Causality Analysis for Concurrent Reactive Systems (Extended Abstract)

Rayna Dimitrova
University of Leicester
Leicester, UK
rd307@leicester.ac.uk

Rupak Majumdar
MPI-SWS
Keiserslautern and Saarbrücken, Germany
rupak@mpi-sws.org

Vinayak S. Prabhu
Colorado State University
Fort Collins, Colorado, United States
vinayak@mpi-sws.org

We present a comprehensive language theoretic causality analysis framework for explaining safety property violations in the setting of concurrent reactive systems. Our framework allows us to uniformly express a number of causality notions studied in the areas of artificial intelligence and formal methods, as well as define new ones that are of potential interest in these areas. Furthermore, our formalization provides means for reasoning about the relationships between individual notions which have mostly been considered independently in prior work; and allows us to judge the appropriateness of the different definitions for various applications in system design. In particular, we consider causality analysis notions for debugging, error resilience, and liability resolution in concurrent reactive systems. Finally, we present automata-based algorithms for computing various causal sets based on our language-theoretic encoding, and derive the algorithmic complexities.

Causality analysis, which investigates questions of the form "Does event e_1 cause event e_2?" plays an important role in many areas of science, medicine and law. In formal methods, causality analysis has been used to determine the *coverage of specifications* [4] (that is, which parts of the system under scrutiny are relevant for the satisfaction of a specification), to *explain counterexamples* [1] (identify points in a counterexample trace that are relevant for the failure of a temporal specification), to *construct fault trees* [9], and to *automatically refine system abstractions* [3]. In artificial intelligence, causality-based explanation finding has applications in natural language processing, automated medical diagnosis, vision processing, and planning. Resolving liabilities in a legal setting often relies on establishing the causal relations between potential causes and the occurred damage [2].

Causality definitions based on *counterfactuals*, which are alternative scenarios where the suspected cause e_1 of e_2 did not happen, date back to [8] and have been extensively studied in philosophy [10]. In computer science, the most prominent and widely used definition of causality is that of [7], in which the authors write "... while it is hard to argue that our definition (or any other definition, for that matter) is the right definition, we show that it deals with the difficulties that have plagued other approaches in the past ...". Halpern and Pearl's approach is based on *structural equations*, which describe causal dependencies between Boolean variables. We extend the Boolean study of causality to the *temporal* setting; specifically, we formalize notions of causality in *concurrent reactive systems* whose behaviors evolve over time. A concurrent reactive system is a composition of interacting components; the system behavior is determined by the *repeated* interaction between the components over time. Moreover, we consider the setting where component implementations are not available for analysis and the designer has only access to specifications of their expected behavior. Thus, when analyzing an *error trace* (an execution of the system that violates a desired system-level property), the only available information about the system consists of the components' specifications and the observed trace.

In our framework, a concurrent reactive system $C_1 \parallel C_2 \parallel \ldots \parallel C_n$ is a composition of components $C_1, \ldots, C_n$. Each component C_i is specified as a tuple $(X_i, \mathsf{inp}(X_i), \mathsf{out}(X_i), \Sigma_i, \varphi_i)$, where

B. Finkbeiner and S. Kleinberg: Third International Workshop on Formal Reasoning about Causation, Responsibility, and Explanations in Science and Technology (CREST 2018)
EPTCS 286, 2019, pp. 31–33, doi:10.4204/EPTCS.286.3

- $X_i = \mathsf{inp}(X_i) \uplus \mathsf{out}(X_i)$ is the set of variables of the component, consisting of the input variables $\mathsf{inp}(X_i)$ and the output variables $\mathsf{out}(X_i)$ (the sets of input and output variables being disjoint);
- Σ_i is the alphabet, consisting of all possible valuations of the variables X_i;
- φ_i is a non-empty prefix-closed language over Σ_i, specifying the set of correct behaviours of C_i.

The composite system $C_1 \parallel C_2 \parallel \ldots \parallel C_n$ has an associated prefix-closed specification θ such that θ contains $\varphi_1 \parallel \ldots \parallel \varphi_n$. Thus, the global requirement is more relaxed than the promised behaviors of the individual components. In other words, the system $C_1 \parallel C_2 \parallel \ldots \parallel C_n$ promises to implement or refine the global requirement θ.

Consider a trace $\mathtt{tr}$ of the system $C_1 \parallel C_2 \parallel \ldots \parallel C_n$ in which the system requirement θ is violated. Let $C'_1, \ldots, C'_k$ be the components which violate their local specifications $\varphi'_1, \ldots, \varphi'_k$. The *causality analysis problem* is to determine which component set $\{D_1, \ldots, D_m\} \subseteq \{C'_1, \ldots, C'_k\}$ is liable for the global system requirement violation in $\mathtt{tr}$. Our analysis reasons about two classes of scenarios to determine if $\mathscr{D} = \{D_1, \ldots, D_m\}$ is a cause:

- *Fault Mitigation Capability* analysis asks whether the *correct behavior* of the components in the set $\mathscr{D}$ is enough to mitigate the faults of all components (*including those of components not in* $\mathscr{D}$), by ensuring that the required system property holds.
- *Fault Manifestation* analysis asks whether the observed *faulty behavior* of the components in the set $\mathscr{D}$ is enough to manifest a global fault (*i.e.*, a system behavior violating the global property), even if the components not in $\mathscr{D}$ were to behave correctly.

These two classifications parallel the classifications of [6, 5] of causes into *necessary causes* and *sufficient causes*. However, our analysis is not limited to specific definitions of counterfactual sets. In contrast, we provide a reasoning framework based on generic counterfactual sets, and introduce several natural instantiations. We demonstrate that the generality and modularity of our definition of causality allow us to seamlessly extend causality analysis to the case of *heterogeneous fault models*, where different components are examined under different fault scenarios. Finally, we present an automata-based method for determining various causal sets in the setting of heterogeneous component-fault models, and derive its algorithmic complexity.

References

[1] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni & Richard J. Trefler (2012): *Explaining counterexamples using causality*. *Formal Methods in System Design* 40(1), pp. 20–40, doi:10.1007/s10703-011-0132-2.

[2] Francesco D. Busnelli, Giovanni Comandé, Herman Cousy, Dan B. Dobbs, Bill W. Dufwa, Michael G. Faure, Israel Gilead, Michael D. Green, Konstantinos D. Kerameus, Bernhard A. Koch, Helmut Koziol, Ulrich Magnus, Miquel Martín-Casals, Olivier Moréteau, Johann Neethling, W. V. Horton Rogers, Jorge Ferreira Sinde Monteiro, Jaap Spier, Lubos Tichy & Pierre Widmer (2005): *Causation*, pp. 43–63. Springer Vienna, Vienna, doi:10.1007/3-211-27751-X_4.

[3] Hana Chockler, Orna Grumberg & Avi Yadgar (2008): *Efficient Automatic STE Refinement Using Responsibility*. In C. R. Ramakrishnan & Jakob Rehof, editors: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, *Lecture Notes in Computer Science* 4963, Springer, pp. 233–248, doi:10.1007/978-3-540-78800-3_17.

[4] Hana Chockler, Joseph Y. Halpern & Orna Kupferman (2008): *What causes a system to satisfy a specification? ACM Trans. Comput. Log.* 9(3), pp. 20:1–20:26, doi:10.1145/1352582.1352588.

[5] Gregor Gößler & Daniel Le Métayer (2013): *A General Trace-Based Framework of Logical Causality*. In José Luiz Fiadeiro, Zhiming Liu & Jinyun Xue, editors: *Formal Aspects of Component Software - 10th International Symposium, FACS 2013, Nanchang, China, October 27-29, 2013, Revised Selected Papers*, *Lecture Notes in Computer Science* 8348, Springer, pp. 157–173, doi:10.1007/978-3-319-07602-7_11.

[6] Gregor Gößler, Daniel Le Métayer & Jean-Baptiste Raclet (2010): *Causality Analysis in Contract Violation*. In Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky & Nikolai Tillmann, editors: *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, *Lecture Notes in Computer Science* 6418, Springer, pp. 270–284, doi:10.1007/978-3-642-16612-9_21.

[7] J.Y. Halpern & J. Pearl (2005): *Causes and explanations: A structural-model approach. Part I: Causes*. *The British journal for the philosophy of science* 56(4), pp. 843–887, doi:10.1093/bjps/axi147.

[8] D. Hume (1748): *An Enquiry concerning Human Understanding*. doi:10.1093/oseo/instance.00032980.

[9] Florian Leitner-Fischer & Stefan Leue (2013): *Probabilistic fault tree synthesis using causality computation*. *IJCCBS* 4(2), pp. 119–143, doi:10.1504/IJCCBS.2013.056492.

[10] D. Lewis (2004): *Void and Object*. In John Collins, Ned Hall & L. A. Paul, editors: *Causation and Counterfactuals*, MIT Press, pp. 277–290.