathtt{Cur\_Res}_{k+1} = \{\mathcal{C} \mid \max(\mathcal{C}) = k+1 \wedge \mathcal{C} \text{ is minimal but-for cause}\}$$

by proving the inclusions separately:

"$\subseteq$": Let $\mathcal{C} \in \text{Cur\_Res}_{k+1}$, that is, by construction of the algorithm, that $\mathcal{C} = \text{Power}_k[i] \cup \{e_{k+1}\}$ for some index i, that $\neg\exists\, \mathcal{C}' \in \text{Cur\_Res}_{k+1,\mathcal{C}}.\, \mathcal{C}' \subseteq \mathcal{C}$ whereby $\text{Cur\_Res}_{k+1,\mathcal{C}}$ denotes the set before the inner loop run in which $\mathcal{C}$ is added to Cur_Res_{k+1} and that $\text{Check\_CF\_BF}(\mathsf{TA}, \rho, \phi, \mathcal{C}) = \textbf{true}$. Thereby, we can immediately conclude that $\max(\mathcal{C}) = k+1$ since we know by induction that $\max(\text{Power}_k[i]) \leqslant k$, that **SAT** is fulfilled since we only consider events from $\mathcal{C}_\xi$ at all, and that $\mathbf{CF}_{\mathsf{BF}}$is fulfilled. It remains to show that also MIN is fulfilled. Towards a contradiction assume that $\mathcal{C}$ is not minimal, that is, that there is a minimal but-for cause $\mathcal{C}' \subsetneq \mathcal{C}$. We consider two cases: If $e_{k+1} \notin \mathcal{C}'$, we have $\mathcal{C}' \subseteq \text{Power}_k[i]$ and therefore $\max(\mathcal{C}') = r < k+1$ for some r. Hence by induction $\mathcal{C}' \in \text{Cur\_Res}_r \subseteq \text{Res}_k$, a contradiction towards $\mathcal{C}' \subseteq \text{Power}_k[i]$ since we know again by induction that there is no such subset for elements of Power_k. If however $e_{k+1} \in \mathcal{C}'$, then again by induction and since $\mathcal{C}'$ is minimal we have $\mathcal{C}' = \text{Power}_k[j] \cup \{e_{k+1}\}$ for some index j. $\mathcal{C}' \subsetneq \mathcal{C}$ then implies also $\text{Power}_k[j] \subsetneq \text{Power}_k[i]$ and by Lemma 23 we obtain $j < i$. Therefore, we know that $\mathcal{C}'$ was considered in an earlier inner loop run then $\mathcal{C}$ and thereby added to Cur_Res_{k+1}, such that due to Line 11 $\mathcal{C} \notin \text{Cur\_Res}_{k+1}$. Again a contradiction, such that **MIN** is fulfilled and $\mathcal{C} \in \{\mathcal{C} \mid \max(\mathcal{C}) = k+1 \wedge \mathcal{C} \text{ is minimal but-for cause}\}$.

→ Lemma 23, p. 77

"$\supseteq$": Let now $\mathcal{C}$ be a minimal but-for cause with $\max(\mathcal{C}) = k+1$ such that $\mathcal{C} = \mathcal{C}' \cup \{e_{k+1}\}$ for some set of events $\mathcal{C}'$ with $\max(\mathcal{C}') \leqslant k$. Since furthermore $\mathcal{C}'$ has no subset that is a minimal but-for cause and, hence, by induction no subset in Res_k, we have again by induction $\mathcal{C}' \in \text{Power}_k$. Hence, $\mathcal{C}$ is considered in the inner loop and indeed also added to Cur_Res since $\mathcal{C}$ fulfills a minimal but-for cause $\mathbf{CF}_{\mathsf{BF}}$but has no subset that is a minimal but-for cause, hence, no subset that fulfills $\mathbf{CF}_{\mathsf{BF}}$and, hence, by construction of the algorithm no subset that was added to Cur_Res such that it is not filtered out by Line 11.

Also the second equation

$$\text{Power}_{k+1} = \{\mathcal{C} \mid \max(\mathcal{C}) \leqslant k+1 \wedge \neg\exists \mathcal{C}' \subseteq \mathcal{C}.\, \mathcal{C}' \in \text{Res}_{k+1}\}$$

is shown by proving both inclusions:

"$\subseteq$": For $\mathcal{C} \in \text{Power}_{k+1}$, by Lemma 22 and induction again $\max(\mathcal{C}) \leqslant k+1$. Furthermore, we know that due to Line 14 $\mathcal{C}$ does not fulfill $\mathbf{CF}_{\mathsf{BF}}$. By monotonicity also no subset of $\mathcal{C}$ does so and, hence, no subset of $\mathcal{C}$ is a minimal but-for cause. Hence by the already proven equation, no subset of $\mathcal{C}$ is in Res_{k+1} such that we have $\mathcal{C} \in \{\mathcal{C} \mid \max(\mathcal{C}) \leqslant k+1 \wedge \neg\exists \mathcal{C}' \subseteq \mathcal{C}.\, \mathcal{C}' \in \text{Res}_{k+1}\}$.

→ Lemma 22, p. 77

"$\supseteq$": Now let $\mathcal{C}$ be such that $\max(\mathcal{C}) \leqslant k+1$ and that $\neg\exists \mathcal{C}' \subseteq \mathcal{C}.\, \mathcal{C}' \in \text{Res}_{k+1}$. Therefore, $\mathcal{C}$ does not fulfill the $\mathbf{CF}_{\mathsf{BF}}$-condition, since otherwise $\mathcal{C}$ would have a minimal but-for cause as subset and, hence, there would exists $\mathcal{C}' \subseteq \mathcal{C}$ with

$\mathcal{C}' \in \mathrm{Res}_{k+1}$. Now we distinguish again two cases: First if $e_{k+1} \in \mathcal{C}$, we can write $\mathcal{C}$ as $\mathcal{C} = \mathcal{C}' \cup \{e_{k+1}\}$ with $\max(\mathcal{C}') \leqslant k$ and also $\neg\exists \mathcal{C}'' \subseteq \mathcal{C}'.\ \mathcal{C}' \in \mathrm{Res}_{k+1}$. Hence also $\neg\exists \mathcal{C}'' \subseteq \mathcal{C}'.\ \mathcal{C}' \in \mathrm{Res}_k$ and by induction $\mathcal{C}' \in \mathrm{Power}_k$. Therefore, we obtain $\mathcal{C} = \mathcal{C}' \cup \{e_{k+1}\} \in \mathrm{Power}_{k+1}$ by adding in Line 17 since otherwise there would be due to Line 11 a subset in Res_{k+1}. If however $e_{k+1} \notin \mathcal{C}$, we have $\mathcal{C} \in \mathrm{Power}_k$: $\max(\mathcal{C}) < k+1$ and assuming the existence of $\mathcal{C}' \subseteq \mathcal{C}$ with $\mathcal{C}' \in \mathrm{Res}_k$ would imply that we also have $\mathcal{C}' \in \mathrm{Res}_{k+1}$, a contradiction. Therefore, $\mathcal{C} \in \mathrm{Power}_{k+1}$. □

With those invariants established we can finally prove the correctness of our algorithm computing minimal but-for causes.

→ Algorithm 7, p. 74

Theorem 25. *Algorithm 7 works as stated, that is, that we have*

$$\textit{Compute\_But-For\_Causes}(\mathsf{TA}, \rho, \phi) = \{\mathcal{C} \mid \mathcal{C} \textit{ is minimal but-for cause for } \phi \textit{ in } \rho \textit{ of } \mathsf{TA}\}$$

for every timed automaton TA*, run* ρ*, and effect* ϕ*.*

Proof. The above considerations yield

$$\begin{aligned}
&\text{Compute_But-For_Causes}(\mathsf{TA}, \rho, \phi) \\
&= \mathrm{Res}_l && \text{Def. } \mathrm{Res}_l \\
&= \bigcup_{k=1}^{l} \mathrm{Cur_Res}_k && \text{Lemma 22.1} \\
&= \bigcup_{k=1}^{l} \{\mathcal{C} \mid \max(\mathcal{C}) = k \wedge \mathcal{C} \text{ is minimal but-for cause}\} && \text{Lemma 24} \\
&= \{\mathcal{C} \mid \mathcal{C} \text{ is minimal but-for cause}\}.
\end{aligned}$$

Therefore, we can after showing the decidability in the previous section now also conclude the computability of but-for and actual causes, recalling again that all presented results apply analogously to the notion of actual causality. □

Corollary 26. *Given a timed automaton* TA*, run* ρ*, and effect* ϕ*, the minimal but-for causes as well as the actual causes for* ϕ *in* ρ *of* TA *are computable.*

→ Theorem 25, p. 80

Proof. Directly by Theorem 25. □

As already for the checking of causes, we finally want to have a brief look on the computational complexity. Therefore, recall our analysis in (the proof of) Lemma 19 particularly of the computations necessary for deciding the $\mathbf{CF}_{\mathsf{BF}}$- and $\mathbf{CF}_{\mathsf{Act}}$-condition as those conditions play also in the computation of causes a major role.

→ Lemma 19, p. 68

We can again show that solving the task takes exponential space. However, with a significant increase in computational effort: the number of model-checking problems that have to be solved now also grows exponentially in the length of the given run.

Theorem 27. *The set of minimal but-for causes and the set of actual causes can be deterministically computed in exponential space.*

Proof. We analyze the complexity of the computation of minimal causes in Algorithm 7. The size of the constructed powerset is $2^{2|\rho|}$, whereby $|\rho|$ denotes the length of the given run. In the worst case, the algorithm has, therefore, to decide $2^{2|\rho|}$ model-checking problems for formula ϕ and automata with the size of the given timed automaton TA times $|\rho|$ in the case of minimal but-for causality, and $2^{2|\rho|}$ model-checking problems for formula ϕ and automata with the size of the given timed automaton TA times $|\rho|^2$ in the case of actual causality. □

→ Algorithm 7, p. 74

Regarding a lower bound for the computation of causes we content with the following remark.

Remark 4.2.1. Showing that exponential space is also necessary for the computation works using a similar argument as in Lemma 20 via a reduction from model-checking. Informally, it is as well easy to convince oneself that the computation of causes must be at least as complex as checking causes. △

→ Lemma 20, p. 70

Notice, that we talk about the worst case behavior of our algorithms occurring when no set of events can be identified as cause. Conversely, the computational effort in the case of multiple and in particular also small causes is indeed measurably lower. We report on concrete runtime measurements for the checking as well as the computation of causes for varying parameters like cause numbers, run length or automaton size in Section 5.3.

→ Section 5.3, p. 90

Chapter 5

Causality Tool

Based on the algorithms presented in the previous chapter, we developed a Python tool for cause checking and computation that is accessible under

https://github.com/FelixJahnFJ/Real-Time-Causality-Tool.

The tool relies on Uppaal, a software for modeling and verifying real-time systems, as well as on the Python library Pyuppaal allowing to use of most of the Uppaal functionalities within Python. We will focus in this chapter on discussing the important aspects of our tool, for more details on Uppaal, we refer to the documentation of Uppaal[1], and regarding details on Pyuppaal to the project page of the library[2].

5.1 Usage and Functionalities

For installing and running the tool, it is basically only required to have a current Python version installed, to install Uppaal, and to install the Python library Pyuppaal. The tool can then be executed by executing its main-file `causality_tool.py` in the command line. The script takes as its first two command line parameters two arguments specifying, firstly, whether causes should be checked or computed, and secondly, which kind of causality notion should be worked with. In addition, paths to the files of the Uppaal-system, of the trace, and (in the case of cause checking) of the set of events that should be processed must be given as program arguments. In Uppaal-system, both the timed automaton as well as the causal effect to consider are specified. For more detailed instructions on the installation and usage of the tool, see the readme-file in the project repository.

[1] https://docs.uppaal.org/ [3]
[2] https://pypi.org/project/pyuppaal/1.0.0/ [1]

```
Causes were computed for:
- Real-time system path: Thesis_Examples/Example_4_2_1/TA.xml
- Trace: Delay trace: <2, a!> <5, a!>
- Effect formula: A[] not Proc_Template.l4
- Causality notion: Minimal But-For Causality

Results: [Delay cause: {(2, 1), (a!, 2)}, Delay cause: {(5, 2)}]
```

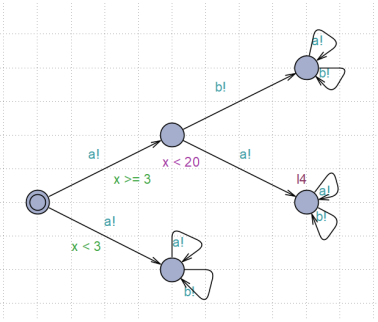
Figure 5.1: Tool Output

The tool checks or computes causes in the sense of our definitions for the given automaton, trace, effect, and set of events. Figure 5.1 shows the output of the tool for the computation of minimal but-for causes in the situation of Example 4.2.1. Recall that it suffices due to our assumption of action-determinism to give a trace as argument to the program as the unique corresponding runs can be reconstructed from its trace. Dependent on whether a delay trace or a timestamp trace is supplied, the tool applies the theoretical definitions either from the delay or from the timestamp perspective. In the tool development, we were driven by the goal to provide automated counterexample explanations, hence, the tool proceeds conversely to our theoretical development: As actual runs, the tool considers runs violating the effect and searches than for counterfactual runs satisfying the effect.

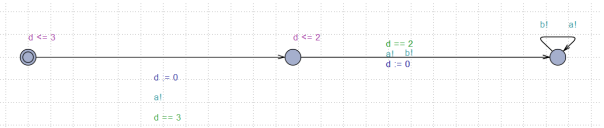
→ Example 4.2.1, p. 75

The automaton and the effect are given as a Uppaal-system. Figure 5.2 shows the Uppaal-models constructed in the process of cause checking, here demonstrated for the processing of Example 3.1.2. As our theoretical causality notions apply only to single automata and not to networks of automata, also the tool supports causal analysis only for a single timed automaton. Uppaal is, however, particularly designed to model networks of timed automata. As a, for our purposes, undesired artifact of this actually intended use of Uppaal, automata modeled in this tool can only perform internal actions by just itself, further actions are only performed as synchronized actions with other automata in a handshaking manner. Hence, a single automaton can only perform indistinguishable internal actions,

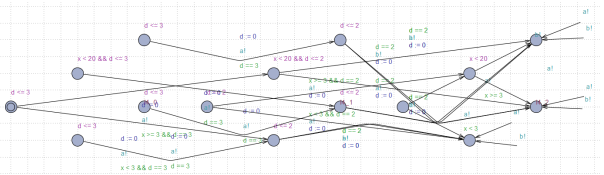
→ Example 3.1.2, p. 31



(a) Initial automaton TA



(b) Counterfactual Trace Automaton $\mathsf{TA}^{\mathcal{C}}_{\xi}$



(c) Intersection automaton $\mathsf{TA} \cap \mathsf{TA}^{\mathcal{C}_2}_{\xi}$

Figure 5.2: Automata Constructions

which is unsuitable for our desired causal analysis considering actions and real-time behavior.

We address this issue by adding a dummy handshaker to the system that allows synchronizations of all declared actions at all times. The handshaker automaton can either be given as an additional template already within the Uppaal input-file itself or is automatically added by the tool.

Also further limitations of Uppaal regarding supported automata transport to our tool, for instance does Uppaal only allow restricted location invariants and guard expressions. Conversely, the tool only supports those system features of Uppaal that we included in our theoretical formalism of timed automata in this thesis as well. Consequently, it does not support features like (global) variable declarations and assignments, the use of parameters, urgent locations, other kinds of channels, etc. As a rule of thumb, the tool does, therefore, roughly support exactly those timed automata expressible in both our theory as well as in Uppaal.

A large restriction of the tool arises from the relatively less-expressive specification language that Uppaal provides. Uppaal only supports to model check the following small fragment of MITL

$$\phi := \Box_I \varphi \mid \Diamond_I \varphi \mid \neg\Box_I \varphi \mid \neg\Diamond_I \varphi,$$

whereby φ is a Boolean combination of atomic propositions.

Actually, Uppaal represents this fragment in a different, rather branching-time mannered syntax, whose semantics does then, however, coincide with the above-stated MITL formulas. We refer to the documentation of Uppaal's specification language for more details on the concrete representation in Uppaal [4].

Also very unfavorable for our work is in this context that Uppaal allows, speaking in MITL terminology, only unique location labeling, that is, different locations cannot be assigned to the same label. Actually, Uppaal uses, again deviating from MITL, no atomic propositions and location labeling at all but refers in the specification formulas to the names of locations. The thereof induced uniqueness confronted us with serious issues in the implementation and, far more unpleasant, causes a clearly perceptible deterioration in the performance of the tool. We will discuss those aspects again in more detail in the following two sections.

Lastly, we want to remark that Uppaal allows, in contrast to MITL, also time-convergent and particularly timelocking runs to qualify as counterexamples to specifications. While timelocks are generally considered as modeling flaws and do therefore usually not receive any major attention, this aspect of Uppaal can result in undesired deviations of the causal analysis provided by the tool from the theoretical causal analysis.

5.2 Implementation Remarks

We discuss our implementation in more detail, point out important design decisions, and comment on challenges and problems that occurred in the programming process.

As already mentioned, we rely our causality tool on Uppaal and its Python library Pyuppaal that enabled us to develop our causality tool in the first place. Nonetheless, we want to remark, that we were a little surprised about the overall paucity of available modeling and verification tools working on real-time systems. While Uppaal was chosen to be used as it seemed to offer overall the best and most functionalities among the easily accessible and usable tools, we still missed at various points certain features we would have expected to exist for model checkers also of real-time systems.

Besides the aforementioned restricted MITL fragment that Uppaal supports, we miss in particular for our purposes the possibility to obtain concrete counterexample traces from the model checker in a case of a formula violation. Uppaal only produces symbolic traces consisting of symbolic locations, that is, sets of locations described by a number of clock constraints [5]. Those symbolic traces, called also timed diagnostic traces, do however not yield directly concrete delay values of counterexamples. Due to the thereof arising compatibility issues between the produced model checking results and the required inputs to our causality notions, it is not ruled out that we will switch for future versions of the tool to at least a different model checker, see for this also the discussions in Section 7.2.

→ Section 7.2, p. 115

In the following, we will discuss the concrete object-oriented implementation in more detail and subdivide this according to the file organization of the source code.

Comprehensive Data Structures for Real-Time Systems

Corresponding Python File: src\ta_structs.py

In particular and relevant for our work, Pyuppaal allows to parse and write xml-files, the common file format in which Uppaal stores and processes real-time systems and, allows to call the Uppaal's model checking functionalities of from Python. Pyuppaal represents the parsed xml-files as a so-called ElementTree, a data structure from Python basically mirroring the tree-like structure of an xml-file with its possible keys, labels, and items at every node. Furthermore, Pyuppaal features a small number of editing options for those real-time systems, which are, however, not even close to allow the elaborate constructions and operations on real-time systems that are required for the implementation of our causality notions.

As this is as we would claim also not properly possible using the ElementTree data structure at all, we wrote in a first step suitable and comprehensive data structures for real-time systems as well as translations from and to the ElementTree format. Combined with the functionalities of Pyuppaal, we thereby then also obtained the possibility to

parse xml-files to our data structure, write systems from our data structure to xml-files, and to model check those systems in Uppaal.

From the bottom up, we wrote Python classes for positions (used for the visual displacement in the Uppaal-GUI), locations, transitions, (timed automata) templates, and finally whole systems. For each of those classes, the constructor method can be used as a translator from the ElementTree representation to the representation in our data structures. For the back translation, the classes contain methods called to_ET, whereby most of the work is at those points handled by constructors existing in Pyuppaal.

Also in those classes, we implemented the intersection of timed automata (cf. Definition 2.7). Again in a bottom-up approach, we implemented the product construction starting with building product locations, building the product of transitions, composed then in the product of templates and systems. As this was for the most parts a technical but relatively straightforward implementation of the formal definition, we encountered a big issue with regard to the goal of designing the intersection in a way that the logical specifications transports to the intersection.

→ Definition 2.7, p. 18

Recall that in the theoretical development, we constructed the labeling function as the projection to the labels of the first location yielding the necessary transport properties. Uppaal and its logic, however, do as mentioned not work with atomic propositions and labeling functions, but the specifications refer to the name of locations that have to be unique among the locations. Hence, lifting the specifications by naming product locations simply after their first component is not possible in Uppaal.

Therefore, we were forced to the following unpleasant solution: we indexed the name of the first location with the id of the second location and then adapt the specification accordingly. Unfortunately, this adaption leads, in addition, inevitably to a factorial blow up of the specification by the number of locations of the second automaton: The in the specification occurring location names l are replaced with the disjunction $l_1 \vee \cdots \vee l_t$ of all corresponding indexed location names, whereby t denotes the number of locations of the second automaton.

The last functionality implemented in the classes of src\ta_structs.py that we want to point out is the construction of the contingency automaton. Again, the implementation of the formal Definition 3.6 is for the most parts straightforward and uses similar techniques as the implementation of the intersection operator. In particular, we again have to resort to the described transforming of location names and specifications. As we will use the contingency automaton also in intersections, this even results in a further factorial blowup of the specification size.

→ Definition 3.6, p. 41

Data Structures for Traces, Runs, and Causes

Corresponding Python File: src\trace_structs.py

Besides the data structures capturing real-time systems we also had to represent the

different kinds of runs, kinds of timed traces, and strongly connected kinds of causes. Recall that by our assumption of considering only timewise action-deterministic automata, the corresponding trace of a run is a unique representation of its run such that it sufficed in the implementation to mainly work with timed traces.

In fact, there is exactly only one point at which we need concrete run configurations, that is, locations and clock assignments of the considered run, namely for the construction of its contingency automaton. We capture this necessary information in a class for configurations, which provides, furthermore, most of the functionalities required for the reconstruction of the run from a given trace. In particular, we therefore had to write methods that check whether guards are satisfied and methods updating clock assignments according to a clock expression.

We represent the different kinds of traces in separate classes for finite delay traces, lasso-shaped delay traces and finite timestamp traces. In each of these classes, we include methods for checking whether the trace satisfies a given set of events and, most importantly, methods for constructing the counterfactual trace automaton of the trace for a given set of events. The implementation of the construction from Definition 3.3 was an easy finger exercise where we took advantage in particular of our object-oriented programming approach, as the exact constructions of the counterfactual trace automaton vary only slightly for the different kinds of traces.

→ Definition 3.3, p. 27

The implementation of the aforementioned sets of events forms the last part of the file, we wrote classes for sets of events in the delay perspective and for sets of events in the timestamp perspective named as DelayCause and TimestampCause.

Functionalities for Cause Checking and Computation

Corresponding Python File: src\cause_checker.py

The major part of the implementation of the algorithms from Chapter 4 is contained in the file src\cause_checker.py. A class named CauseChecker provides for each of the conditions of the causality notions (cf. Definitions 3.5 and 3.7) a method checking whether the particular condition is fulfilled. This is again a mainly straightforward implementation of the pseudo code presented in Chapter 4, we only want to comment on two aspects:

→ Chapter 4, p. 61

→ Definitions 3.5, 3.7, p. 30 and 42

Firstly, recall that we consider in the tool counterexamples to specifications, that is, runs that do not satisfy a given effect ϕ. In the counterfactual simulation, we therefore have to decide whether there exist counterfactual runs satisfying ϕ. This can be done by model checking the negation of ϕ. As UPPAAL allows in its restricted specification language, however, no top-level negations $\neg\phi$, we had to implement a propagation of the negation to the inside of the formula that flips the temporal operators.

Secondly, for checking whether a finite run satisfies the given effect, we had to deviate from the construction in the theoretical development. Recall that we defined the satisfiability of an effect for finite runs via the notion of good prefixes what was then

verified by considering all possible run extensions. We did so by using the construction from Definition 4.1, where we allowed the extension to be arbitrarily labeled. As we cannot properly mirror this idea in Uppaal again due to its lack of location labeling, we must help ourselves by exploiting a special peculiarity of the Uppaal model checker. As mentioned above Uppaal considers also time-locking runs as counterexamples to specifications (in contrast to MITL that only considers time-divergent runs). By enforcing a time lock at the end of the run, we achieve an at least for the restricted logical fragment of Uppaal suitable solution for checking the effect satisfiability also in finite runs.

→ Definition 4.1, p. 64

The advanced Algorithm 7 computing minimal but-for and actual causes for a given effect in a run of timed automata is implemented in a separate `CauseComputer` class, taking as expected use of the functionalities of cause checking.

→ Algorithm 7, p. 74

Executable Main-File

Corresponding Python File: `causality_tool.py`

At the top-level of the repository, we placed as mentioned a standard main-file for the execution of the causality tool. It includes a basic command line parsing and handles the method calls for cause checking and computation accordingly to the given program arguments.

Further Implementation Remarks

In the further files and directories we included scripts and code used for the experiments, use studies and measurements with our tool, which we will also discuss in more detail in the following two sections. We want to conclude this section with adding some remarks about further challenges and specifics in the development of the tool.

As mentioned, Pyuppaal converts real-time systems from xml-format only to a representation in terms of `ElemenTrees`. Even though we, hence, decided as discussed to write for our purposes better suited own data structures for real-time systems, those structures have one major shortcoming: most of the attributes stored as strings in the `ElementTree`, like for instance guards, expressions, system declarations, queries etc., are not parsed to a representation as abstract syntax trees but simply copied to our structures as strings. Consequently, we dealt for all of those attributes in the whole following computations just with strings without any further syntactic or semantic information.

The therefore numerous necessary string operations for editing transition guards, setting transition expressions, changes in the (system) declarations, or adaptions of specification formulas turned out to be one of the biggest challenges in the whole development. In fact, here lies also one of the main weaknesses in the tool: we implemented those string operations as mentioned firstly only for the parts of Uppaal that are also part of our formal setting and secondly, we assumed the occurring attributes likes guards, expressions, declarations or formulas to be valid attributes in Uppaal. For

invalid attributes, the behavior is not further defined and very likely we do also not intercept at every point all possible invalid inputs. Thirdly, we require certain aspects of the models like the (system) declarations to be specified in a very particular format (cf. usage instructions in the project repository), as this tremendously simplified the implementations of the string operations.

In retrospect, it might have been the better choice to invest the time for implementing data structures and parsing functionalities at least for some particular of the string stored attributes, we refrained from it as the expression language of Uppaal is a quite expressive one which would have required to write a rather elaborate parser, a complex task not directly located in our intended field of research.

5.3 Experiments and Measurements

We report on various experiments and measurements we have carried out on the developed tool. First of all, recall that all the presented examples from the previous Chapters 3 and 4 are as Uppaal files together with the respective considered traces and causes available in the project repository. On all those examples, the tool proceeds as expected and computes exactly the causes we also obtain on paper.

5.3.1 Experiments on Examples from the Literature

Existing examples and popular benchmarks for real-time systems in the literature deal almost entirely with network scenarios and can, hence, not directly be handled by our tool. Nonetheless, we want to present two use studies on literature examples that we made suitable for our tool by manually constructing corresponding product automata. Both examples are again provided also for own testing in the repository.

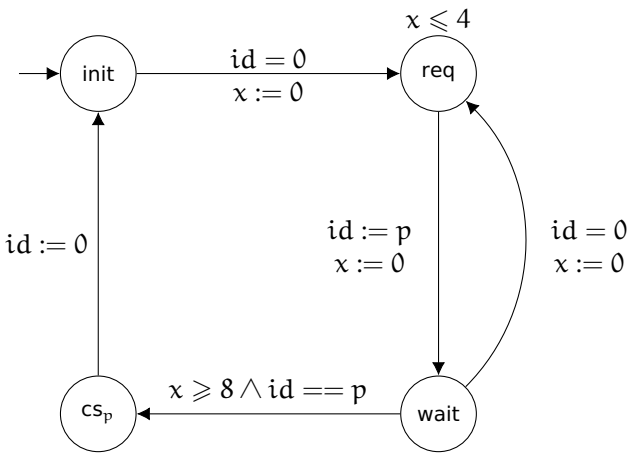


Figure 5.3: Fischer's Protocol, Process TA_i

As a first use study, we look at the famous Fischer's protocol, a mutual exclusion protocol for real-time systems [51]. In the protocol, processes TA_p with $p = 1, \ldots, n$ as depicted in Figure 5.3 are considered.

→ Figure 5.3, p. 90

Hereby, the processes use a common global variable id and assign it when entering the "wait"-location with their individual process index[3].

For $n = 2$, we constructed the product automaton in UPPAAL using actions τ_1 and τ_2 to distinguish which one of the processes takes a transition. The global variable id with finite domain ($id \in \{0, 1, 2\}$) is encoded in distinguished locations. Overall, we obtained an automaton with 21 reachable locations. We test our tool on multiple runs that satisfy the effect $\phi = \square \neg cs_2$, that is, runs in which the second process never reaches the critical section (a kind of fairness violation effect).

Experiment 1. We consider the runs ρ_1, ρ_2, and ρ_3 of the product system that have the following corresponding traces respectively:

$$\xi_1 := \Big(\langle 1,\tau_1\rangle\langle 1,\tau_1\rangle\langle 8,\tau_1\rangle\langle 1,\tau_1\rangle\Big)^\omega$$

$$\xi_2 := \langle 1,\tau_2\rangle\Big(\langle 1,\tau_1\rangle\langle 1,\tau_1\rangle\langle 8,\tau_1\rangle\langle 1,\tau_1\rangle\Big)^\omega$$

$$\xi_3 := \langle 1,\tau_1\rangle\langle 1,\tau_2\rangle\langle 1,\tau_2\rangle\Big(\langle 1,\tau_1\rangle\langle 8,\tau_1\rangle\langle 1,\tau_1\rangle\langle 1,\tau_1\rangle\Big)^\omega$$

Those runs have for the effect ϕ each with different causes, whereby the minimal but-for and actual causes coincide:

Causes for ϕ in ρ_1: $\mathcal{C}_1 = \{(\tau_1,1),(\tau_1,2),(\tau_1,3)\}$

$\mathcal{C}_2 = \{(1,1),(8,3),(\tau_1,1),(\tau_1,4)\}$

$\mathcal{C}_3 = \{(1,1),(8,3),(\tau_1,2),(\tau_1,3)\}$

Causes for ϕ in ρ_2: $\mathcal{C}_1 = \{(1,3),(\tau_1,2),(\tau_1,3)\}$

$\mathcal{C}_2 = \{(8,4),(\tau_1,2),(\tau_1,3)\}$

Causes for ϕ in ρ_3: $\mathcal{C}_1 = \{(1,3),(\tau_1,1)\}$

$\mathcal{C}_2 = \{(\tau_2,3),(\tau_1,4),(\tau_1,5)\}$

$\mathcal{C}_3 = \{(\tau_1,4),(\tau_1,5),(\tau_1,7)\}$

$\mathcal{C}_4 = \{(8,5),(1,7),(\tau_1,6),(\tau_1,7)\}$

$\mathcal{C}_5 = \{(8,5),(1,7),(\tau_1,4),(\tau_1,5)\}$

$\mathcal{C}_6 = \{(1,3),(1,7),(\tau_2,2),(\tau_2,3),(\tau_1,5),(\tau_1,6),(\tau_1,7)\}$

We furthermore measure the time that it takes to decide some of those causes with differing sizes and measure the time that it takes to compute the causes, both for but-for

[3]Global variables are a feature that is not included in our formalism of timed automata, but can be added without losing decidability and is present in UPPAAL as well.

as well as actual causality. Timeout (TO): after 1 hour.
Results: Table 5.4 and Table 5.5.

Run	Run Length	Checking BF-Cause	Checking Min. BF-Cause Cause size = 2	Checking Min. BF-Cause Cause Size = 3	Checking Min. BF-Cause Cause Size = 4	Computing BF-Causes
ρ_1	4	0.17s	–	0.46s	0.54s	19.0s
ρ_2	5	0.20s	–	0.51s	0.60s	87.1s
ρ_3	7	0.18s	0.35s	0.46s	0.52s	1248s

Table 5.4: Fischer's Protocol – But-For Causality

Run	Run Length	Checking Act-Cause Cause size = 2	Checking Act-Cause Cause size = 3	Checking Act-Cause Cause size = 4	Computing Act-Cause
ρ_1	4	–	1.76s	1.62s	66.9s
ρ_2	5	–	1.91s	2.15s	348.7s
ρ_3	7	1.61s	2.93s	3.02s	TO

Table 5.5: Fischer's Protocol – Actual Causality

△

Overall, the cause analysis in the above experiment identifies sets of events as causes that we would also intuitively expect to be causes. In particular with increasing run length, the analysis yields, however, also further sets that were manually less obvious to discover. Also with increasing run length, we can already investigate despite the still relatively small run length the rapidly increasing computational effort that the computation of causes requires.

As a second example from the literature, we consider the running example in Coenen et al.'s paper on explaining HyperLTL violations [22] considering the automaton in Figure 5.6.

As they work in a discrete setting, time will play no crucial role in this application. It is still a particularly interesting example to test on our work as the authors

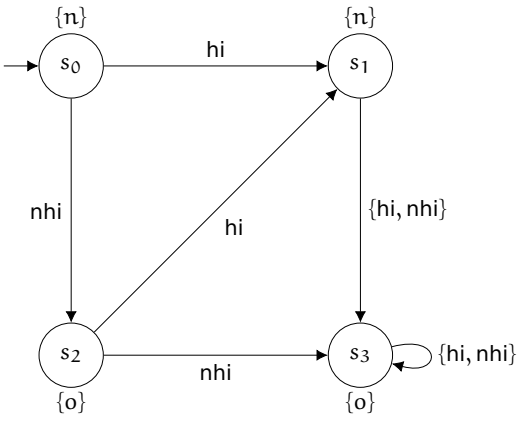


Figure 5.6: Coenen et al.'s HyperLTL Example

give for this automaton a preemption example that can be handled by their definition of actual causality and what we desire from or notion our as well. Coenen et al. explain as mentioned hyperlogic properties, i.e., properties that express relations between multiple system executions. Concretely, they consider for this example violations of the hyperproperty of observational determinism, stating simplified and transported to the real-time setting that $\forall s_1, s_2.\Box\,(o_{s_1} \leftrightarrow o_{s_2})$, whereby s_1 and s_2 are signals and o_s expresses that o holds on the signal s.

We transport this consideration of hyperproperties to our setting again by a product automaton construction. Furthermore, we encode the in each step independently taken actions in the two runs in indexed actions $\mathit{Act} = \{\mathsf{hi}_1, \mathsf{nhi}_1, \mathsf{hi}_2, \mathsf{nhi}_2\}$ and their separate labels as paired atomic propositions $\mathit{AP} = \{\langle o, o\rangle, \langle o, n\rangle, \langle n, o\rangle, \langle n, n\rangle\}$. We then consider in the product automaton the effect $\phi = \Diamond\langle o, n\rangle \vee \langle n, o\rangle$[4].

We mimic the consideration of the hyperproperty of the two runs by enforcing alternating actions between the actions in $\{\mathsf{hi}_1, \mathsf{nhi}_1\}$ (representing actions in the first run) and the actions in $\{\mathsf{hi}_2, \mathsf{nhi}_2\}$ (representing actions in the second run). As this enforcement is encoded in the automaton, the number of locations in the automaton is enlarged to overall 18 reachable locations.

Experiment 2. We consider the run ρ that has the corresponding trace

$$\xi := \langle 1, \mathsf{nhi}_1\rangle\langle 1, \mathsf{hi}_2\rangle\langle 1, \mathsf{nhi}_1\rangle\langle 1, \mathsf{hi}_2\rangle\langle 1, \mathsf{nhi}_1\rangle\langle 1, \mathsf{nhi}_2\rangle$$

The causal analysis based on our notions of but-for and actual causality coincides as desired exactly with the analysis of Coenen et al.'s notions as also we obtain the following causes:

Minimal but-for causes for ϕ in ρ: $\mathcal{C}_1 = \{(\mathsf{nhi}_1, 1)\}$

$\mathcal{C}_2 = \{(\mathsf{hi}_2, 2), (\mathsf{hi}_2, 4)\}$

$\mathcal{C}_3 = \{(\mathsf{hi}_2, 2)(\mathsf{nhi}_1, 3)\}$

Actual causes for ϕ in ρ: $\mathcal{C}_4 = \{(\mathsf{nhi}_1, 1)\}$

$\mathcal{C}_5 = \{(\mathsf{hi}_2, 2)\}$

In particular, we observe a typical preemption case: $\mathcal{C}_5 = \{(\mathsf{hi}_2, 2)\}$ is identified as actual cause but not as minimal but-for cause since $(\mathsf{hi}_2, 2)$ preempts $(\mathsf{hi}_2, 4)$. Hence, only together those two events form the but-for cause $\mathcal{C}_2$.

Again, we measure the time that it takes the tool to check and compute but-for and actual causes.

Results: Table 5.7 (on the next page). △

[4]Notice that Coenen et al. considered as effect violations of the property. We instead look for effects for satisfied formulas, hence, our formula is modulo encoding the negation of the hyperproperty formula.

Run	Run Length	Checking BF-Cause	Checking Min. BF-Cause Cause size = 2	Computing BF-Cause	Checking Act-Cause Cause size = 1	Computing Act-Causes
ρ	6	0.19s	0.37s	124.1s	0.18s	582.5s

Table 5.7: Coenen et al.'s HyperLTL Example

For both examples, we refer to the UPPAAL-files for details on the concrete adaption via product encodings to our setting.

5.3.2 Cause Checking Measurements

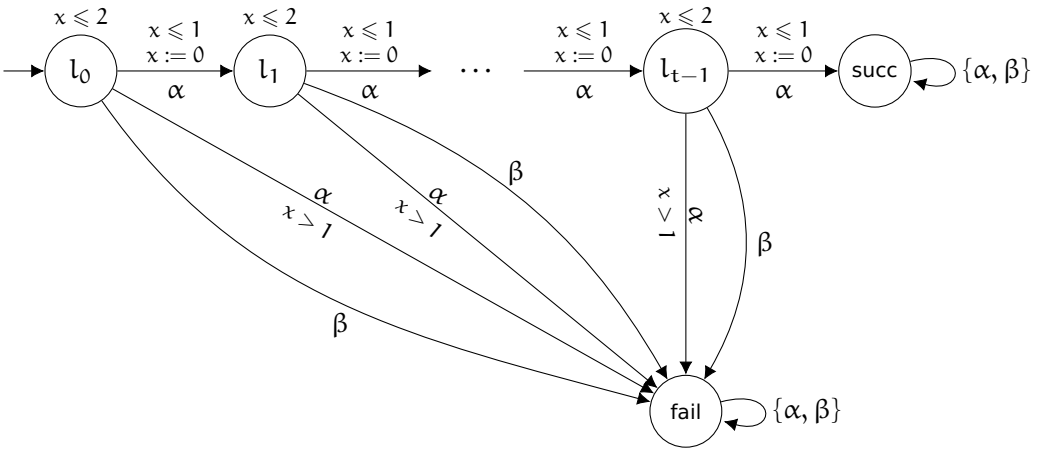
Going beyond the above use studies, we want to systematically analyze the runtime and efficiency of our implementations and the underlying algorithms from Chapter 4 in more detail. In particular, we want to investigate the dependencies on varying parameters like the size of the automaton, the length of the given run, or the cause size. Therefore, we present synthetic experiments constructed exactly designed to run measurements with the mentioned varying parameters. The corresponding Python scripts are available under

→ Chapter 4, p. 61

https://github.com/FelixJahnFJ/Real-Time-Causality-Tool/tree/main/src/Experimenter_Scripts.

We aim to measure both the checking of a cause as well as the computation of causes both for (minimal) but-for as well as for actual causality. As the runtime in particular of the computation of causes highly dependents on where the events forming causes appear in the order of the processed events, we want to include a randomization of which events form causes and then look at the average over a sample of measurements.

To efficiently achieve the aim of checking-measurements with varying parameters and randomized causes, we use the following construction: For a parameter t specifying the size of the timed automaton, we consider the following timed automaton TA:



The idea is that the only run in the automaton reaching the "succ"-location is the run $\rho' := l_0 \xrightarrow{1}\xrightarrow{\alpha} l_1 \xrightarrow{1}\xrightarrow{\alpha} \ldots \xrightarrow{1}\xrightarrow{\alpha} l_{t-1} \xrightarrow{1}\xrightarrow{\alpha} \mathsf{succ}$. We now, however, consider runs with a deviating delay of 2 or a deviating action β at randomized points, that hence, lead the run to the "fail"-location. More formally, for a certain run length $r \leqslant t$ (the run should not be longer as the automaton), we pick two random index sets $I_D \subseteq \{1,\ldots,r\}$ and $I_A \subseteq \{1,\ldots,r\}$ and consider the run ρ with corresponding trace

$$\xi = \langle \delta_1, \alpha_1 \rangle \ldots \langle \delta_r, \alpha_r \rangle,$$

whereby

$$\delta_i := \begin{cases} 1, & i \notin I_D, \\ 2, & i \in I_D, \end{cases} \qquad \alpha_i := \begin{cases} \alpha, & i \notin I_A, \\ \beta, & i \in I_A, \end{cases} \qquad i = 1,\ldots,r.$$

By this construction, we then obtain as the only minimal but-for as well as actual cause for the effect $\phi := \Diamond\, \mathsf{fail}$ in run ρ of TA exactly the set of events

$$\mathcal{C} = \{(2,i) \mid i \in I_D\} \cup \{(\alpha,i) \mid i \in I_A\},$$

for which we will then measure how long it takes to verify its cause property. Notice furthermore, that the cause has size $|I_D|+|I_A|$, i.e., the size of the randomly picked index sets, which allows us to also vary the size of the cause using a third parameter c.

This approach results in the following measurements series:

Measurement 1. We increase the size t of the timed automaton TA together with the length r of run ρ (we let $r = t$) and consider varying sizes c of the cause. We measure but-for, minimal but-for and actual causality checking separately. Sample size per parameter combination: 10.
Results: Table 5.8, Table 5.9, and Table 5.10 (on the next page).

	$c=2$	$c=5$	$c=10$	$c=15$	$c=20$
$t=r=10$	0.21s	0.20s	0.20s	0.21s	0.19s
$t=r=25$	0.74s	0.74s	0.72s	0.73	0.72s
$t=r=50$	4.29s	4.35s	4.37s	4.33s	4.27s
$t=r=75$	16.2s	16.0s	15.9s	15.7s	15.6s
$t=r=100$	45.5s	45.3s	45.3s	44.8s	44.4s

Table 5.8: Min. But-For Causality

	$c=2$	$c=5$	$c=10$	$c=15$	$c=20$
$t=r=10$	0.49s	0.84s	1.51s	2.13s	2.83s
$t=r=25$	1.59s	2.61s	4.56s	6.20s	8.19s
$t=r=50$	8.22s	14.3s	24.7s	35.0s	45.7s
$t=r=75$	28.4s	50.0s	86.2s	123s	160s
$t=r=100$	76.8s	135s	232s	329s	427s

Table 5.9: Checking Actual Causality

△

	$c = 2$	$c = 4$	$c = 6$	$c = 8$
$t = r = 5$	0.78s	1.35s	1.92s	2.58s
$t = r = 10$	3.43s	5.90s	8.78s	11.7s
$t = r = 15$	17.4s	30.2s	44.6s	60.1s
$t = r = 20$	65.9s	116s	168s	218s
$t = r = 25$	216s	373s	522s	704s

Table 5.10: Actual Causality

➔ Lemma 19, p. 68

We observe multiple things: First of all, the measurements reflect the analysis on the algorithmic complexity in Lemma 19. Unsurprisingly, the computational effort of cause checking increases in general with increasing automaton and run size. Furthermore, the effort of checking minimal causes increases with the cause size as for larger causes there are more subsets that need to be considered to verify minimality. Lastly, Table 5.8 suggests that checking but-for causality (i.e., without checking minimality) gets, in particular for large automata and runs, less expensive for increasing cause sizes.

All those observations can be further confirmed, whereby we refine at first the latter suggestion.

Measurement 2. We fix the automaton and run size ($t = r = 100$) and measure for increasing cause sizes c the computational effort it takes to check the **SAT**- and $\mathbf{CF}_{\mathsf{BF}}$-condition. Sample size per cause size: 10.
Results: Table 5.11.

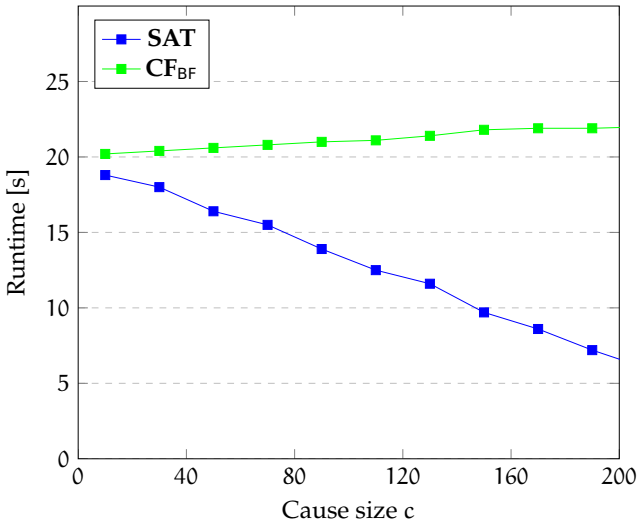


Table 5.11: Checking **SAT** and $\mathbf{CF}_{\mathsf{BF}}$ △

The measurement locates the decreasing computational effort of but-for cause checking in a decreasing runtime of checking the **SAT**-condition. The reason for this lies hidden in the model checking process: Recall that we model check for the **SAT**-condition the satisfaction of the effect ϕ. For the case at hand, the effect is the reachability property $\Diamond\,\mathsf{fail}$. As the model checking proceeds in a breadth first manner, the reachability can be confirmed faster for runs that reach in fewer steps the searched location. For the present experiment setup, this happens with the first appearing deviation from timed actions $\langle 1, \alpha \rangle$. Hence, an earlier occurring of the first causal events, which correlates with a larger size of the considered causes, results in a faster satisfiability check. We do not believe that this observation can be fully generalized to arbitrary scenarios with arbitrary automata and specifications.

Apart from this, the crucial parameters affecting the necessary computational effort are the automata size and run length. We investigate their influence on the counterfactual reasoning.

Measurement 3. We compare the time that it takes to check $\mathbf{CF}_{\mathsf{BF}}$- and $\mathbf{CF}_{\mathsf{Act}}$-condition in dependence to the size t of the automaton and length run r (we let $r = t$). The cause size remains fixed ($c = 2$). Sample size per automaton size: 10.
Result: Table 5.12.

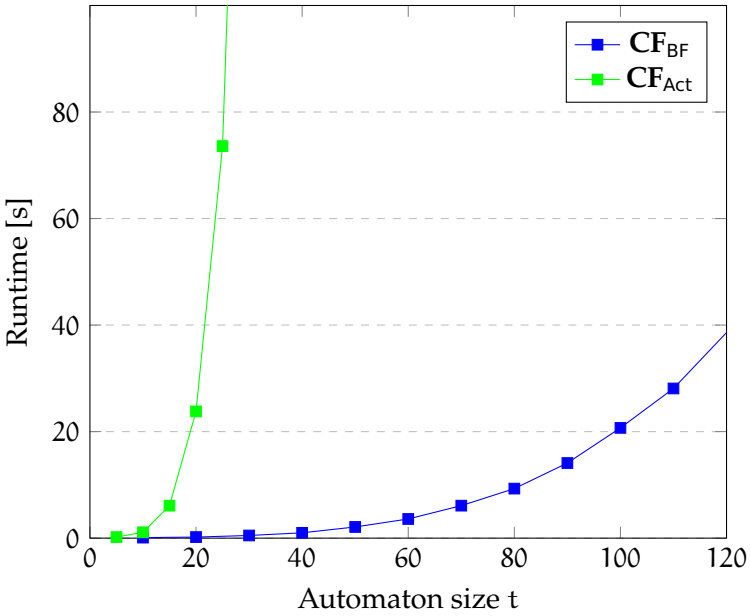


Table 5.12: Checking **CF**-Conditions △

The measurement shows the expected exponential growth in both conditions, whereby checking $\mathbf{CF}_{\mathsf{Act}}$ grows enormously much faster and times out already for a relatively small automaton size. Again, this is perfectly in line with the complexity analysis: We know model checking to grow exponentially and discussed that the size

of the checked automaton increases due to the contingency construction in the $\mathbf{CF}_{\mathsf{Act}}$-condition with an additional factor of the run length r. Furthermore, we think that the runtime of the tool explodes even stronger due to the in the previous section discussed blowup of the specification (and model checking is usually extremely sensitive to the specification size).

The results regarding the counterfactual conditions combine to the following overall measurements for cause checking.

Measurement 4. We compare the time that it takes to check the three notions of but-for, minimal but-for, and actual causality in dependence to the automaton size t and run length r (we let again $r = t$). We fix the cause size ($c = 2$). Sample size per automaton size: 10.

→ Table 5.13, p. 98

Results: Table 5.13.

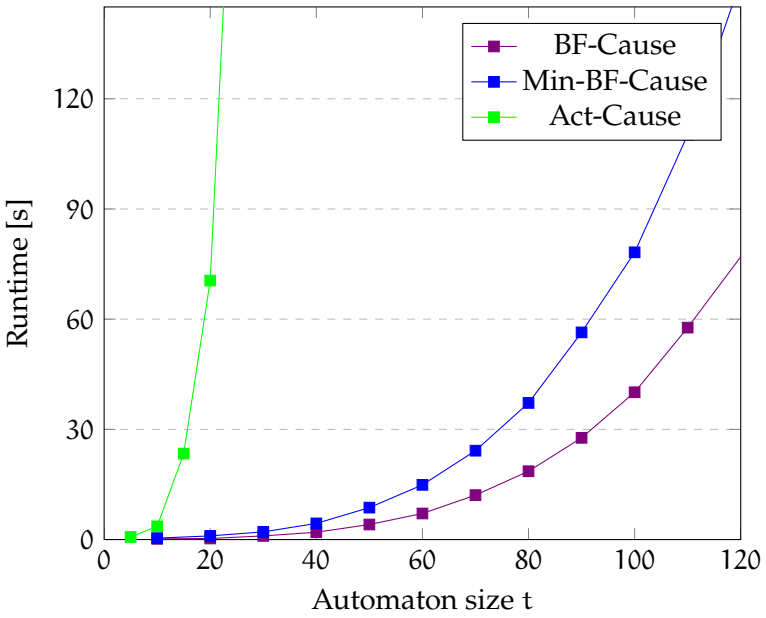


Table 5.13: Checking Causality Notions

△

The measurements show ideally how cause checking is composed out of multiple model-checking problems: For checking but-for causes, we have to solve two model-checking problems, for checking minimal causes of size 2, there are four problems to solve. Hence, checking minimal causes of this size takes about twice as long which is reflected very accurately in the measurements. Similar relations for further cause sizes can also be observed in Measurement 1.

→ Measurement 1, p. 95

Finally, we examine the influence of the run length varied independently to the automata.

Measurement 5. We vary the run length r for different fixed values of the automaton size t. We again fix the cause size ($c = 2$) and measure the time it takes to check the $\mathbf{CF}_{\mathsf{BF}}$- and $\mathbf{CF}_{\mathsf{Act}}$-condition. Sample size per parameter combination: 10.
Result: Table 5.14 and Table 5.15.

	$r = 5$	$r = 10$	$r = 25$	$r = 50$	$r = 75$	$r = 100$
$t = 10$	0.08s	0.11s	–	–	–	–
$t = 25$	0.13s	0.18s	0.37s	–	–	–
$t = 50$	0.20s	0.32s	0.79s	2.20s	–	–
$t = 75$	0.26s	0.46s	1.44s	4.08s	7.66s	–
$t = 100$	0.33s	0.64s	2.21s	6.35s	12.3s	20.3s

Table 5.14: Checking $\mathbf{CF}_{\mathsf{BF}}$

	$r = 5$	$r = 10$	$r = 15$	$r = 20$	$r = 25$
$t = 10$	0.32s	1.14s	–	–	–
$t = 15$	0.46s	2.03s	5.96s	–	–
$t = 20$	0.59s	2.93s	9.38s	22.8s	–
$t = 25$	0.76s	4.24s	13.7s	34.4s	75.0s

Table 5.15: Checking $\mathbf{CF}_{\mathsf{Act}}$ △

This last measurement on cause checking shows that the necessary computational effort grows indeed in the size of the automaton as well as in the length of the actual run. However, the run length seems thereby to be the by far more influential factor. This is probably again due to the discussed shortcoming of our implementation in which the model-checked formula grows in the length of the run. Also clearly visible is in all the measurements that this growth is sharper for the checking of actual causes. This reflects the complexity analysis, in which we discussed that with the contingency automaton also the model-checked intersections grow by an additional factor of the run length. Additionally, again also the formula grows for actual causality as discussed with a factor of r^2 instead of only r in the but-for case.

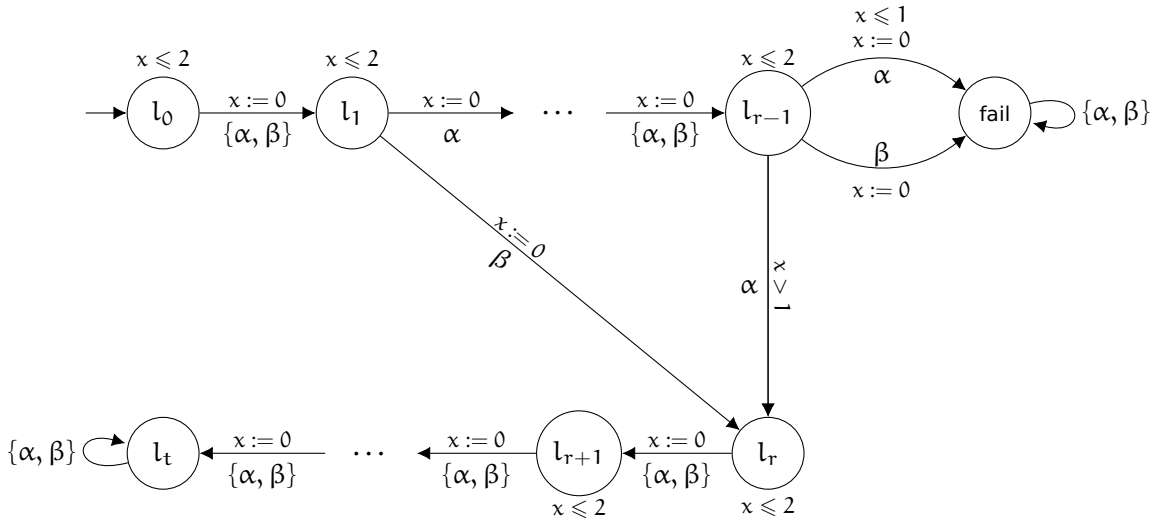
5.3.3 Cause Computation Measurements

We head over to the computation of causes and again present randomized experiments for varying parameter, whereby for the computation now rather the number of causes for a given effect instead of their size will become of importance.

We use a similar idea for the test setup, proceed, however, in some sense conversely as for the checking of causes: While it is analogously to the construction above again the particular trace $\xi := \langle 1, \alpha\rangle \dots \langle 1, \alpha\rangle$ whose corresponding run ρ leads through a particular sequence of locations, we construct the setting now no longer in a way, that this run becomes the searched counterfactual run, but conversely in a way, that this run is the given run in which the effect $\phi := \Diamond\,\mathsf{fail}$ appears. For a given randomized choice of events, we then design the automaton in such a way that a deviation from exactly these events in ξ prevents the effect.

More concretely, we proceed in the following way: in a step whose delay and action events are not in the randomly selected set of events, deviation from ξ should still only allow to continue in the same successor location as ρ. Hence, the transition allows all actions and is not guarded. If, however, the delay event at this point is selected, a deviating delay of 2 instead of 1 should change the successor location. Hence, the α-transition to the successor becomes guarded and an alternative transition enabled for a delay of 2 is added. Similarly for a selected action event, a deviating action of β instead of α at this point should change the successor location and hence, the β-transition is redirected to an alternative successor location. The whole construction reminds of the construction of the counterfactual trace automaton, as for each possible combinations of selecting or unselecting delay or action events at a certain position we do obtain differing transitions.

For an example set of events $\mathcal{C} := \{(\beta, 2), (1, r-1)\}$, the considered TA automaton looks, for instance, as follows:



For this example set, we obtain both as minimal but-for and as actual causes for ϕ in ρ of TA the two singleton sets $\{(\beta, 2)\}$ and $\{(1, r-1)\}$. Also in general for a randomly picked

set of c events $\{e_1, \ldots, e_c\}$, the construction results in having exactly the c singleton causes $\{e_1\}, \ldots, \{e_c\}$ as minimal but-for and actual causes. Notice as last aspect of the construction that the locations $l_{r+1}, \ldots l_t$ have no influence at all for the causality analysis but are added to allow varying the automaton size t independently to the run size r.

We make similar measurements as for checking now for the computation of causes:

Measurement 6. Similar to Measurement 1, we measure for varying automata sizes t together with run lengths r ($r = t$) on the one hand and varying numbers of causes c on the other hand now the computation of minimal but-for and actual causes. Sample size per parameter combination: 10. Timeout (TO): after 600 seconds.
Results: Table 5.16 and Table 5.17.

→ Measurement 1, p. 95

	$c = 2$	$c = 4$	$c = 6$	$c = 8$	$c = 10$	$c = 12$
$t = r = 2$	0.27s	0.23s	–	–	–	–
$t = r = 4$	3.92s	1.27s	0.64s	0.59s	–	–
$t = r = 6$	102s	33.6s	9.75s	3.40s	2.02s	1.90s
$t = r = 8$	TO	531s	176s	47.5s	13.5s	5.15s

Table 5.16: Computing Min. But-For Causes

	$c = 2$	$c = 4$	$c = 6$	$c = 8$	$c = 10$	$c = 12$
$t = r = 2$	0.82s	0.69s	–	–	–	–
$t = r = 4$	14.8s	4.43s	2.16s	1.96s	–	–
$t = r = 6$	339s	83.1s	23.6s	7.07s	4.04s	3.78s
$t = r = 8$	TO	TO	663s	170s	43.6s	15.5s

Table 5.17: Computing Actual Causes

△

As expected, we observe a rapid runtime explosion with increasing size of the automaton and length of the run while the number of causes decreases the necessary computational effort. The upcoming two measurements address these two aspects in more detail. Furthermore, the higher complexity in checking the $\mathbf{CF}_{\mathsf{Act}}$-condition in comparison to checking the $\mathbf{CF}_{\mathsf{BF}}$-condition induces also a significantly larger runtime in the computation of actual causes compared to minimal but-for causes.

Measurement 7. We increase the automaton size t together with the run length r and let the number of causes c be fixed ($c = 2$). Sample size per cause size: 10.
Results: Table 5.18 (on the next page).

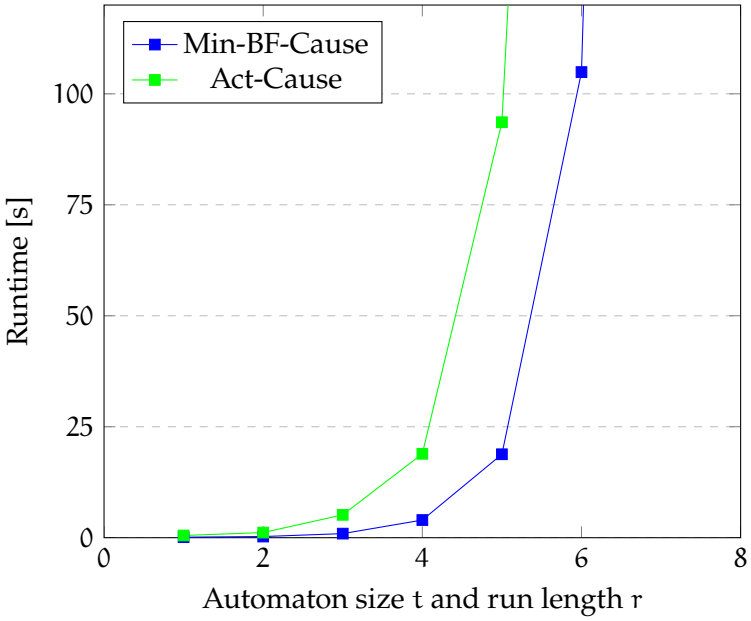


Table 5.18: Cause Computation – Increasing Automaton Size

△

The measurement shows again the rapid exponential blow-up in the runtime of cause computations with increasing run length. This is above all due to the exponential growth of the powerset of event sets we have to consider as potential causes. For run length r, we have precisely 2^{2r} possible sets of events for which the **CF**-conditions are checked. Recall however, that our optimized algorithm for the computation of causes allows in the case of detected causes to exclude its supersets from the further computation. Indeed, the positive impact of this optimization on the runtime, especially in scenarios with a large number of causes, can be confirmed experimentally.

Measurement 8. We increase the number of causes c for fixed automaton size and run length $(t = r = 5)$ and measure the computation of minimal but-for and actual causes. Sample size per cause size: 10.
Results: Table 5.19 (on the next page). △

→ Theorem 27, p. 81

Lastly, we confirm our suggestion and the complexity analysis in Theorem 27, that it is in fact especially the run length and less the automaton size that leads to the runtime explosion in the computation of causes.

→ Measurement 5, p. 99

Measurement 9. Similar to Measurement 5, we vary the run length r and automaton size t independently of each other, while the cause size remains fixed $(c = 2)$. Sample size per parameter combination: 10.
Result: Table 5.20 and Table 5.21 (on the next pages). △

This concludes the discussion of the developed causality tool. As observed at several points, the tool still comes with a number of shortcomings, limitations, and restricted functionalities, and is hence, as further detailed in Section 7.2, definitely a major subject in intended future improvements of our work.

→ Section 7.2, p. 115

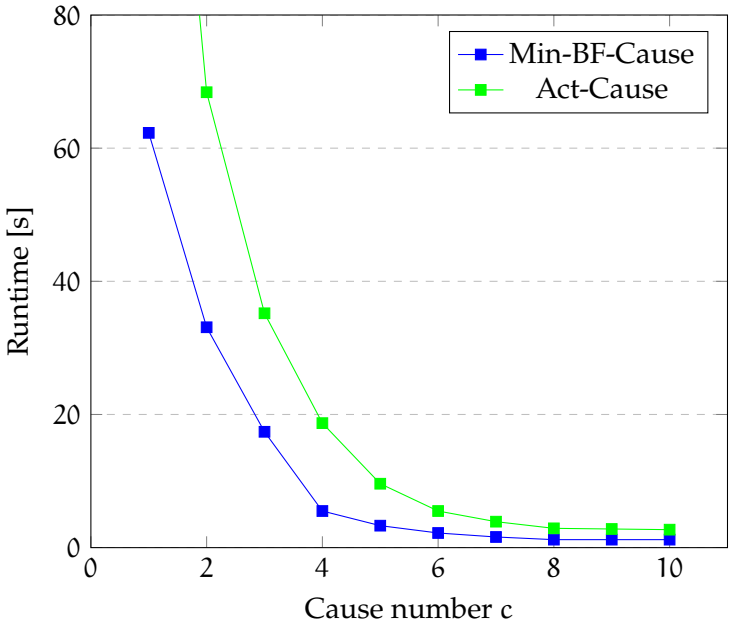


Table 5.19: Cause Computation – Increasing Cause Number

	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$	$r = 6$
$t = 2$	0.23s	0.45s	–	–	–	–
$t = 3$	0.22s	0.47s	1.60s	–	–	–
$t = 4$	0.23s	0.53s	1.65s	6.76s	–	–
$t = 5$	0.27s	0.55s	1.87s	7.43s	31.3s	–
$t = 6$	0.25s	0.63s	1.96s	6.41s	23.3s	117s

Table 5.20: Computing Min. But-For Causes

	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$	$r = 6$
$t = 2$	0.32s	0.79s	–	–	–	–
$t = 3$	0.34s	0.83s	2.99s	–	–	–
$t = 4$	0.34s	0.86s	3.15s	14.8s	–	–
$t = 5$	0.35s	0.90s	3.37s	14.8s	72.4s	–
$t = 6$	0.35s	0.93s	3.52s	15.3s	76.6s	328s

Table 5.21: Computing Actual Causes

Chapter 6

Related Work

We report on previous work related to this thesis. In Section 6.1, we give in a first step an overview of the research on causality in the area of computer science and verification in general and look then, in Section 6.2, in more detail into causality-related work particularly for real-time settings.

→ Section 6.1, p. 105
→ Section 6.2, p. 108

6.1 Causality in Computer Science

With the aim of explaining and understanding the behavior of computer systems, the originally philosophical concept of causality was started to be applied to computer science contexts. The also in this thesis followed idea of counterfactual causality – going back to first thoughts of Hume [46], retrieved by Lewis [53], and then established through Halpern and Pearl's influential formalizations of actual causality [40, 41, 37, 38], became one of the most important approaches for integrating causality in computer science contexts. This is particularly the case for research areas focusing on verification.

The principle of counterfactual reasoning was thereby – sometimes implicitly – used from early on in many approaches. One of the first approaches in this manner was so-called program slicing, which tries to analyze dependencies between different parts of a program [65, 44, 43].

For component-based systems, their individual components are considered as (parts of) possible causes for a certain system behavior [33, 32, 64, 36]. Already well before the formal work of Halpern and Pearl, Reiter searched for minimal sets of components, that explain a discrepancy between an observed faulty and the desired correct system behavior [58]. Similarly for configurable systems, which in the meantime have not only gained in importance but also in terms of complexity, it was tried to detect the features responsible again for specification violations, finding nowadays crucial applications in the debugging process of such configurable systems [10].

The concepts of vacuity and coverage are used to gain knowledge specifically in the case of a successful verification: While vacuity tries to rule out that an undesired trivial behavior of the system caused the positive result [15, 13], coverage analysis proceeds conversely and tries to identify parts of a system that were irrelevant for a positive result [45, 20]. Coverage takes as one of only a few counterfactual reasoning approaches a forward-looking perspective on causality: system parts are analyzed in terms of their global ability to affect the verification result.

This approach differs from the perspective taken in our work, which can be classified as a representative of the rather backwards-oriented field of error localization. Error localization [14, 63, 47] tries to reduce a counterexample to the relevant parts that were causal for the violation, a backward perspective in the sense that its starting point is an already finished system execution, whose events are then analyzed in terms of their effect on the observed outcome. Thereby an important advantage of model checking in contrast to, for instance, deductive verification methods is exploited, as model checking algorithms cannot only answer the question of specification satisfaction but provide as well a certificate or counterexample justifying the model checking result. Those system executions form then the starting point of error localization[1].

Early tries in the subject of error explanation used distance metrics, that measure, loosely speaking, the similarity of system executions. Given, for instance, a counterexample trace, the, with respect to this metric, closest traces leading to a successful execution are then computed [35]. Those closest traces then in turn induce minimal causal events by choosing exactly the events that differ between the closest successful and the originally violating trace.

One of the most groundbreaking works in the field was the one of Beer et al. [16], in which the violation of an LTL formula in paths in Kripke structures is explained by formally applying Halpern and Pearl's causality notion to their setting. Their basic approach to explaining the behavior of discrete systems strongly inspired us in the choice of the setup in our work. But also many works before us took the work of Beer et al. as a model for developing a notion of causality in the sense of Halpern and Pearl for different kinds of systems or specifications: Datta et al. consider, for instance, concurrent systems [25] and Gössler and Stefani lift the ideas to abstract hyperproperties [36].

Coenen et al. then concretize this to explanations of HyperLTL violations. Therefore, they use in the counterfactual reasoning the conceptual idea of counterfactual automata [22]. We closely followed this approach and represent, as explored by Coenen et al., all the possible counterfactual system execution as an automaton and reduce cause checking in this way to the model-checking problem. Also Gössler and Stefani in [36]

[1] Note that the term "error localization" originated from the major use case of this kind of methods, namely to explain violations of specification. As demonstrated also in this thesis, the technique can be applied also to explain the satisfaction of a specification in a system execution.

proceed similarly and define for their component-based systems so-called counterfactual builders, in which the possible counterfactual simulations were represented.

Based on counterfactual trace automata, Coenen et al. include, to the best of our knowledge, as the first authors working with reactive system contingencies in the counterfactual reasoning [22], resulting in not only an implementation of but-for causality but true actual causality in the flavor of Halpern and Pearl. Coenen et al. generalize in a further publication their notion of actual causality to general systems [23]. Again inspired by their work, also we include contingencies to the reasoning enabled by carefully composed automata constructions. As discussed in Section 3.3, we chose to this end, however, a differing technique: While Coenen et al. allow contingencies by extending the possible inputs of their systems, we do so by adding contingency transitions.

→ Section 3.3, p. 55

Another notable discovery from working with hyperproperties is that causality can be elegantly defined in hyperlogics, only with the increase in complexity of one additional quantifier alternation [22]. This also leads us to a last and again very different approach to counterfactual causality in verification contexts, that we want to discuss: the idea to describe causality in terms of logical formulas. To give a brief, superficial impression of how such a description can look like, notice that besides implicative relations expressable in temporal logics also their temporal operators are well suited to formalize the intuitions behind the single causality conditions: $\Diamond C$ and $\Diamond E$ exactly express, that a cause C and effect E occur in a system execution, that is, exactly the requirement in the **SAT**-condition. The until-operator helps in approaching the counterfactual principles in the **CF**-condition: $\neg E \,\mathcal{U}\, C$ for instance expresses that the effect E does not occur until the cause C occurs[2]. Temporal logics are, furthermore, exactly designed to specify relations between events in the course of a system execution. Consequently, formulas in, for instance, a so-called event order logic might be considered by themselves as possible causes for a system behavior [52, 18, 31].

Lastly, we briefly report on two further approaches, differing fundamentally from counterfactual explanations, both again also applied in various works in computer science contexts. First, Chockler and Halpern extend the notion of actual causality to the concept of responsibility [19]. While counterfactual causality is a qualitative notion (a set of events is either a cause for an occurring effect or it is no cause), notions of responsibility add a quantitative measure to the analysis (to what degree is a set of events responsible for an occurring effect). For these kinds of concepts, especially using the so-called Shapley value [59], measuring the influence of events on effects led to successful explanation techniques for system behaviors [26, 60, 6]. Secondly, probabilistic causation analyzes events and effects with regard to the probability of their occurrence. In a nutshell, events that increase with their occurrence the probability of the occurrence

[2] One has, however, to be aware of the well-known pitfall that such a temporal dependence does not necessarily imply a causel dependence.

of an effect are then identified as causes for this effect [61, 29]. Naturally, this approach to causality plays a major role in particular for the work with probabilistic systems [48, 12].

For further details on research about causality in verification-related work, we also recommend the excellent overview paper by Baier et al. [11]. For a closer discussion of methods and approaches summarized under the popular term of "explainable AI", we refer to the systematic review of this field by Vilone and Longo [62].

6.2 Causality in Real-Time Systems

While the range and progress of research on causality in the computer science context is as seen in general already quite large and still continues to expand rapidly, we are only aware of a small number of publications considering real-time systems. We discuss the in Chapter 1 already briefly addressed state of the current research on causality in real-time system in more detail and compare previous works to the one in this thesis.

→ Chapter 1, p. 1

Dierks et al. develop an automated abstraction refinement technique for timed automata [28]. Verification of various systems, in particular also of real-time systems, often proceeds by checking not the given system directly, but instead a simplified abstraction of it. As those abstractions are designed in a way that they over-approximate the concrete system, the correctness of the abstractions implies also the concrete system to behave as intended. In the case of a reported specification violation of the abstract system, the detected counterexample might, however, be spurious, that is, that there exists for the abstract counterexample no corresponding concrete counterexample in the concrete system. In this case, the abstraction needs to be refined in a way that the spurious counterexample is excluded and is then again started to be model checked, resulting in a so-called refinement loop until a valid verification result is found [21, 24].

The crucial step in this verification process is the refinement of the abstractions – a step whose automation can, however, strongly benefit from causality concepts, as they might help to analyze why the detected counterexample was spurious and, consequently, how the abstraction must be refined. For the abstractions of real-time systems, Dierks et al. try to identify variables and clocks causing the spuriousness. Besides the differing application context, as we do not work on abstraction refinements but in the direction of error localization, our causal reasoning differs, therefore, in particular regarding the considered objects for causes: we look for events in a given run while in [28] parts of the real-time system are considered.

This also marks the main difference to the work of Wang et al., who develop a framework for the causal analysis of component-based real-time systems [64]. As briefly mentioned in the previous section, works on component-based systems mainly try to identify faulty components that contributed to a faulty system behavior. Based on

this consideration of causes and effects, Wang et al. then introduce a counterfactual reasoning that simulates the system with the causal ("faulty") components replaced with alternative ("good") components. Identification of faulty components can then ease the repair of component-based systems.

Kölbl et al. follow a similar direction and propose a repairing technique of timed systems focusing on the repair of its clock bounds [49]. Again, also they gain knowledge on what system parts might have caused a violation of a specification by considering a counterexample trace. More precisely, they process counterexamples represented as timed-diagnostic traces, that include timing constraints arising from different guards and invariants of the system (cf. Section 5.2). They encode those constraints from the timed diagnostic trace in linear real arithmetic and search then for admissible alternative clock bounds preventing the violation (admissibility here means the possibility to change those clock bounds in the timed system without crashing the overall functional behavior of the system). As timed diagnostic traces can describe executions of system networks as well, the development of Kölbl et al. handles, in contrast to our work, also networks of timed automata.

→ Section 5.2, p. 86

An advantage that is present also in a further contribution of Kölbl et al.: While in their previously discussed work, syntactic features of the systems like their clock bounds are considered as causes, and hence, called static actual causes, now in [50] the at runtime non-deterministically chosen delay values of timed systems are considered causal for the system behavior, consequently called dynamic actual causes. More precisely, they compute causal delay values and causal delay ranges in timed diagnostic traces violating reachability properties [50]. Those values and ranges can thereby not only refer to a single delay in the trace but also to sums of multiple delays, a feature allowing to detect certain mathematical relations between delays as causes.

While we allow no such delay relations as causes, we consider, however, the performed actions of traces as part of possible causes as well. In particular, we are able to detect a combination of certain delays and actions as causal for system behaviors. In contrast to the approaches of Kölbl et al., the handled system behaviors are in our work also not limited to reachability properties.

Mari et al. propose an explaining technique for the violation of safety properties in real-time systems [56]. The approach is fully based on their corresponding work on explaining discrete systems [34]: Given a real-time system, they construct in a very first step a finite discrete abstraction of the semantic transition system, consisting of action and delay transitions (cf. Definition 2.4). Then, their in [34] developed effective choice explanations for discrete event systems are applied to this abstraction. Their effective choice explanations are, however, not concluded from counterfactual reasoning in its original Halpern and Pearl flavor, but rather computed, based on a so-called level of choice function, describing which transitions have brought the system closer to the violation.

→ Definition 2.4, p. 15

While similar to our work, Mari et al. can identify both actions as well as delays (and also arbitrary combinations of those events) to be causal for a certain system behavior, the approaches differ in particular in the kind of causality concept underlying the causal reasoning. Furthermore, we present an approach defining the counterfactual reasoning fully on the side and in terms of real-time systems and do not start with a translation to the discrete setting.

Chapter 7

Future Work

In this chapter, we want to sketch several possible lines of future research resulting from questions that were raised as a result of our work. We thereby divide the discussion in two parts and comment firstly in Section 7.1 on possible future theoretical work and address secondly in Section 7.2 possible upgrades and further developments of the causality tool.

→ Section 7.1, p. 111
→ Section 7.2, p. 115

7.1 Future Theoretical Work

A first steadily ongoing task arises from Joseph Halpern's words already quoted in Chapter 1. With regard to causality notions in general he remarked:

→ Chapter 1, p. 1

"The way you show your definition is good is to show that – gee! – look at how well it does it all in all the examples."

And goes on to say:

"And that works fine until somebody comes along and constructs an example to show your definition maybe wasn't as good as you thought it was..."

– Joseph Y. Halpern[1] –

It remains, therefore, also in our setting desirable to continue considering further and new kinds of examples, to investigate how our notions handle these and whether they behave in accordance with our intuition, and, in the case of detected counter-intuitive phenomena, to further improve the notions. In this regard however, one has to point out that refining the notions to capture a further class of scenarios without deteriorate on currently captured examples seems in particular in our setting hard, if not sometimes

[1] Recording of his talk at AAAI 2018: *Actual Causality: A Survey* [39]

even impossible. In fact, we made in some sense such an observation already in the course of this thesis: As discussed in large detail in Section 3.2, neither the introduced delay nor the alternative timestamp perspective can suitably capture different kinds of everyday examples as a singular approach.

→ Section 3.2, p. 46

Therefore, it might, however, be particularly interesting to think about ways to combine the two perspectives into one notion. The most natural idea for this would be to investigate what happens when choosing as the whole set of events $\mathcal{E}$ the union of delay events, timestamp events, and unchanged action events, i.e., $\mathcal{E} := \mathcal{DE} \dot\cup \mathcal{TE} \dot\cup \mathcal{AE}$ (cf. Definitions 3.1 and 3.8). Besides the technical aspect that we would need to adapt our definitions of delay and timestamp events in order to obtain indeed disjoint unions of $\mathcal{DE}$ and $\mathcal{TE}$, the first occurring question would again be how to construct the counterfactual trace automaton for these combined sets of events. We think that simply combining the ideas for the constructions in the delay as well as in the timestamp perspective into one construction should result in an again faithful and meaningful definition of a counterfactual trace automaton. We have, however, not yet carried out and investigated this construction in detail.

→ Definitions 3.1, 3.8, p. 24 and 49

Furthermore, one might want to rethink what qualifies as minimal causes in this approach. Probably, the minimality condition should be sharpened in a way, that not only all the strict subsets of a cause $\mathcal{C}$ are checked, but also those sets with less elements than $\mathcal{C}$ for which there exists for each of their delay and timestamp events a delay or a timestamp event with the same position in $\mathcal{C}$ (the action events should of course still be completely contained in $\mathcal{C}$). Only this might enable an analysis of scenarios like the one of the agricultural field (cf. Example 3.2.1) in which the set with multiple delay events is not identified as minimal cause but only the singleton timestamp event set.

→ Example 3.2.1, p. 47

Another, much more "radical" idea for extending our notion of causality in real-time systems is an approach, that allows causal timestamp events to change the position, at which its corresponding action is performed, in the order of action events. Recall again Example 3.2.1 of the agricultural field and imagine now a scenario in which the farmer does everything as required for a successful harvest, except for the fact that he forgets to sow the field in March but does it only in October[2]. Intuitively one could blame the incorrect time, at which the field was sown, to be a cause for the failed harvest, as only changing the month of sowing from October to March would result in a successful harvest.

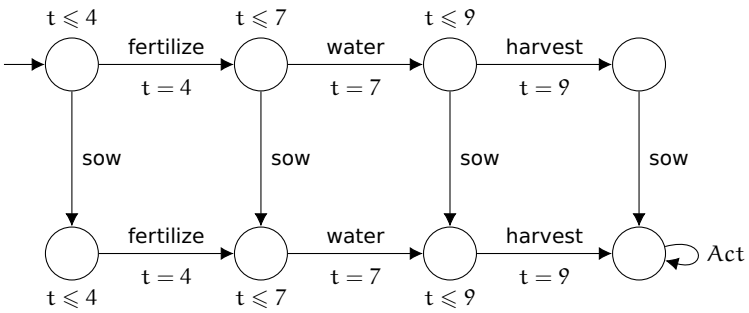
→ Example 3.2.1, p. 47

Our notions of causality (both the delay as well as the timestamp perspective) will, however, not identify the time of sowing as singleton cause as they do not allow to change the order of the performed actions. The actions in possible counterfactual traces would still be in the "incorrect" order, i.e., for this example in the order fertilize, water, harvest,

[2] For whatever reason he still sows the field in October, coming up with a reasonable everyday scenario is by this footnote also declared as future work.

sow. Changing the order of actions to the "correct" one sow, fertilize, water, harvest would require in our causality notions to make all actions causal, a large deviation from intuition.

An idea for the construction of a counterfactual trace automaton in the timestamp perspective that allows changing the position of timed actions with causal timestamps could work as follows: While the non-causal timestamps, and thereby the timing of their respective actions, are again enforced by the known construction, actions with causal timestamps can be executed arbitrarily in between. For the above example, this could be expressed with the following counterfactual trace automaton for the set of events $\mathcal{C} := \{(10, 4)\}$ (i.e, the singleton cause only containing the too late time of sowing):



In this counterfactual trace automaton, we can indeed find the desired counterfactual timestamp trace $\langle 3, \mathsf{sow}\rangle\langle 4, \mathsf{fertilize}\rangle\langle 7, \mathsf{water}\rangle\langle 9, \mathsf{harvest}\rangle$. In general for sets of events with n causal timestamps, the counterfactual trace automaton would have the shape of an $(1 + n)$-dimensional cuboid: Along the first dimension, the timed actions with non-causal timestamp are enforced (speaking in terms of a usual coordinate system, this corresponds in the above depiction to the x-direction), the execution of the timed actions with causal timestamp is then allowed in one of the remaining n dimensions (in the above depiction the execution of the only action with causal timestamp sow is handled in the y-direction).

For both of the sketched approaches, the combination of delay and timestamp events as well as the idea of changing the order of actions for causal timestamp events, it would be of increased interest to find a way to process not only finite but also infinite timestamp traces. Therefore, the entry point might be to look again at the possibility of representing infinite timestamp traces in a lasso-shaped form as well, and to then exploit, for instance, the concept of repeated unrolling of the lasso-part up to the positions of interest for the causal analysis. However, we fear that this approach of (possibly infinitely many) unrollings could break the decidability and computability of our notions.

Talking about considering further or different objects as the events in a system execution, it might be suitable in our real-time setting to deviate from starting with a run containing concrete delays, but to look at symbolic trace descriptions like timed diagnostic traces and work with their timing constraints (cf. the remarks on the work of Kölbl et al. [50] in Section 6.2). In the flavor of setting up our counterfactual simulation, interval constraints could in the counterfactual trace automaton then probably be enforced again by describing them in terms of guards, alternatives for causal constraints could then again be allowed by omitting these guards. Going even further, one could follow the idea of specifying events and, therefore, causes also in the real-time setting fully in terms of temporal formulas, which, as mentioned in Chapter 6, could already be demonstrated for other settings in previous works [22, 31].

→ Section 6.2, p. 108

→ Chapter 6, p. 105

The enhancement, we definitely wish to accomplish the most, is to extend our notions to networks of timed automata. It thereby seems for our run- and trace- based approach necessary to carefully develop a notion of projected traces, that is, given a global trace in the full network we want to extract the sequence of timed actions performed locally by the individual agents. Inspiration on how to obtain and work with such projections might be found in the work of Datta et al.: They compute in a discrete setting causal actions in concurrent systems and consider to this end projections of the global log to the individual threads [25].

Based on such a notion for projected traces, we might then be able to formalize the counterfactual reasoning agent-wise: Given a network $\mathsf{TA}_1 \parallel \ldots \parallel \mathsf{TA}_n$ of timed automata $\mathsf{TA}_1, \ldots, \mathsf{TA}_n$ (whereby "$\parallel$" denotes the usual parallel composition of automata), we consider agent-specific sets of events $\mathcal{C}_1, \ldots, \mathcal{C}_n$, that is, one set of events particular for each of the agents. Given now a trace ξ in the network, we firstly compute the projected traces $\xi_1, \ldots, \xi_n$ and use these then to construct the counterfactual trace automata $\mathsf{TA}_{\xi_1}^{\mathcal{C}_1}, \ldots, \mathsf{TA}_{\xi_n}^{\mathcal{C}_n}$ again agent-wise. Exactly following the approach in this work, we would then search for a counterfactual run in the network composed of the intersections, i.e., $\mathsf{TA}_1 \cap \mathsf{TA}_{\xi_1}^{\mathcal{C}_1} \parallel \ldots \parallel \mathsf{TA}_n \cap \mathsf{TA}_{\xi_n}^{\mathcal{C}_n}$. This notion would then allow to conclude events like "the fourth delay of 7 in the second agent and the action β as first action in the third agent" as causes for a certain behavior of the network.

Extending our work to networks might also improve on a further limitation: As discussed in Section 3.3, our allowed contingencies are relatively coarse-grained leading to the inability of our actual causality notion to handle certain kinds of preemption scenarios. The above proposed approach for lifting the causality notions to networks might, however, yield the possibility of taking contingencies in a more fine-grained way as the contingency automata could be constructed, analogously to the counterfactual trace automata, agent-wise and not globally. Therefore, it might then be possible to fix the configuration of only certain agents and thereby especially the locations of only certain agents at the respective points in the run.

→ Section 3.3, p. 55

A last line of future theoretical research, we want to mention, is the work on further optimizations of the presented algorithms. While we see no proper room for improvements in the process of cause checking, the computation of causes can possibly be further optimized: By now, we use the monotonicity of the **CF**-conditions only to gain knowledge in one direction, by excluding in the case that a set of events is identified as cause all its supersets from the computation. While this speeds things up, the algorithm still has to consider all the sets not fulfilling **CF** – a quite expensive task, particularly in scenarios with a small number of causes compared to the length of the run.

Exploiting monotonicity in the converse direction as well could improve on this issue: If we detect a large set of events to not fulfill the **CF**-condition, monotonicity implies that neither do all its subsets and can, hence, be excluded from the computation as well. Therefore, an algorithm working not solely in a bottom-up or top-down approach, but proceeds in a back-and-forth like manner to alternatingly propagate knowledge both from small sets to their supersets as well as from large sets to their subsets, could significantly improve the performance of cause computations.

7.2 Desired Upgrades of the Causality Tool

Heading to the developed causality tool, we wish as a natural first goal to keep pace with possible future refinements of the causality definitions, that is, that we want further developed causality notions to be implemented as well in the tool.

In particular, we would like to extend the tool to cover based on a respective causality notion also networks of timed automata. This would enable us not only to test the tool more extensively on examples from the literature, but would also be way more in accordance with the contemplated modeling of systems in Uppaal. Challenges in implementing the approach sketched above might thereby especially arise in the extraction of projected traces from the global system execution as well as in the handling of synchronizations between processes.

In general for causality notions considered in the future, their suitability for practical implementation will remain a compelling criterion.

Moreover, we also desire to overcome current shortcomings of the tool. As an ultimate goal, we would, for instance, like to provide the functionality of fully explainable model checking, that is, that we can decide a specification on a system and return in the case of a violation a counterexample together with an explanation (e.g., again by highlighting causal events in the counterexample). By now, this is if at all only possible by manually composing functionalities from Uppaal and our tool. For this end, Uppaal lacks, however, as discussed in Chapter 5 of the possibility to reliably return concrete counterexamples, but provides in certain scenarios only a sequence of clock constraints. Hence, we desire to find a way of extracting the concrete delays of run from

→ Chapter 5, p. 83

those constraints, maybe achievable based on ideas from existing works, e.g., on finding causal delay values and ranges [50].

A different solution for obtaining concrete counterexamples might also be to use another model checker for the verification process that offers this functionality. One of the tools we have in mind for this purpose is the TACK tool developed by Menghi et al. [57]. As this model checker covers, in addition, full MITL, an exchange could also eliminate a further limitation of the current tool version, namely, that it is restricted for the specification of effects to Uppaal's language expressing as discussed only a small fragment of MITL.

Lastly, it would be highly desirable to find ways to improve the runtime of the developed tool, especially for the computation of causes. Besides the aforementioned algorithmic optimizations, in particular tackling the problem of the blow up in the Uppaal-formula could contribute to this goal. While it is unclear, whether this can be done at all for automata modeled in Uppaal, it might also for this reason become necessary to consider software alternatives.

Chapter 8

Summary

Guided by the aim of explaining the behavior of real-time systems, this thesis developed different notions of counterfactual causality in the flavor of Halpern and Pearl in the real-time setting. As a major contribution, our causality notions consider both the performed actions and the real-time behavior as potential causal events. We argued for the usefulness and meaningfulness of our definitions by discussing their application to numerous examples, which demonstrated how the causality concepts yield desired explanations. We further supported our notions by showing them to fulfill commonly required properties of causality.

The key for setting up formal counterfactual reasoning in the real-time setting was to represent the possible counterfactual system behavior in terms of automata. Using counterfactual trace automata, we can in particular capture the infinite number of possible alternatives to real-time events without losing the decidability of our notions. Instead, the automata-based approach enables us to define real-time causes in terms of decidable model-checking problems. Extending on the basic notion of but-for causality, we added contingencies to the causal reasoning resulting in a real-time notion of actual causality that can solve (at least for some cases) the problem of preemption.

We observed that adopting a different perspective on time results in differing causal analyzes. Thereby, neither the delay perspective nor the timestamp perspective revealed themselves to be the generally better-performing notion, both showed their suitability for a precise causal analysis in particular scenarios. Also, apart from the particular perspective on time, the wide range of possible real-time examples seems difficult to be captured in one universal causality notion.

It was crucial for counterfactual causality in real-time systems to analyze not only finitely large or finitely many counterfactual simulations. Hence, we work in a causal theory with infinitely many variables that do, moreover, partially range over an infinite domain. Nonetheless, we are still able to establish causes to be decidable and computable and could as well analyze the complexity of those problems. This gives rise

to new decidability and complexity results for causality in infinite theories as previous works addressing decidability and complexity focus only on finite causal models.

The genesis of this thesis was characterized by a persistent balance between providing a clean theoretical development on the one side and its practical implementability on the other side. More than once, we had to decide on details in the notions that either would have led to losses in the theoretical contribution, or that would have evoked additional issues in Uppaal and the developed tool. Nonetheless, we were able to set up a rather clean, coherent, and expressive theory on causality in real-time systems, that, at the same time, could be implemented with Uppaal for practical applications.

The tool features automated explanations for counterexamples of Uppaal specifications, for instance, by computing causal events in a given violating run. For being deployed in real practical contexts, we see a particular need in improving on the yet relatively vast restrictions of handleable systems and specifications and to address the enormous runtime explosion, especially in the computation of causes. The latter challenge might be approached by further research on the algorithmic side as well as optimizations in the tool, for instance, by getting rid of the blowup in the model-checked formula.

Despite its limitations, the causality tool is useful for basic experiments on real-time causality, which in fact already helped us at some points to improve our causal understanding of real-time systems. Furthermore, the designed data structures and implemented functionalities the tool relies on might form a good technical starting point for further attempts in gaining automated real-time system explanations. The importance of this will only continue to increase.

Bibliography

[1] Pyuppaal library webpage. URL https://pypi.org/project/pyuppaal/1.0.0/.

[2] Uppaal webpage, . URL uppaal.org.

[3] Uppaal documentation, . URL https://docs.uppaal.org/.

[4] Uppaal language reference, . URL https://docs.uppaal.org/language-reference/requirements-specification/symb_queries/.

[5] Uppaal language reference, . URL https://docs.uppaal.org/gui-reference/symbolic-simulator/symbolic-traces/.

[6] Kjersti Aas, Martin Jullum, and Anders Løland. Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artificial Intelligence*, 298:103502, 2021. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103502. URL https://www.sciencedirect.com/science/article/pii/S0004370221000539.

[7] Gadi Aleksandrowicz, Hana Chockler, Joseph Y. Halpern, and Alexander Ivrii. The computational complexity of structure-based causality, 2014.

[8] Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 8–22, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48683-1.

[9] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, jan 1996. ISSN 0004-5411. doi: 10.1145/227595.227602. URL https://doi.org/10.1145/227595.227602.

[10] Uwe Aßmann, Christel Baier, Clemens Dubslaff, Dominik Grzelak, Simon Hanisch, Ardhi Putra Pratama Hartono, Stefan Köpsell, Tianfang Lin, and Thorsten Strufe.

Tactile computing: Essential building blocks for the Tactile Internet, page 293–317. Academic Press, 2021. ISBN 978-0-12-821343-8. doi: 10.1016/B978-0-12-821343-8.00025-3. 46.23.01; LK 01.

[11] Christel Baier, Clemens Dubslaff, Florian Funke, Simon Jantsch, Rupak Majumdar, Jakob Piribauer, and Robin Ziemek. From verification to causality-based explications. *CoRR*, abs/2105.09533, 2021. URL `https://arxiv.org/abs/2105.09533`.

[12] Christel Baier, Florian Funke, Simon Jantsch, Jakob Piribauer, and Robin Ziemek. Probabilistic causes in markov chains. *CoRR*, abs/2104.13604, 2021. URL `https://arxiv.org/abs/2104.13604`.

[13] Thomas Ball and Orna Kupferman. Vacuity in testing. In Bernhard Beckert and Reiner Hähnle, editors, *Tests and Proofs*, pages 4–17, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-79124-9.

[14] Thomas Ball, Mayur Naik, and Sriram K. Rajamani. From symptom to cause: Localizing errors in counterexample traces. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '03, page 97–105, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136285. doi: 10.1145/604131.604140. URL `https://doi.org/10.1145/604131.604140`.

[15] Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient detection of vacuity in actl formulas. In Orna Grumberg, editor, *Computer Aided Verification*, pages 279–290, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69195-2.

[16] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard Trefler. Explaining counterexamples using causality. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 94–108, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-02658-4.

[17] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003. doi: 10.1007/978-3-540-27755-2_3. URL `https://doi.org/10.1007/978-3-540-27755-2_3`.

[18] Georgiana Caltais, Sophie Linnea Guetlein, and Stefan Leue. Causality for general LTL-definable properties. *Electronic Proceedings in Theoretical Computer Science*, 286:

1–15, jan 2019. doi: 10.4204/eptcs.286.1. URL https://doi.org/10.4204%2Feptcs.286.1.

[19] Hana Chockler and Joseph Y. Halpern. Responsibility and blame: a structural-model approach. *CoRR*, cs.AI/0312038, 2003. URL http://arxiv.org/abs/cs/0312038.

[20] Hana Chockler, Joseph Y. Halpern, and Orna Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Logic*, 9(3), jun 2008. ISSN 1529-3785. doi: 10.1145/1352582.1352588. URL https://doi.org/10.1145/1352582.1352588.

[21] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45047-4.

[22] Norine Coenen, Raimund Dachselt, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Tom Horak, Niklas Metzger, and Julian Siber. Explaining hyperproperty violations, 2022.

[23] Norine Coenen, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Niklas Metzger, and Julian Siber. Temporal causality in reactive systems. In Ahmed Bouajjani, Lukás Holík, and Zhilin Wu, editors, *Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings*, volume 13505 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2022. doi: 10.1007/978-3-031-19992-9_13. URL https://doi.org/10.1007/978-3-031-19992-9_13.

[24] Dennis Dams and Orna Grumberg. *Abstraction and Abstraction Refinement*, pages 385–419. Springer International Publishing, Cham, 2018. ISBN 978-3-319-10575-8. doi: 10.1007/978-3-319-10575-8_13. URL https://doi.org/10.1007/978-3-319-10575-8_13.

[25] Anupam Datta, Deepak Garg, Dilsun Kaynar, Divya Sharma, and Arunesh Sinha. Program actions as actual causes: A building block for accountability. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 261–275, 2015. doi: 10.1109/CSF.2015.25.

[26] Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016. doi: 10.1109/SP.2016.42.

[27] Alexandre David and Kim Larsen. A tutorial on uppaal 4.0. 01 2006.

[28] Henning Dierks, Sebastian Kupferschmid, and Kim Larsen. Automatic abstraction refinement for timed automata. pages 114–129, 10 2007. ISBN 978-3-540-75453-4. doi: 10.1007/978-3-540-75454-1_10.

[29] Ellery Eells. *Probabilistic Causality*. Cambridge Studies in Probability, Induction and Decision Theory. Cambridge University Press, 1991. doi: 10.1017/CBO9780511570667.

[30] Thomas Eiter and Thomas Lukasiewicz. Complexity results for structure-based causality. *Artificial Intelligence*, 142:53–89, 11 2002. doi: 10.1016/S0004-3702(02)00271-0.

[31] Bernd Finkbeiner and Julian Siber. Counterfactuals modulo temporal logics. In Ruzica Piskac and Andrei Voronkov, editors, *LPAR 2023: 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, June 4-9, 2023*, volume 94 of *EPiC Series in Computing*, pages 181–204. EasyChair, 2023. doi: 10.29007/qtw7. URL `https://easychair.org/publications/paper/sWZw`.

[32] Gregor Gössler and Daniel Le Métayer. A General Trace-Based Framework of Logical Causality. Research Report RR-8378, INRIA, October 2013. URL `https://inria.hal.science/hal-00873665`.

[33] Gregor Gössler, Daniel Le Métayer, and Jean-Baptiste Raclet. Causality analysis in contract violation. In Howard Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Roşu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Runtime Verification*, pages 270–284, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-16612-9.

[34] Gregor Gössler, Thomas Mari, Yannick Pencolé, and Louise Travé-Massuyès. Towards Causal Explanations of Property Violations in Discrete Event Systems. In *DX'19 - 30th International Workshop on Principles of Diagnosis*, pages 1–8, Klagenfurt, Austria, November 2019. URL `https://inria.hal.science/hal-02369014`.

[35] Alex Groce. Error explanation with distance metrics. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 108–122, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24730-2.

[36] Gregor Gössler and Jean-Bernard Stefani. Causality analysis and fault ascription in component-based systems. *Theoretical Computer Science*, 837:158–180, 2020. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2020.06.010. URL `https://www.sciencedirect.com/science/article/pii/S0304397520303510`.

[37] Joseph Y. Halpern. A modification of the halpern-pearl definition of causality. *CoRR*, abs/1505.00162, 2015. URL http://arxiv.org/abs/1505.00162.

[38] Joseph Y. Halpern. *Actual Causality*. MIT Press, 2016.

[39] Joseph Y. Halpern. Actual Causality: A Survey: Joseph Halpern, 2018. URL https://www.youtube.com/watch?v=hXnCX2pJ0sg.

[40] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part i: Causes. *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI 2001)*, pages 194–202, 2001.

[41] 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. ISSN 00070882, 14643537. URL http://www.jstor.org/stable/3541870.

[42] Joseph Y. Halpern and Spencer Peters. Reasoning about causal models with infinitely many variables. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):5668–5675, Jun. 2022. doi: 10.1609/aaai.v36i5.20508. URL https://ojs.aaai.org/index.php/AAAI/article/view/20508.

[43] Mark Harman and Robert Hierons. An overview of program slicing. *Software Focus*, 2, 12 2001. doi: 10.1002/swf.41.

[44] Susan B. Horwitz, Thomas Reps, and Dave Binkley. Interprocedural slicing using dependence graphs. *SIGPLAN Not.*, 23(7):35–46, jun 1988. ISSN 0362-1340. doi: 10.1145/960116.53994. URL https://doi.org/10.1145/960116.53994.

[45] Yatin Hoskote, Timothy Kam, Pei-Hsin Ho, and Xudong Zhao. Coverage estimation for symbolic model checking. pages 300–305, 02 1999. ISBN 1-58113-092-9. doi: 10.1109/DAC.1999.781330.

[46] David Hume. *Philosophical Essays Concerning Human Understanding*. Andrew Millar, 1748. URL https://books.google.de/books?id=LB4VAAAAQAAJ.

[47] Manu Jose and Rupak Majumdar. Cause clue clauses: Error localization using maximum satisfiability, 2011.

[48] Samantha Kleinberg and Bud Mishra. The temporal logic of causal structures. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 303–312, Arlington, Virginia, USA, 2009. AUAI Press. ISBN 9780974903958.

[49] Martin Kölbl, Stefan Leue, and Thomas Wies. *Clock Bound Repair for Timed Systems*, pages 79–96. 07 2019. ISBN 978-3-030-25539-8. doi: 10.1007/978-3-030-25540-4_5.

[50] Martin Kölbl, Stefan Leue, and Robert Schmid. *Dynamic Causes for the Violation of Timed Reachability Properties*, pages 127–143. 08 2020. ISBN 978-3-030-57627-1. doi: 10.1007/978-3-030-57628-8_8.

[51] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 5(1): 1–11, jan 1987. ISSN 0734-2071. doi: 10.1145/7351.7352. URL https://doi.org/10.1145/7351.7352.

[52] Florian Leitner-Fischer and Stefan Leue. Causality checking for complex system models. 01 2013. doi: 10.1007/978-3-642-35873-9_16.

[53] David Lewis. Causation. *Journal of Philosophy*, 70(17):556–567, 1973. doi: 10.2307/2025310.

[54] David K. Lewis. *Counterfactuals*. Cambridge, MA, USA: Blackwell, 1973.

[55] Jane W. S. Liu. *Real-Time Systems*. Always Learning. Pearson Education, 2006. ISBN 9788177585759. URL https://books.google.de/books?id=ZVd6U_DXnSAC.

[56] Thomas Mari, Thao Dang, and Gregor Gössler. Explaining Safety Violations in Real-Time Systems. Research Report RR-9420, INRIA ; Verimag, Université Grenoble Alpes, September 2021. URL https://inria.hal.science/hal-03348046.

[57] Claudio Menghi, Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. Model checking mitl formulae on timed automata: A logic-based approach. *ACM Trans. Comput. Logic*, 21(3), apr 2020. ISSN 1529-3785. doi: 10.1145/3383687. URL https://doi.org/10.1145/3383687.

[58] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(87)90062-2. URL https://www.sciencedirect.com/science/article/pii/0004370287900622.

[59] Lloyd S. Shapley. *A Value for N-Person Games*. RAND Corporation, Santa Monica, CA, 1952. doi: 10.7249/P0295.

[60] Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. *CoRR*, abs/1908.08474, 2019. URL http://arxiv.org/abs/1908.08474.

[61] Patrick Suppes. *A Probabilistic Theory of Causality*. Amsterdam: North-Holland Pub. Co., 1968.

[62] Giulia Vilone and Luca Longo. Notions of explainability and evaluation approaches for explainable artificial intelligence. *Information Fusion*, 76:89–106, 2021. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2021.05.009. URL https://www.sciencedirect.com/science/article/pii/S1566253521001093.

[63] Chao Wang, Zijiang Yang, Franjo Ivančić, and Aarti Gupta. Whodunit? causal analysis for counterexamples. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis*, pages 82–95, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-47238-4.

[64] Shaohui Wang, Anaheed Ayoub, BaekGyu Kim, Gregor Gössler, Oleg Sokolsky, and Insup Lee. A causality analysis framework for component-based real-time systems. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification*, pages 285–303, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40787-1.

[65] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4): 352–357, 1984. doi: 10.1109/TSE.1984.5010248.