ISSN 2186-7437

NII Shonan Meeting Report

No. 2019-139

Causal Reasoning in Systems

Gregor Gössler
Stefan Leue
Shin Nakajima

June 24–27, 2019

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Causal Reasoning in Systems

Organizers:

Gregor Gössler
Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Stefan Leue
University of Konstanz, Konstanz, Germany

Shin Nakajima
NII, Japan

June 24–27, 2019

Abstract

The discussion of causality, which has its roots in the philosophy of sciences, has recently become the subject of discussion in relation to IT systems, in particular software, hardware, and cyber-physical systems. Determining causalities is of essential importance when asserting the safety of a system (what can cause a hazardous situation to occur?), when analyzing failures of critical systems (why and how did an accident occur?) or when performing fault localization in hardware or software, amongst others. The goal of this seminar was to gain and deepen an understanding of the available means to reason about causality, the approaches that various disciplines inside computer science as well as in adjacent fields are using to determine causality, and what notions of causality need to be developed in order to deal with changing paradigms of computing.

1 Meeting Schedule

	Monday	Tuesday	Wednesday	Thursday
09:00-09:30	movie welcome and introduction of participants	Stefan Leue *Causality in Models of Computation* 9:00 - 9:40	Alexander Pretschner *Efficiently Computing Halpern-Pearl Causality for Binary Models with SAT Solving and ILP*	break-out consolidation of results
09:30-10:00	overview of meeting	Gregor Goessler *Causality Analysis and Fault Ascription from First Principles* 9:40 - 10:20	break-out	presentation break-out results
10:00-10:30	meeting structure case studies		break-out	presentation break-out results
10:30-11:00	coffee break	coffee break 10:30 - 10:50	coffee break	coffee break
11:00-11:30	Georgiana Caltais *Causality for (software defined) networks*	Norine Coenen *Causality & Hyperproperties* 10:50 - 11:30	break-out	closing - book prop.
11:30-12:00	Ruzica Piskac *Using SAT Solvers to Prevent Causal Failures in the Cloud*	Vitaliy Batusov *Actual Causality in Situation Calculus* 11:30 - 12:10	break-out	closing
12:00-12:30				
12:30-13:00				
13:00-13:30				
13:30-14:00	Armen Aghasaryan *Root Cause Analysis (RCA) for Future Networks: Technology, Vision and Challenges*	group photo	excursion	
14:00-14:30	Mohammad Mousavi *Using Model-Based Testing for Doping Detection in Cyber-Physical Systems*	Ebru Aydin Gol *Cause Mining with STL* *14:00 - 14:40*	excursion	
14:30-15:00	Richard Trefler *Parameterized Protocol Analysis and Causal Reasoning*	Samantha Kleinberg *Causality and Decision-Making* *14:40 - 15:20*	excursion	
15:00-15:30			excursion	
15:30-16:00	coffee break	coffee break 15:20 - 15:50	excursion	
16:00-16:40	Thomas Wies *Finding Minimum Type Error Sources*	Shoji Yuen *Automating Time-series Safety Analysis for Automotive Control Systems using Weighted Partial Max-SMT*	excursion	
16:40-17:20	Alexander Pretschner *Causal Models*	Mario Gleirscher *Risk Structures: Concepts and Purpose*	excursion	
17:20-18:00	Ashish Gehani *Causal Inference for Attack Investigation*	topics for break-out	excursion	

2 Overview of Talks

2.1 Foundations of Causality

Causality in Models of Computation

Stefan Leue, University of Konstanz

The automated discovery and documentation of causal structures is of great importance in Software and Systems Engineering. It helps in identifying and localizing design faults, in establishing safety cases for safety-critical systems, and in explaining and repairing system design and software errors. I will discuss actual cause analysis in the context of models of computation in general, and consider transition systems as a particular model of computation for concurrent system. This is particularly relevant in order to support model-based design as well as automated, algorithmic approaches towards actual cause analysis. We classify causes into dynamic causes, which relate to choices made by the environment or a scheduler during runtime, and static causes, which manifest themselves in syntactic features or the model or program that is being considered.

I will review work on Causality Checking [4], which aims at identifying ordered sequences of events as dynamic actual causes for the violation of reachability properties. Causality checking is based on a counterfactual notion of cause and is an adaptation of the Halpern-Pearl [1] structural model of actual causation to the transition system model of computation that our analysis is based on. The causes in this case are dynamic since, for instance, the dynamically scheduled order of concurrent events may be determined to be causal for the property violation. I will present a tool called QuantUM, which implements Causality Checking, as well as case studies from the automotive domain where we applied Causality Checking in order to synthesize Fault Trees explaining causes for the violation of functional safety properties [2]. In order to illustrate static actual causes I will then discuss the identification of real-time bounds that are determined to be causal for the violation of timed reachability properties in Timed Automata models [3]. The analysis enables a partial MaxSMT based automated repair of the identified causes. I will conclude by discussing research challenges in static and dynamic actual cause analysis for models of computation.

References

[1] Joseph Y. Halpern. *Actual Causality.* MIT Press, 2016.

[2] Martin Kölbl and Stefan Leue. Automated functional safety analysis of automated driving systems. In *FMICS*, volume 11119 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2018.

[3] Martin Kölbl, Stefan Leue, and Thomas Wies. Clock bound repair for timed systems. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2019.

[4] Florian Leitner-Fischer and Stefan Leue. Causality checking for complex system models. In *VMCAI*, volume 7737 of *Lecture Notes in Computer Science*, pages 248–267. Springer, 2013.

Logical Foundations for Actual Causality

Vitaliy Batusov and Mikhail Soutchanski, York University, CA

We develop a novel definition of actual cause in the context of situation calculus (SC) action theories. Situation calculus [1] is a many-sorted dialect of second-order logic which captures the notion of action and change by modelling all possible world histories as an infinite tree of situations stemming from a common root. Each situation corresponds to a sequence of actions, which are the sole source of change in the world, and the domain of application is described by situation-dependent functions and predicates. A *basic action theory* (BAT, [2]) is a well-behaved SC theory containing action precondition axioms and successor state axioms.

To describe a causal scenario, we use *causal settings* of the form $\mathcal{C} = \langle \mathcal{D}, \sigma, \phi \rangle$ where $\mathcal{D}$ is a BAT describing the initial state and the general dynamics of the world, σ is a ground situation describing a *complete* narrative of events which have transpired in the world, and ϕ is a first-order sentence describing the effect of interest. In this framework, actual causes of the effect ϕ, if any, are to be found among the actions of σ. We recognize two distinct causal roles that actions may take. For one, an action may realize the effect of interest; that is, it may change the truth value of ϕ from $false$ to $true$; we call such actions *achievement causes*. Secondly, an action may prevent an effect from being lost; we call such actions *maintenance causes*. *Actual causes* are an umbrella term covering all combinations of achievement and maintenance.

We formalize achievement causes in two steps. First, we say that if an action $\alpha \in \sigma$ triggers ϕ to become $true$ and there is no action in σ after α that changes the truth value of ϕ back to $false$, then α is deemed an achievement cause in the setting $\mathcal{C}$. Second, we appeal to the mechanism of regression, afforded to SC by the special form of its successor-state axioms; namely, to the fact that it is trivial to obtain, from a sentence ϕ about the situation $do(\alpha', s)$, a logically equivalent sentence $\rho[\phi, \alpha]$ about the situation s. Despite being logically equivalent with the situation terms fixed as shown, ϕ and $\rho[\phi, \alpha]$ are generally different formulas which are realized at different points of the narrative — in fact, if α is a cause of ϕ, then $\rho[\phi, \alpha]$ is guaranteed to be achieved prior to it. Thus, if α is an achievement cause in $\langle \mathcal{D}, \sigma, \phi \rangle$ such that $do(\alpha, \sigma') \sqsubseteq \sigma$, then $\rho[\phi, \alpha](\sigma')$ is a necessary and sufficient condition for achieving ϕ via α. We say that the achievement cause of the new setting $\langle \mathcal{D}, \sigma', \rho[\phi, \alpha] \rangle$ is also an achievement cause of the original setting $\mathcal{C}$. Repeating this method over each such new setting until the narrative σ runs out, we can uncover the *achievement causal chain* of the setting $\mathcal{C}$ — that is, a chain of actions selected from σ, each of which constructively contributes to the total effect ϕ.

We define maintenance causes using an auxiliary notion of *threats* — actions τ which occur in the narrative and have the hypothetical capacity of destroying the effect ϕ, but do not have that effect in the given setting. A maintnenance cause of $\mathcal{C}$ is then the achievement cause of the setting $\langle \mathcal{D}, \sigma'', \rho[\phi, \tau] \rangle$, where $do(\tau, \sigma'') \sqsubseteq \sigma$. The criterion for identifying threats is in essence a counterfactual.

We further observe that each new causal setting obtained via the discovery of achievement or maintenance causes of $\mathcal{C}$ can be subject to the same analysis as the top-level setting $\mathcal{C}$. We use the generic term *actual cause of* $\mathcal{C}$ to refer to all causes discovered by top-level analysis as well as all causes discovered by recursively applying the top-level analysis to each new causal setting identified in the process. With this approach, we can successfully identify non-trivial actual causes of complex conditions expressed in first-order logic. We show that the state-of-the-art definition of actual cause due to [3] can be effectively simulated in our approach. Using examples, we show that long-standing disagreements between alternative definitions of actual causality can be mitigated by faithful SC modelling of the domains.

References

[1] John McCarthy and Patrick J Hayes, "Some philosophical problems from the standpoint of artificial intelligence". *Readings in artificial intelligence*. Morgan Kaufmann, 1981. 431–450.

[2] Raymond Reiter, *Knowledge in action: logical foundations for specifying and implementing dynamical systems*, MIT press, 2001.

[3] Joseph Y Halpern. *Actual causality*. MIT Press, 2016.

Causality Analysis and Fault Ascription from First Principles

Gregor Gössler, INRIA

We present a general framework for counterfactual analysis and fault ascription in component-based concurrent systems [4]. Our framework uses configuration structures [5] as a general semantical model to handle truly concurrent executions, partial and distributed observations in a uniform way. We define a set of formal requirements a counterfactual function should satisfy in order to enable the construction of meaningful causality analyses for such systems. These requirements will serve as firm ground for guiding the definition of concrete analyses and reasoning about their properties. This approach contrasts with current practice of evaluating definitions of counterfactual causality *a posteriori* on a set of toy examples [2, 3].

Our formal requirements include well-behavedness under varying observability (better observability improves precision of the analysis) and incrementality (adding new observations to a log improves precision). The first requirement is crucial for causality analyses to work on abstractions. The second one is essential for incremental and on-the-fly analysis. Furthermore, we state expected properties for the behavior of counterfactual functions under several notions of refinement.

We state many of our results for hyperproperties [1]. Hyperproperties have been shown necessary to deal with requirements such as security and quality of service, and taking them into account in our framework ensures our causality analysis can deal with violation of these kinds of properties as well. We discuss the use of analyzing the causation (of violation) of hyperproperties on a motivating example. This is joint work with Jean-Bernard Stefani.

References

[1] M. R. Clarkson and F. B. Schneider. Hyperproperties. Journal of Computer Security 18(6), 2010.

[2] C. Glymour, D. Danks, B. Glymour, F. Eberhardt, J. Ramsey, R. Scheines, P. Spirtes, C. Teng, and J. Zhang. Actual causation: a stone soup essay. Synthese 175(2), 2010.

[3] G. Gössler, O. Sokolsky, and J.-B. Stefani. Counterfactual causality from first principles?, in: Proceedings 2nd International Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies (CREST'17), EPTCS 259, 2017.

[4] G. Gössler and J.-B. Stefani. Causality Analysis and Fault Ascription in Component-Based Systems. Research Report RR-9279, INRIA, 2019. `https://hal.inria.fr/hal-02161534`

[5] R. J. van Glabbeek and G. D. Plotkin. Configuration structures, event structures, and Petri nets, Theoretical Computer Science 410 (41), 2009.

Causality and Hyperproperties

Norine Coenen, Saarland University
✉: `coenen@react.uni-saarland.de`

Hyperproperties specify relations between multiple execution traces of a system. This is necessary to express requirements like secrecy where for every possible value of a secret variable there has to exist a computation where the value is different while the observations made by an external observer are the same.

Causal relations are often defined using counterfactuals. Intuitively, an action c is considered to be a cause for an effect e if there is one witness where c and e occur and another witness similar to the first where c and e both do not occur. Thus, causality itself is a hyperproperty that compares multiple system traces. In the following, we make this observation more formal.

HyperLTL [1] is a temporal logic that is able to express hyperproperties [2]. It extends linear-time temporal logic (LTL) [16] with trace quantifiers. HyperLTL formulas are built according to the following grammar:

$$\psi ::= \forall \pi.\psi \mid \exists \pi.\psi \mid \varphi$$
$$\varphi ::= a_\pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \Diamond\varphi \mid \Box\varphi,$$

where π is a trace variable and a is an atomic proposition. The semantics of a HyperLTL formula is defined with respect to a *trace assignment* that maps trace variables to actual traces in the system under consideration. Universal and existential trace quantification ($\forall\pi.\psi$, $\exists\pi.\psi$) modify the trace assignment by choosing system traces for the trace variable π. a_π states that the atomic proposition a should hold on the system trace that is identified by π. Additionally, HyperLTL has the usual Boolean connectives and also inherits the temporal operators *next* ($\bigcirc$), *until* ($\mathcal{U}$), *eventually* ($\Diamond$) and *globally* ($\Box$) from LTL.

To determine causal dependencies in a system we need to find system traces that are witnesses for the counterfactual analysis. Leitner-Fischer and Leue [15]

gave a causality definition for this setting as follows: Let t, t' and t'' be system traces and Z and W partition the variables. ψ is considered a *cause* for φ if

$$\begin{aligned}
\text{AC1: } & \exists t.\ t \vDash \psi \wedge t \vDash \varphi \\
\text{AC2(1): } & \exists t'.\ t' \nvDash \psi \wedge t' \nvDash \varphi\ \wedge (val_Z(t) \neq val_Z(t')\ \vee val_W(t) \neq val_W(t')) \\
\text{AC2(2): } & \forall t''.\ t'' \vDash \psi \wedge val_Z(t) = val_Z(t'') \wedge val_{W'}(t) \neq val_{W'}(t'') \rightarrow t'' \vDash \varphi \\
\text{AC3: } & \text{Minimality of } \psi.
\end{aligned}$$

(AC1) ensures that cause and effect both occur on t while (AC2(1)) requires the existence of a different trace t' on which both cause and effect do not occur. (AC2(2)) intuitively states that on all system traces on which the cause occurs also the effect has to occur. (AC3) states that only minimal causes are considered, that means no ψ' smaller than ψ can cause φ.

Clearly, in this definition several system traces are compared to one another. This is exactly what HyperLTL allows us to do so it is natural to formalize this causality definition in HyperLTL. The following HyperLTL formula checks if ψ is a cause for φ in a system:

$$\exists \pi.\ \exists \pi'.\ \forall \pi''.\ (\psi_\pi \wedge \varphi_\pi) \wedge (\neg\psi_{\pi'} \wedge \neg\varphi_{\pi'} \wedge \pi \neq \pi') \wedge (\psi_{\pi''} \wedge \pi =_Z \pi'' \rightarrow \varphi_{\pi''}).$$

Analogue to the causality definition above, the first conjunct ensures that cause and effect both occur on π, the second conjunct ensures that π' is a different trace on which both cause and effect do not occur and the third conjunct stipulates that on all system traces on which the cause occurs also the effect occurs. The minimality condition is not reflected in this definition due to the reasons explained in the summary of the working group discussion "Logics for Causality" below.

Given this formulation, we can now leverage all the work that has been done in the area of hyperproperties. There has been work on different logics for hyperproperties (linear-time and branching-time [1], first-order and second-order style [3, 13]). For HyperLTL in particular, there has been work on the satisfiability problem [5, 6, 8], the model checking problem [4, 11, 12], the synthesis problem [4, 7] and the monitoring problem [9, 10, 14]. The developed tools can now be used to verify causal dependencies using the causality formulation in HyperLTL. We can also use this logical foundation to compare different notions of causality that have been introduced and used in the literature.

We thus can use logics for hyperproperties for the formulation and analysis of causal dependencies in systems and benefit from the existing tool support. This is joint ongoing work with Bernd Finkbeiner.

References

[1] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Proceedings of POST*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

[2] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[3] Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, Canada, June 24-27, 2019*. IEEE Computer Society, 2019.

[4] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2019.

[5] Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *Proceedings of CONCUR*, volume 59 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[6] Bernd Finkbeiner, Christopher Hahn, and Tobias Hans. MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In *Proceedings of ATVA*, volume 11138 of *LNCS*, pages 521–527. Springer, 2018.

[7] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesizing reactive systems from hyperproperties. In *Proceedings of CAV*, volume 10981 of *LNCS*, pages 289–306. Springer, 2018.

[8] Bernd Finkbeiner, Christopher Hahn, and Marvin Stenger. EAHyper: Satisfiability, implication, and equivalence checking of hyperproperties. In *Proceedings of CAV*, volume 10427 of *LNCS*, pages 564–570. Springer, 2017.

[9] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. In *Proceedings of RV*, volume 10548 of *LNCS*, pages 190–207. Springer, 2017.

[10] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In *Proceedings of TACAS*, volume 10806 of *LNCS*, pages 194–200. Springer, 2018.

[11] Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. Model checking quantitative hyperproperties. In *Proceedings of CAV*, volume 10981 of *LNCS*, pages 144–163. Springer, 2018.

[12] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *Proceedings of CAV*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.

[13] Bernd Finkbeiner and Martin Zimmermann. The First-Order Logic of Hyperproperties. In Heribert Vollmer and Brigitte Vallee, editors, *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[14] Christopher Hahn, Marvin Stenger, and Leander Tentrup. Constraint-based monitoring of hyperproperties. In *Proceedings of TACAS*, volume 11428 of *LNCS*, pages 115–131. Springer, 2019.

[15] Florian Leitner-Fischer and Stefan Leue. Causality checking for complex system models. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, volume 7737 of *Lecture Notes in Computer Science*, pages 248–267. Springer, 2013.

[16] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

Efficiently Computing Halpern-Pearl Causality for Binary Models with SAT Solving and ILP

Alexander Pretschner and Amjad Ibrahim, TU Munich

A composite fact X causes an effect Q in the sense of Halpern-Pearl actual causality if (1) X and Q actually happen; (2) changing some of the original values of X while keeping a possibly empty set W of the remaining variables at their original value leads to Q not happening anymore; and (3) X is minimal The complexity of the general problem has been established elsewhere. We show that when restricting ourselves to binary causal modelsi.e., every atom in X is Booleanthe problem can effectively and efficiently be solved using both SAT solving and Integer Linear Programming.

The problem is not trivial because intuitively, we firstly need to enumerate all sets W from condition (2) above and secondly need to check minimality as required by condition (3). We show how to encode both properties as propositional formula instead which immediately gives rise to using a SAT solver to determine if a hypothesized cause satisfies all three properties. In order to compare SAT solving results to ILP, we then proceed by translating the propositional formulas into ILP programs as suggested by textbooks, and then define an objective function that essentially encodes the distance of an unknown cause X to the hypothesized cause X. Using an ILP solver, if we now manage to find an X that is smaller than X, then we know that X satisfies conditions (1) and (2) above but is not a minimal cause.

Using causal models with more than 4000 variables, we show that both approaches can answer causal queries in a matter of seconds, and that the ILP encoding in many cases outperforms the pure SAT encoding. Parts of the results are published in [1].

References

[1] Amjad Ibrahim, Simon Rehwald, Alexander Pretschner: Efficiently Checking Actual Causality with SAT Solving. To appear in Engineering Secure and Dependable Systems, IOS Press, 2020.

2.2 Causality in Virtual Networks and the Cloud

Causality for (software defined) networks

Georgiana Caltais, University of Konstanz

This work introduces a concept of explanations with respect to the violation of safe behaviours within software defined networks (SDNs) expressible in NetKAT. The latter is a network programming language that is based on a well-studied mathematical structure, namely, Kleene Algebra with Tests (KAT). Amongst others, the mathematical foundation of NetKAT gave rise to a sound and complete equational theory.

We also provide an equational framework that computes all relevant explanations witnessing a bad, or an unsafe behaviour, whenever the case. In our setting, safety is characterised by a NetKAT policy which does not enable forwarding packets from ingress to an undesirable egress. The proposed equational framework is a slight modification of the sound and complete axiomatisation of NetKAT, and is parametric on the size of the considered hop-by-hop switch policy. Our approach is orthogonal to related works which rely on model-checking algorithms for computing all counterexamples witnessing the violation of a certain property. A corresponding tool for automatically computing the explanations could be straightforwardly implemented in a programming language like Maude, for instance; we leave this exercise as future work.

The results in this presentation are part of a larger project on (counterfactual) causal reasoning on NetKAT. Lewis formulates the counterfactual argument, which defines when an event is considered a cause for some effect (or hazardous situation) in the following way: a) whenever the event presumed to be a cause occurs, the effect occurs as well, and b) when the presumed cause does not occur, the effect will not occur either. The current result corresponds to item a) in Lewis' definition, as it describes the events that have to happen in order for the hazardous situation to happen as well. The next natural step is to capture the counterfactual test in b). This reduces to tracing back the explanations to the level of the switch policy, and rewrite the latter so that it disables the generation the paths leading to the undesired egress. The generation of a correct" switch policy can be seen as an instance of program repair.

In the future we would be, of course, interested in defining notions of causality (and associated algorithms) with respect to the violation of other relevant properties such as liveness, for instance. We would also like to explain and eventually disable routing loops (i.e., endlessly looping between A and B) from occurring. Or, we would like to identify the cause of packets being not correctly filtered by a certain policy.

Using SAT Solvers to Prevent Causal Failures in the Cloud

Ruzica Piskac, Yale

Today's cloud systems heavily rely on redundancy for reliability. Nevertheless, as cloud systems become ever more structurally complex, independent infrastructure components may unwittingly share deep dependencies. These unexpected common dependencies may result in causal failures that undermine redundancy efforts. The state-of-the-art efforts, e.g., post-failure forensics, not

only lead to prolonged failure recovery time in the face of structurally complex systems, but also fail to avoid expensive service downtime. In this talk, we present a series of work [1, 2] towards preventing causal failures not only in a single cloud data center but also across multiple cloud providers.

The INDaaS system [1] attacked the problem of causal failures in the cloud from the systems research perspective: INDaaS first analyzes the network and automatically collects dependencies between network entities. Next, it constructs a fault graph to model the target system stacks, and then using Monte Carlo simulations, it analyzes the fault graph to identify potential risks. Scalability, efficiency and the lack of formal guarantees were the main problems of the INDaaS system. For the structurally complex cloud systems with tens of thousands of components and multi-layered hardware/software stacks,it is challenging to make fault graph analysis approaches achieve both accuracy and efficiency.

We developed RepAudit [2], an auditing system which builds on the INDaaS system's fault graph, to identify causal failures in the cloud. To ensure the practicality, efficiency, and accuracy of our approach, we further equip RepAudit with a domain-specific auditing language framework, a set of high-performance auditing primitives. Inspired by successful applications of SAT/SMT solvers to formal verification of large-scale programs, we leverage them to construct efficient fault graph analysis algorithms. To empirically evaluate this claim, we run RepAudit on a real-world cloud storage dataset and we observed that RepAudit is 300 times more efficient than other state-of-the-art auditing tools. Furthermore, the returned results are guaranteed to be 100% accurate within the given fault graph.

References

[1] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, Bryan Ford: Heading Off Correlated Failures through Independence-as-a-Service. OSDI 2014: 317-334

[2] Ennan Zhai, Ruzica Piskac, Ronghui Gu, Xun Lao, Xi Wang: An auditing language for preventing correlated failures in the cloud. PACMPL 1(OOPSLA): 97:1-97:28 (2017)

Root Cause Analysis (RCA) for Future Networks: Technology, Vision and Challenges

Armen Aghasaryan, Nokia Bell Labs

Surveillance of complex dynamic communication networks is challenged by high volumes of measurements and alarms to be processed and interpreted. In the future, the number of possible failures will grow exponentially with the increasing number of terminals (IoT), network elements (5G), and traffic volumes. Delayed understanding of a software or hardware failure does not allow to prevent end-to-end performance degradations, negatively impacts the user satisfaction and can result in severe violations of service level agreements.

Today, instead of logically reasoning on the causes of observed data, current AI approaches teach the machine to associate patterns to specific meaning.

This approach fails when the possibilities to learn are too many, and there is not enough data evidence to generalize over the unobserved possibilities. True machine intelligence must be built based on the ability to reason on why it is observing a specific phenomenon and the ability to derive from where it is originating. It can then focus the attention to causally relevant aspects of the ever-growing information flow, instantaneously understand the current situation, and take the most appropriate decisions.

Root Cause Analysis (RCA) is a method of problem solving that aims at diagnosing and correcting the root causes of undesirable events, as opposed to simply addressing their symptoms. When the model of characterizing the relationships between the causes and the symptoms is known, then RCA becomes a model inversion problem where given the symptom observations the most probable underlying causes need to be identified. There is a plethora of techniques and algorithms that can address this problem for various types of models : labelled automata, Petri Nets, Hidden Markov models, Bayesian Networks, etc. However, in most of the practical cases, such a model is unknown and is it extremely hard to be designed manually. Therefore, for practical reasons, before developing diagnosis algorithms one needs to develop methods for automated causal model discovery. Here, we consider two types of environments. For actionable environments, we elaborate interventional learning techniques allowing the discovery of stimulus-reaction behaviors of networked components through injection of perturbations of computing resources. For passively observed environments, we can still infer some of the causal relationships under certain assumptions of causal precedence in observations and availability of some topological knowledge . Open challenges in both contexts will be discussed.

2.3 Explainability and Accountability

Using Model-Based Testing for Doping Detection in Cyber-Physical Systems

Mohammad Reza Mousavi, University of Leicester

Doping is a piece of software or system showing a behviour that is in conflict with the user's intentions and requirements and is caused by an intervention by the manufacturer or malicious agents. Examples of doping include printer or mobile manufacturers locking in users and forcing them to use their original parts and also the infamous diesel emission scandal. Doping detection is a timely subject matter that has been recently addressed by different researchers and several techniques have been proposed for doping detection. I present an overview of the different notions of conformance testing for cyber-phsycial systems and make a case why it makes stuiable tool for doping detection. We show how using conformance testing improves upon a state of the art method for detecting causality. We empirically evaluate our proposed technique on actual data from NOx diesel emission tests and we show that using conformance testing will lead to better use- and a more accurate interpretation of emission data, leading to better results in doping detection.

Parameterized Protocol Analysis and Causal Reasoning

Richard Trefler, U Waterloo

Model checking network protocols is challenging due to state explosion. For instance, when checking safety properties of single protocol instance, a model checker must show that all reachable states of the protocol satisfy the given safety property. However, the number of reachable states, even if finite, may be exponential in the textual program description of the protocol instance. This problem is exacerbated when analyzing whole families of protocol instances, as the general problem becomes undecidable, for instance when the protocol processes communicate over a ring network [8].

Local symmetry reduction [1, 2, 3, 4, 5, 6, 8] is an abstraction technique that may offer substantial, even exponential savings. In essence, the technique is used by first establishing a finite number of equivalence classes of the process types that appear in any network instance. Then analysis is performed on individual process representatives of each of the types. Using over approximations of the reachable states of the representatives, local safety properties of the representatives generalize to global safety properties of the entire network protocol, and further more, to entire families of protocol instances.

When local symmetry reduction fails to prove a protocol correct a failure trace is produced. Causal reasoning has been used to explain error traces found in model checking [7]. In that case, causal reasoning can be used to pinpoint the root causes of a bug. Essentially, the technique locates exactly which values to program variables at which points in the protocol execution have led to the occurrence of a bad state, a bug.

In the context of parametric protocol analysis, however, several added challenges arise. First, a failure to establish a proof of safety for a protocol may occur because the local, abstract protocol instance is simply too abstract to

establish the proof of safety. More detail, more context between 'distant' neighbors may be an essential part of the protocol. On the other hand, even if the protocol is in fact buggy, it may be unclear what the minimum sized process instance is where the bug arises. Addressing these challenges to using causal reasoning to explain counter examples produced in local symmetry reduction are the focus of ongoing research.

References

[1] Rylo Ashmore, Arie Gurfinkel, Richard J. Trefler: Local Reasoning for Parameterized First Order Protocols. NFM 2019: 36-53

[2] Kedar S. Namjoshi, Richard J. Trefler: Symmetry Reduction for the Local Mu-Calculus. TACAS (2) 2018: 379-395

[3] Kedar S. Namjoshi, Richard J. Trefler: Parameterized Compositional Model Checking. TACAS 2016: 589-606

[4] Kedar S. Namjoshi, Richard J. Trefler: Loop Freedom in AODVv2. FORTE 2015: 98-112

[5] Kedar S. Namjoshi, Richard J. Trefler: Analysis of Dynamic Process Networks. TACAS 2015: 164-178

[6] Kedar S. Namjoshi, Richard J. Trefler: Uncovering Symmetries in Irregular Process Networks. VMCAI 2013: 496-514

[7] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, Richard J. Trefler: Explaining counterexamples using causality. Formal Methods in System Design 40(1): 20-40 (2012)

[8] Kedar S. Namjoshi, Richard J. Trefler: Local Symmetry and Compositional Verification. VMCAI 2012: 348-362

[9] Krzysztof R. Apt, Dexter Kozen: Limits for Automatic Verification of Finite-State Concurrent Systems. Inf. Process. Lett. 22(6): 307-309 (1986)

Using Causal Models for Accountability

Alexander Pretschner, Severin Kacianka, and Amjad Ibrahim, TU Munich

The power of modern information technology, and cyber-physical systems is rooted, among other things, in the possibility to easily compose them using software. The ease of composition is due to the virtual nature of software: if a system provides an application programming interface, then other systems can in principle directly interact with that system. It is inherently impractical to specify all legal, or safe, or secure interactions with a system. In turn, this means that the possibility of illegal, unsafe or insecure interactions cannot be excluded at design time. As a consequence, we cannot ensure the adequate functioning of such open systems; hence need to be prepared for failures of the system; and therefore need accountability mechanisms that help us identify the root cause, or responsibility, for such a failure [3].

Traditionally, safety and also security analysis of critical systems start by determining the systems boundary. We will concentrate on the technical boundary here. Any accessible software interface, or API, then is part of that boundary. In practice, these interface descriptions are syntactic in nature. Among other things, this means that they abstract from restrictions on the usage of the API. For instance, a collection of data may need to be initialized before it is used, which is a requirement that is not part of the syntactic interface but may require additional specification in the form of sequencing requirements.

This example generalizes: Many restrictions on the potential usage of an API are left implicit, or come as comments in the best case. The academic community has therefore suggested, for a long time, to provide more detailed interface descriptions. A typical example for such interfaces are contracts that require the specification of preconditions and postconditions for the usage of a specific piece of functionality. In practice, todays software interfaces continue to be mostly syntactic, in spite of decades of research and impressive results on richer notions of interfaces that also incorporate (more detailed descriptions of) the behavior of a component. We prefer not to speculate about the lack of adoption of these ideas in practice. Instead, we would like to remind that any interface description provides a behavior abstractionand in this sense, the syntactic interface, or API, provides such a coarse abstraction as well: data elements of a certain type are mapped to data elements of another type. Arguably, this is the coarsest abstraction of behavior that still is useful in practice. At the other end of the spectrum of levels of abstraction, one may argue that the finest possible abstraction is that of the code itself. However, we do share the perspective that code itself provides an abstraction of behavior in that in most programming languages it does not explicitly talk about time or resource utilization. In this sense, code is not the finest possible abstraction of behavior. Be that as it may, the above shows that there is a huge spectrum of possible levels of abstraction for describing the behavior, or an interface, of a system. It is important to remember that none of these levels as such is better than another level: this depends on the purpose of the abstraction, as is the case for any kind of model.

All this does not necessarily constitute novel insights. There is a consequence, however, that we feel has been underestimated in the past and that is the basis for the work presented here: Regardless of the level of abstraction of an interface that we choose, there must, by definition, always be parts of the behavior that are left unspecified. And this, in turn, means that the boundary of a software-intensive system usually is not, should not, and most importantly: cannot be specified without leaving open several degrees of freedom in terms of how to legally use the system. It hence cannot be excluded that a system S1 is used by another system S2 in a way that the developer of S1 has never envisaged and which violates implicit assumptions that were made when developing S1, possibly leading to a runtime failure of S1 and, by consequence, also of S2.2. One consequence is that software-rooted failures for composed, or open, systems cannot be excluded by design. Because these systems are becoming ever more complex, we consider it hence mandatory to provide mechanisms for understanding the root causes of a failure, both for eliminating the underlying (technical) problem and for assigning blame.

We call a system accountable that can help answer questions regarding the root cause of some (usually undesired) event (other notions of accountability are

studied and formalized elsewhere). Accountability of a system requires at least two properties: The system must provide evidence, usually in the form of logs, that helps understand the reasons of the undesired event. Moreover, there must be some mechanism that can reason about causality. Different forms of causality are useful in our context. Just to mention two of them, Granger causality identifies causality with a reduction of error when it comes to predicting one time series on the grounds of another; and model-based diagnosis computes candidate sets of potentially faulty components that can explain the failure of a system. In this talk, we focus on one technical aspect of inferring one specific kind of causality, namely Halpern-Pearl-style actual causality for counter-factual arguments: Given a failure of a system (an effect) and a potential cause, we efficiently compute if, by counterfactual argument, the potential cause is an actual cause.

In order to perform this kind of post-hoc analysis, we need to provide causal models. We argue that existing artifacts can be used as a baseline for these models. (1) Using the example of drones, we show how fault trees automatically inferred from an interpretation of the Four-Variable-Model by Parnas and Madey can be used for causal inference [4], and how data analysis techniques on drone logs can be used to infer further causal models. (2) We show how attack trees can be derived from network structures [1]. (3) We show how Timed Failure Propagation Graphs can server as a basis for causal models. We finally hint at prior work that shows how to combine such models [2].

References

[1] Ibrahim, Amjad; Bozhinoski, Stevica; Pretschner, Alexander: Attack graph generation for microservice architecture. Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing - SAC '19, pages 1235-1242, ACM, 2019.

[2] Ibrahim, Amjad; Kacianka, Severin; Pretschner, Alexander; Hartsell, Charles; Karsai, Gabor: Practical Causal Models for Cyber-Physical Systems. In Nasa Formal Methods 2019, pages 211-227, Springer Lecture Notes in Computer Science, 2019.

[3] Amjad Ibrahim, Simon Rehwald, Alexander Pretschner: Efficiently Checking Actual Causality with SAT Solving. To appear in Engineering Secure and Dependable Systems, IOS Press, 2020.

[4] Zibaei, Ehsan; Banescu, Sebastian; Pretschner, Alexander: Diagnosis of Safety Incidents for Cyber-Physical Systems: A UAV Example. 2018 3rd International Conference on System Reliability and Safety (ICSRS), IEEE, 2018.

2.4 Decision Making and Risk Management

Causality and decision-making

Samantha Kleinberg, Stevens Institute of Technology

Risk Structures: Concepts, Purpose, and Causality

Mario Gleirscher, University of York, United Kingdom

Autonomous machines have *safety mechanisms* to predict, detect, and react to dangerous events autonomously until a *situation-specific minimum risk state* is reached. Corresponding *safety policies* have to be *verified against a specification* covering the variety of situations. Engineers could benefit from design techniques integrating hazard analysis and risk assessment with the specification and verified synthesis of policies for autonomous safety mechanisms.

My research aims at a design technique for autonomous safety mechanisms based on models of the *controlled process* and of *risk*: The *process model* abstracts from actions of the process and the *risk model* describes which of these actions can increase or decrease risk in certain situations.

Risk Structures. Let V be a set of variables defining the *concrete process state space*. While such states are the concrete outcomes of actions performed by the process, we describe the process by a *scenario model*, a tuple $(S, \rightarrow_s)$ with the set S of *(operational) situations*, abstracting from physical dynamics or other activity observable through V, and the transition relation $\rightarrow_s \subseteq S \times S$, abstracting from the process to sequences of situations.

For risk modelling, we use *risk structures* [1]. Let F be a set of *risk factors*, a generalisation of safety-related faults. Each factor $f \in F$ defines a partition of the state space over V. Each member of this partition can be described by a predicate over V. Let $R(F)$ be the set of *risk states* resulting from the Cartesian product of the partitions described by F [1]. Then, each risk state represents the predicates a concrete process state satisfies.

A risk structure $\mathfrak{R}$ is a tuple $(\Sigma, M(F), \leq_{\mathsf{m}}, \rightarrow_r)$ with the *risk space* $\Sigma \subseteq S \times R(F)$, the *mitigation capability* $M(F)$ and an *endangerment* action $\tau \notin M(F)$, a linear *mitigation order* $\leq_{\mathsf{m}} \subseteq \Sigma \times \Sigma$, and the *labelled transition relation* $\rightarrow_r \subseteq \Sigma \times M(F) \cup \{\tau\} \times \Sigma$. Furthermore, let $\mu \subset \Sigma$ be the set of *mishaps*.

Given $m \neq \tau$, we call $(\sigma, m, \sigma') \in \rightarrow_r$ a *successful mitigation* if $\sigma \leq_{\mathsf{m}} \sigma'$, *ineffective* otherwise. Given $a \in M(F) \cup \{\tau\}$, we call a transition $(\sigma, a, \sigma') \in \rightarrow_r$ an *endangerment* iff $\sigma' \leq_{\mathsf{m}} \sigma$.

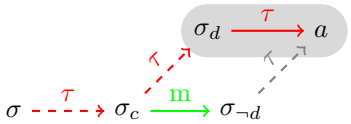
Risk structures aim at helping engineers in the modelling and examination of risk scenarios. In support of the planning layer of an autonomous machine (collective), risk structures aim at the design of *mitigations*, that is, safety monitors and controllers for run-time handling of risk in such scenarios.

The construction of $\mathfrak{R}$ involves the abstraction of actions with determined, uncertain, or probabilistic outcomes into actions with risk-increasing and risk-neutral outcomes with respect to F. My contribution to the seminar focused on the abstraction used to construct risk structures, on the purpose of risk structures, and on the relationship to causal reasoning.

Causal Reasoning in Risk Structures. A *cause* is any member of any *minimal set of actual conditions* that are jointly *sufficient*, given the laws, for the existence of the *effect* [2]. Lewis' *counterfactual conditional* $A \boxdot\!\!\rightarrow C$ is non-vacuously true iff C holds at *all* the closest A-worlds. In other words, a counterfactual is non-vacuously true iff it takes less of a departure from actuality to make C true along with A than it does to make A true without C [2].

In $\mathfrak{R}$, we consider *dependencies* between factors $a, b \in F$. For example, the activation of a (probably) *causes* the activation of b. After activating a, the process performs actions that, without possible intervention (and at a certain likelihood), activate b. Causal reasoning helps to identify risk factors, their roles as *causes* and *consequences* of other risk factors, and the capability $M(F)$.

Causal Abstraction. The choice of $(F, M(F))$ induces the distinction of *indirect* and *direct* causes or consequences. Let $c, d \in F$; $\sigma, \sigma_c, \sigma_{\neg d}$, $\sigma_d, a \in \Sigma$; $a \in \mu$; $m \in M(F)$. For example, assume the following fragment in $\rightarrow_r$:



Risk states separating direct causes of an active factor or separating a factor from its direct consequences should belong to an equivalence class (i.e., highlighted in grey and including σ_d, a) and, hence, shape the abstraction of $R(F)$. From a probabilistic point of view, states immediately reachable by endangerments with high likelihood (a) can be merged with states previously reached in the process (σ_d). However, states where the system is capable of making mitigation decisions (σ_c) are highly relevant for $R(F)$. We call $\mathfrak{R}$ *causally indirect* iff $\rightarrow_r$ has only *indirect* causal dependencies.

Mitigation and Counterfactuals. Assume that σ_d contains at least one cause d of mishap a; σ_c contains at least one *indirect* cause c likely leading to σ_d if not intercepted; m removes c from σ_c by transition to $\sigma_{\neg d}$; and $\sigma_{\neg d}$ contains strictly less (or less likely) causes of a than σ_d. Then, we call m (of d, a) *successful*, otherwise ineffective. Overall, we say that $\mathfrak{R}$ is *feasible* if $\forall \sigma \in \Sigma \setminus \mu \; \exists \sigma' \in \Sigma \setminus \mu, m \in M(F) \colon (\sigma, m, \sigma') \in \rightarrow_r$ successful.

According to the notion of closest A-world [2], $\sigma_d \boxdot\!\!\rightarrow a$ requires that $\sigma_{\neg d}$ represents all worlds closest to σ_d but with the cause d of a (indirectly) removed by m (through removing c).

Safety Policies. A *safety policy* for $\mathfrak{R}$ is a map $P \colon \Sigma \rightarrow 2^{\rightarrow_r}$. We call P a *successful policy* if it returns only successful mitigations. Given that a mitigation $m \in M(F)$ can be described as a guarded statement and with $\sigma \in \Sigma$, we require for all transitions $(\sigma, m, \sigma') \in P(\sigma)$ that m terminates and, for the weakest predicate *post* where $\sigma \Rightarrow wp(m, post)$, that $post \Rightarrow \sigma'$.

My research concerns the building blocks of a compositional method that guides the step-wise construction of risk structures and their use for the synthesis of successful safety policies deployed in an autonomous machine (collective).

References

[1] Mario Gleirscher. Risk structures: Towards engineering risk-aware autonomous systems. Unpublished working paper, Department of Computer Science, University of York, 2019.

[2] David Lewis. Causation. *The Journal of Philosophy*, 70(17):556, 10 1973.

2.5 Fault Localisation, Cause Mining, and Provenance

Finding Minimum Type Error Sources

Thomas Wies, New York University

Automatic type inference is a popular feature of functional programming languages. However, compilers for these languages are notorious for generating confusing type error messages. When the compiler detects a type error, it typically reports the program location where the type checking failed as the source of the error. Since other error sources are not even considered, the actual root cause is often missed. In this talk, I present a general framework for automatic localization of type errors. Our algorithm finds all minimum error sources, where the exact definition of minimum is given in terms of a compiler-specific ranking criterion. Our approach works by reducing the search for minimum error sources to an optimization problem that we formulate in term of weighted maximum satisfiability modulo theories (MaxSMT). Experiments with a prototype implementation indicate that the technique has the potential to significantly improve the quality of type error reports produced by state-of-the-art compilers.

Cause Mining with STL

Ebru Aydin Gol, Middle East Technical University

Designing cyber-physical systems that achieve complex tasks is a difficult and error prone process. The resulting models are, in general, complex and composed of various sub-modules. Once the model is developed, its traces are checked against the specifications for verification. While it is relatively easy to simulate the system and mark the erroneous behaviors, it is extremely challenging to locate the root cause in the model that lead to the unexpected behaviors. To expedite this process, we develop methods and tools to find temporal properties that lead to the unexpected behaviors from labeled system traces in an automated way. We express these properties as past time Signal Temporal Logic (ptSTL) formulas. A ptSTL formula consists of Boolean and past temporal operators. Essentially, as opposed to the logics with future operators, a ptSTL formula is evaluated with respect to the trace segment preceding the current point of reference. Thus, it offers an intuitive formalization to express the causes of the marked events.

We consider the following problem: given a set of labeled traces, i.e., time-series data, find a ptSTL formula such that the evaluation of the formula along the traces mimics the given labels. We develop unsupervised formula synthesis methods for this problem. In particular, given the labeled dataset, we generate a formula in a fully automated way. The first method we developed performs parameter optimization for each parametric formula with a given number of operators. The operator count, thus the formula expressivity, is increased iteratively until a formula with sufficient fitness value is found. While this method can potentially generate the optimal formula, it is computationally expensive due to its greedy nature. The second method is performs parameter optimization for a set of parametric formulas and iteratively combines optimized formulas.

Compared to the greedy approach, the second method generates sub-optimal formulas with a significant improvement in the computation time.

We illustrate the proposed approaches on two examples. In the first example, we consider an aircraft longitudinal flight control model and identify conditions which cause the aircraft's longitudinal motion to disturb. In the second example, we consider the traces generated by a traffic simulator and generate formulas describing events resulting in traffic congestion. In both cases, The properties generated by our approach give an insight on the underlying cause.

Using Causal Models to Infer Data Provenance for Attack Investigation

Ashish Gehani, SRI

Knowledge about the provenance of data has numerous applications. In diverse settings, such as databases, operating systems, and workflow managers, it can help with a multitude of tasks, that include helping identify a computation's external dependencies, assessing the impact of a security breach, and tracking cross-layer performance metrics. This has led to a number of provenance-focused activities over the past two decades. Early workshops on the topic were held in 2002, 2003, and 2005 in Chicago, Edinburgh, and Boston, respectively. The biennial *International Provenance and Annotation Workshop (IPAW)* [17] started in 2006. The U.K.'s eScience Institute sponsored a theme on the *Principles of Provenance* [4], starting in 2008. It hosted a series of symposia locally, and led to the creation of USENIX's *Theory and Practice of Provenance (TaPP)* [31] that has been held annually since 2009. A Dagstuhl Seminar on the *Principles of Provenance* [3] was organized in 2012. Starting in 2014, IPAW and TaPP have been co-located as part of an umbrella ProvenanceWeek event [27], along with other provenance-related satellite workshops, every other year. A series of Provenance Challenge events [26] were held from 2006 to 2010, to help study provenance capture, the interoperability of systems, and validate the Open Provenance Model [24, 23]. The W3C standard PROV [28] was created in 2013. To facilitate comparisons, Provenance Benchmark events [25] collected traces in 2013 and 2014.

At SRI, we designed systems for certifying video provenance [5], trading the storage overhead and reliability of provenance verification [6], performing forensic analysis using provenance [7], optimizing cluster provenance replication [8], verifying multi-domain provenance [9], using provenance to track Grid infections [10, 32], integrating scientific provenance [11], auditing provenance in distributed environments [12], sketching provenance [21], collecting application-level provenance using compiler-based instrumentation [33], minimizing workflow re-execution using provenance [20], using provenance for mobile diagnostics [16], declaratively processing provenance [22], framing provenance integration as an optimization problem [14], vetting applications for sensitive data flows [35], studying tradeoffs in byte-, function-, and system-call level provenance tracking [30], scaling to "big provenance" streams [15], compressing provenance [1], detecting Advanced Persistent Threats (APTs) using provenance [2], and released cross-platform provenance data sets [13]. As with most efforts in the community, the causal models used to infer provenance were based on expert

knowledge of target domains.

From 2015 to 2019, SRI's SPADE [29] was deployed in 5 engagements of the DARPA Transparent Computing program [34]. It inferred data provenance based on Linux Audit events. The underlying causal models used to detect dependencies were constructed by studying the semantics of over 300 system calls and crafting interpretations for 65 of them by hand. An alternate automated approach was studied, where model-based causality inference (MCI) [19] is used to analyze operating system audit logs. It can be employed without application instrumentation or kernel modification on a target system. Instead, it leverages a lightweight dual execution (LDX) [18] engine to efficiently build system-call-level causal models using a combination of *counterfactual reasoning* and compiler instrumentation. LDX is used on a training system to acquire precise causal models of primitive behavior in applications that are expected to be present in the target system. The causal models consist of *explicit dependencies* that can be inferred directly from the audit records of system calls, as well as *implicit dependencies* that arise due to in-memory data flow from one call to another. Models can be described in regular, context-free, or context-sensitive languages, with increasing parsing complexity. After the models are acquired on a training system, they can be deployed to parse logs from a target system. The provenance inferred using these causal models has higher precision than that which could be derived from the target system's audit logs alone. Of particular note, the models can be combined to support parsing logs of composite behavior in the target system. More precise provenance facilitates improved forensic analysis, intrusion detection, and access control decisions.

References

[1] Raza Ahmad, Melanie Bru, and Ashish Gehani, **Streaming Provenance Compression**, *Provenance and Annotation of Data and Processes, Lecture Notes in Computer Science*, Vol. 11017, Springer, 2018.

[2] Mathieu Barre, Ashish Gehani, and Vinod Yegneswaran, **Mining Data Provenance to Detect Advanced Persistent Threats**, *11th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2019.

[3] Dagstuhl *Principles of Provenance* Seminar, `https://www.dagstuhl.de/12091`

[4] U.K. eScience Institute *Principles of Provenance* Theme, `https://web.archive.org/web/20150908020424/http://wiki.esi.ac.uk/Principles_of_Provenance`

[5] Ashish Gehani and Ulf Lindqvist, **VEIL: A System for Certifying Video Provenance**, *9th IEEE International Symposium on Multimedia (ISM)*, 2007.

[6] Ashish Gehani and Ulf Lindqvist, **Bonsai: Balanced Lineage Authentication**, *23rd Annual Computer Security Applications Conference (ACSAC), IEEE Computer Society*, 2007.

[7] Ashish Gehani, Florent Kirchner, and Natarajan Shankar, **System Support for Forensic Inference**, *5th IFIP International Conference on Digital Forensics*, 2009.

[8] Ashish Gehani, Minyoung Kim, and Jian Zhang, **Steps Toward Managing Lineage Metadata in Grid Clusters**, *1st Workshop on the Theory and Practice of Provenance (TaPP)*, 2009.

[9] Ashish Gehani and Minyoung Kim, **Mendel: Efficiently Verifying the Lineage of Data Modified in Multiple Trust Domains**, *19th ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.

[10] Ashish Gehani, Basim Baig, Salman Mahmood, Dawood Tariq, and Fareed Zaffar, **Fine-Grained Tracking of Grid Infections**, *11th ACM/IEEE International Conference on Grid Computing (GRID)*, 2010.

[11] Ashish Gehani, Dawood Tariq, Basim Baig, and Tanu Malik, **Policy-Based Integration of Provenance Metadata**, *12th IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, 2011.

[12] Ashish Gehani and Dawood Tariq, **SPADE: Support for Provenance Auditing in Distributed Environments**, *13th ACM/IFIP/USENIX International Conference on Middleware*, 2012.

[13] Ashish Gehani and Dawood Tariq, **Cross-Platform Provenance**, *1st ACM Provenance Benchmark Challenge*, 2013.

[14] Ashish Gehani and Dawood Tariq, **Provenance-Only Integration**, *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2014.

[15] Ashish Gehani, Hasanat Kazmi, and Hassaan Irshad, **Scaling SPADE to "Big Provenance"**, *8th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2016.

[16] Nathaniel Husted, Sharjeel Qureshi, Dawood Tariq, and Ashish Gehani, **Android Provenance: Diagnosing Device Disorders**, *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2013.

[17] International Provenance and Annotation Workshop, `https://dblp.org/db/conf/ipaw/`

[18] Yonghwi Kwon, Dohyeong Kim, William Sumner, Kyungtae Kim, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu, **LDX: Causality Inference by Lightweight Dual Execution**, *21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[19] Yonghwi Kwon, Fei Wang, Weihang Wang, Kyu Hyung Lee, Wen-Chuan Lee, Shiqing Ma, Xiangyu Zhang, Dongyan Xu, Somesh Jha, Gabriela Ciocarlie, Ashish Gehani, and Vinod Yegneswaran, **MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation**,

25th Annual Network and Distributed System Security Symposium (NDSS), 2018.

[20] Hasnain Lakhani, Rashid Tahir, Azeem Aqil, Fareed Zaffar, Dawood Tariq, and Ashish Gehani, **Optimized Rollback and Re-computation**, *46th IEEE Hawaii International Conference on Systems Science (HICSS)*, IEEE Computer Society, 2013.

[21] Tanu Malik, Ashish Gehani, Dawood Tariq, and Fareed Zaffar, **Sketching Distributed Data Provenance**, *Data Provenance and Data Management for eScience, Studies in Computational Intelligence*, Vol. 426, Springer, 2013.

[22] Scott Moore, Ashish Gehani, and Natarajan Shankar, **Declaratively Processing Provenance Metadata**, *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2013.

[23] Luc Moreau, Ben Clifford, Juliana Freire, Yolanda Gil, Paul Groth, Joe Futrelle, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche, The Open Provenance Model core specification (v1.1), *Future Generation Computer Systems*, 2010.

[24] Open Provenance Model, `https://openprovenance.org/opm/`

[25] Provenance Benchmarks, `https://sites.google.com/site/provbench/`

[26] Provenance Challenges, `https://openprovenance.org/provenance-challenge/WebHome.html`

[27] Provenance Week, `http://provenanceweek.org`

[28] W3C PROV, `http://www.w3.org/TR/prov-overview/`

[29] Support for Provenance Auditing in Distributed Environments, http://spade.csl.sri.com

[30] Manolis Stamatogiannakis, Hasanat Kazmi, Hashim Sharif, Remco Vermeulen, Ashish Gehani, Herbert Bos, and Paul Groth, **Tradeoffs in Automatic Provenance Capture**, *Provenance and Annotation of Data and Processes, Lecture Notes in Computer Science*, Vol. 9672, Springer, 2016.

[31] USENIX Theory and Practice of Provenance, `https://dblp.org/db/conf/tapp/`

[32] Dawood Tariq, Basim Baig, Ashish Gehani, Salman Mahmood, Rashid Tahir, Azeem Aqil, and Fareed Zaffar, Identifying the provenance of correlated anomalies, *26th ACM Symposium on Applied Computing*, 2011.

[33] Dawood Tariq, Maisem Ali, and Ashish Gehani, **Towards Automated Collection of Application-Level Data Provenance**, *4th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2012.

[34] DARPA Transparent Computing, `https://www.darpa.mil/program/transparent-computing`

[35] Chao Yang, Guangliang Yang, Ashish Gehani, Vinod Yegneswaran, Dawood Tariq, and Guofei Gu, **Using Provenance Patterns to Vet Sensitive Behaviors in Android Apps**, *11th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2015.

Automating Time Series Safety Analysis for Automotive Control Systems using Weighted Partial Max-SMT

Shoji Yuen, Nagoya University

We propose a method to detect unexpected time-series signal disturbances for a given unsafe property to assist safety analysis. A signal disturbance is incorporated by an in-equivalences between internal variables of before and after transitions. A signal disturbance exists if a value of variable x is altered by another value. A *cushion variable* x' for x is introduced to hold the altered value. With no signal disturbance, $x = x'$ holds, but $x \neq x'$ holds if a signal disturbance of x exists. A trace of the system is described, and we add these extra equations to characterize the trace. We develop a method to efficiently detect the signal disturbance by using a weighted partial maximum satisfiability modulo theories (Max-SMT) technique as a set of variables altered by faults resulting in an undesirable condition. By assigning the weights properly to the equations, we control the derivation of signal disturbance patterns with the required property. We present an experimental application of our method to a simplified cruise control system as a practical case study following the system theoretic process analysis (STPA) for the automatic detection of time-series signal disturbances. (Joint work with Shuichi Sato, Yutaka Inamori[TOYOTA Central R& D], Shogo Hattori, and Hiroyuki Seki[Nagoya University]).

3 Summaries of Working Groups

Logics for Causality

Participants: Norine Coenen, Gregor Gössler, Stefan Leue, Ruzica Piskac, Richard Trefler, and Thomas Wies.

Our main goal was to identify characteristics that a logic should have such that it is suitable for causal reasoning in systems. In the following, we present the most relevant aspects:

- Temporal Order: While no accepted definition of causality exists, all definitions respect the temporal order between cause and effect. A cause has to always precede its effect. A logic for causality, thus, needs to have the capability to capture this temporal aspect.
- (Close) Alternate Worlds / Counterfactuals: Counterfactual reasoning compares the actual world in which cause and effect occur with a similar alternate world in which cause and effect both do not occur. A logic for causality, thus, needs to compare different worlds in order to express the counterfactual reasoning.
- Minimality: Usually, we are interested in finding minimal causes for an effect. However, it is not necessary to express that in the logic directly. It is sufficient to express the causal dependency between a cause c and an effect e within the logic and, additionally, test that no causal dependency between a cause c' smaller than c and e exists. Minimality can then be expressed on the semantical level as logical entailment.
- Mechanization of the Logic: In order to benefit from automation in the analysis of causal dependencies, it is important that the logic, or at least a reasonable fragment thereof, is decidable.
- (No) Statistical Operators: Finding suitable candidate causes and effects can be done using statistical techniques such as regression-based inference in a preprocessing step. Given these candidates, we use the corresponding logical formula to check whether there indeed is a causal dependency between the suggested causes and effects.

HyperLTL [1] is a natural choice for a logic for causality. As a temporal logic for hyperproperties, HyperLTL already combines the temporal aspect with the possibility to compare different system traces to identify the counterfactuals. Moreover, at least fragments of HyperLTL are decidable [3], including the fragment containing the causality formulation described in the talk abstract "Causality and Hyperproperties" above, and useful tools for checking satisfiability [3, 4, 6], model checking [2, 9, 10], synthesis [2, 5] and runtime monitoring [7, 8, 11] exist.

References

[1] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Proceedings of POST*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

[2] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2019.

[3] Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *Proceedings of CONCUR*, volume 59 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[4] Bernd Finkbeiner, Christopher Hahn, and Tobias Hans. MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In *Proceedings of ATVA*, volume 11138 of *LNCS*, pages 521–527. Springer, 2018.

[5] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesizing reactive systems from hyperproperties. In *Proceedings of CAV*, volume 10981 of *LNCS*, pages 289–306. Springer, 2018.

[6] Bernd Finkbeiner, Christopher Hahn, and Marvin Stenger. EAHyper: Satisfiability, implication, and equivalence checking of hyperproperties. In *Proceedings of CAV*, volume 10427 of *LNCS*, pages 564–570. Springer, 2017.

[7] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. In *Proceedings of RV*, volume 10548 of *LNCS*, pages 190–207. Springer, 2017.

[8] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In *Proceedings of TACAS*, volume 10806 of *LNCS*, pages 194–200. Springer, 2018.

[9] Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. Model checking quantitative hyperproperties. In *Proceedings of CAV*, volume 10981 of *LNCS*, pages 144–163. Springer, 2018.

[10] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *Proceedings of CAV*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.

[11] Christopher Hahn, Marvin Stenger, and Leander Tentrup. Constraint-based monitoring of hyperproperties. In *Proceedings of TACAS*, volume 11428 of *LNCS*, pages 115–131. Springer, 2019.

4 List of Participants

- Armen Aghasaryan, Nokia Bell Labs
- Vitaliy Batusov, York University, CA
- Georgiana Caltais, University of Konstanz
- Norine Coenen, Saarland University
- Ashish Gehani, SRI
- Mario Gleirscher, University of York, UK
- Gregor Gössler, INRIA
- Ebru Aydin Gol, Middle East Technical University
- Samantha Kleinberg, Stevens Institute of Technology
- Stefan Leue, University of Konstanz
- Mohammad Reza Mousavi, University of Leicester
- Ruzica Piskac, Yale
- Alexander Pretschner, TU Munich
- Richard Trefler, U Waterloo
- Thomas Wies, New York University
- Shoji Yuen, Nagoya University