The Journal of Systems and Software 209 (2024) 111915

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Feature causality[☆],[☆☆]

Clemens Dubslaff[a,b,*], Kallistos Weis[d], Christel Baier[b,c], Sven Apel[d]

[a] *Formal System Analysis Group, Eindhoven University of Technology, Eindhoven, The Netherlands*
[b] *Centre for Tactile Internet with Human-in-the-Loop (CeTI), Dresden, Germany*
[c] *Institute of Theoretical Computer Science, Dresden University of Technology, Dresden, Germany*
[d] *Saarland University, Saarland Informatics Campus, Germany*

A R T I C L E   I N F O

Keywords:
Configurable systems
Causality
Formal methods
Feature interactions
Variability
Root causes

A B S T R A C T

The detection and understanding of reasons for defects and inadvertent behavior in software is challenging due to its ever increasing complexity. One major aspect contributing to this complexity is the multitude of features a user might select from in *configurable systems*. In this article, we tackle this challenge by introducing the notion of *feature causality* that identifies features and their interactions which are the reasons for a system showing certain functional and non-functional properties seen as *effects*. Feature causality operates at the level of system configurations and is based on *counterfactual reasoning*, inspired by the seminal definition of actual causality by Halpern and Pearl.

Towards turning feature causality into meaningful explanations for the reasons why an effect emerges, we present various *explication* methods, e.g., by cause–effect covers, quantifications of causal impacts based on notions like *responsibility* and *blame*, causal reasoning with uncertainty, and feature interactions. Through a close connection of feature causality to prime implicants, we derive algorithms to effectively compute feature causes and causal explications. By means of an evaluation on a wide range of configurable software systems, including community benchmarks and real-world systems, we demonstrate the feasibility of our approach: We illustrate how our notion of causality facilitates to identify root causes, estimate the impact of features on effect properties, and detect feature interactions.

1. Introduction

Most of nowadays computing systems are configurable, offering a wide variety of configuration options and parameters from which users can control the functionalities of the system. A common view on configuration options is by *features* that encapsulate optional or incremental units of functionality (Zave, 2001) and pave the way for the well-established concept of feature-oriented software engineering (Apel et al., 2013). The features of a configurable system can influence critical functional properties such as safety, and non-functional properties such as low probability of failure or performance. Often, the configuration spaces are exponential in the number of features, rendering the detection, prediction, and explanation of defects and inadvertent behavior challenging tasks. While there are specifically tailored methods that tackle this challenge and enable configurable systems analysis (Thüm et al., 2014), research on their *explainability* is still in its infancy (Baier et al., 2021). The huge amounts of system configurations and corresponding analysis results, bug reports, or other feature-dependent properties demand techniques for a meaningful and feasible interpretation.

In this article, we present a set of fundamental concepts and methods to identify and interpret properties of configurable systems *at the level of features* by *causal reasoning*. We introduce the notion of *feature causes* as those feature activations or deactivations that are the reason for emergent system behaviors given as a set of *effect configurations*. This notion takes inspiration from the seminal counterfactual definition of *actual causality* by Halpern and Pearl (2001a), Halpern (2015), also known as *HP causality*, that seeks to determine

☆ Editor: Laurence Duchien.

☆☆ The authors are supported by the DFG, Germany through the Collaborative Research Center TRR 248 (see https://perspicuous-computing.science, project ID 389792660) and the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy).

* Corresponding author at: Formal System Analysis Group, Eindhoven University of Technology, Eindhoven, The Netherlands.
E-mail addresses: c.dubslaff@tue.nl (C. Dubslaff), kallistos@cs.uni-saarland.de (K. Weis), christel.baier@tu-dresden.de (C. Baier), apel@cs.uni-saarland.de (S. Apel).

https://doi.org/10.1016/j.jss.2023.111915
Received 1 March 2023; Received in revised form 15 September 2023; Accepted 25 November 2023
Available online 7 December 2023

the causes for *effect events*. As with HP causality, feature causality assumes full information about the situations where the effect emerges. To enable causal reasoning also on incompletely specified sets of effect configurations or sets of valid system configurations, we extend this view and present a generalization of feature causes by means of *feature precauses*. Incomplete specifications of sets of configurations arise, e.g., when variability spaces are not known (Thüm, 2020), the analysis of the effect involves noise, e.g., when using approximative methods for determining non-functional effect properties, or when an exhaustive analysis is impossible or infeasible, e.g., when relying on real-world bug reports or testing.

Relevant analysis and reasoning tasks that we then address include to determine the set of feature causes for a bug to emerge, the degree to which some configurations are responsible for bad system performance, how robust are feature causes w.r.t. uncertainty in emerging effects, or which features necessarily have to interact for inadvertent behavior.

We develop techniques to perform such tasks in a very generic way, i.e., independent from language-, architecture-, nor environment-specific properties. In fact, our methods are applicable within any effective method to analyze or test variability-aware properties that describe the effect for which reasons are of interest. To this end, causal reasoning on both variability-aware white-box and black-box analyses is supported, complementing existing causal reasoning techniques for the detection of root causes: Approaches such as delta-debugging (Zeller, 2002; Cleve and Zeller, 2005), causal testing (Johnson et al., 2020), or causal trace analysis (Beer et al., 2012) require a white-box analysis that operates at the level of code and are not variability-aware. Hence, they usually would have to be applied on a multitude of system configurations for a variability-aware causal analysis, suffering from a combinatorial blowup well known to arise in configurable systems.

Explications from feature causality. Since features correspond to system functionalities specified by software engineers, they often have a dedicated meaning in the target application domain (Apel et al., 2013). To this end, defects (and other behaviors of interest) detected at the level of features can provide important insights for the resolution of *variability bugs* (Garvin and Cohen, 2011; Rhein et al., 2018; Abal et al., 2018) and *configuration-dependent behavior* (Siegmund et al., 2012; Siegmund et al., 2015; Guo et al., 2018; Nair et al., 2020). They hence are certainly more informative and actionable than low-level program traces that do not include variability information. We introduce and discuss several means to explicate reasons for properties that can be obtained from feature causes and precauses: concise logic formulas by a new method called *distributive law simplification (DLS)*, cause–effect covers, feature interactions, and causal measures by means of responsibility and blame (Chockler and Halpern, 2004). With explicated feature causality at hand, developers may choose to focus on those feature implementations identified as root causes of bugs or simply disallow or coordinate the activation of certain features when defects are related to them. We envision applications of feature causality at those development phases where analysis methods are used, for instance, in software product line engineering. Also in production-level deployments our techniques shall be useful to optimize software through causally relevant configurations.

Evaluation. We present algorithms to compute feature causes, feature precauses, and causal explications for them. Our prototypical implementation relies on *binary decision diagrams (BDDs)* (Bryant, 1986) and the computation of prime implicants using the de-facto standard two-level logic minimizer Espresso (McGeer et al., 1993). By means of an analysis of several configurable systems, including community benchmarks and real-world systems, we investigate feature causes and their properties. We demonstrate that our notion of feature causes and methods to represent them help to pinpoint features relevant for the configurable system's properties and illustrate how feature interactions can be detected and quantified. In particular, our evaluation is driven by the following research questions:

(RQ1) Can feature causes be effectively computed in real-world settings and support the detection of reasons for different effects of interest?
(RQ2) Do DLS representation, cause–effect covers, and degrees of responsibility and blame provide concise causal explications?
(RQ3) Is feature causality beneficial for guiding the configuration of systems under variability-aware constraints?
(RQ4) Can feature interactions and configuration-dependent anomalies be detected and isolated based on feature causality?
(RQ5) To which extend can feature precauses for underspecified effect sets already provide insights on causal relationships?

Contributions. In summary, our contributions are:

(1) We introduce the notion of *feature causality* inspired by the well-established counterfactual definition of actual causality by Halpern and Pearl (2001a), Halpern (2015).
(2) We show that feature causes for effects given as sets of configurations coincide with certain prime implicants that cover the effect configurations, leading to an algorithm to effectively compute all feature causes (Strzemecki, 1992).
(3) We extend feature causality and algorithms to support incomplete specifications and uncertainties of effects and configuration space, leading to *feature precauses*.
(4) We provide methods to interpret and represent feature causes and precauses by propositional formulas, cause–effect covers, responsibility and blame, and potential feature interactions.
(5) We offer a BDD-based prototype to compute and represent feature causes, feature precauses, and feature interactions.
(6) We conduct several experiments illustrating how to determine and reason about feature causes in different realistic settings.

About this article. This article is an extended version of the conference publication titled "Causality in Configurable Software Systems" (Dubslaff et al., 2022). The main additional material not presented in the conference version comprises the definition, computation, and evaluation of *feature precauses* (see above (3) and parts of (4), (5), and (6)). Besides this, the article provides full proofs, additional technical details and examples, a formal comparison to the binary case of HP causality, as well as discussions on cause–effect prime covers and their relation to minimal sum of products from circuit optimization. The source code of our implementation and raw data to reproduce our experiments are publicly available Dubslaff et al. (2023a,b).

2. Background

In this section, we revisit basic concepts and notions from logics and configurable systems used throughout the paper.

Interpretations. A *partial interpretation* over a set X is a partial mapping $\partial : X \rightharpoonup \{\mathtt{true}, \mathtt{false}\}$. We denote by $\mathsf{supp}(\partial)$ the *support* of ∂, i.e., the set of all elements $x \in X$ where $\partial(x)$ is defined. We say that ∂ is a *total interpretation* if $\mathsf{supp}(\partial) = X$ and denote by $\Delta(X)$ and $\Theta(X)$ the set of partial and total interpretations, respectively. Given a partial interpretation $\partial \in \Delta(X)$, we define its semantics $[\![\partial]\!] \subseteq \Theta(X)$ as the set of all total interpretations $\theta \in \Theta(X)$ where for all $x \in \mathsf{supp}(\partial)$ we have $\partial(x) = \theta(x)$. For a set of partial interpretations $\mathcal{P} \subseteq \Delta(X)$, we define $[\![\mathcal{P}]\!] = \bigcup_{\partial \in \mathcal{P}} [\![\partial]\!]$. The x-*expansion* of a partial interpretation $\partial \in \Delta(X)$ is the partial interpretation $\partial\!\uparrow_x \in \Delta(X)$ where $\mathsf{supp}(\partial\!\uparrow_x) = \mathsf{supp}(\partial) \backslash \{x\}$ and where $\partial\!\uparrow_x(y) = \partial(y)$ for all $y \in \mathsf{supp}(\partial) \backslash \{x\}$.

We formalize switching of polarities in interpretations, i.e., mapping of true to false assignments and vice versa, by a function

$$\mathsf{switch} : \wp(X) \times \Theta(X) \to \Theta(X)$$

where for any $Y \subseteq X$ and $\theta \in \Theta(X)$, we have $\mathsf{switch}(Y, \theta)(y) = \theta(y)$ if $y \notin Y$ and $\mathsf{switch}(Y, \theta)(y) = \neg\theta(y)$ otherwise.

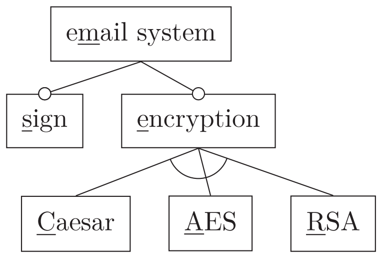

Fig. 1. Feature diagram for the email system example.

Covers and prime implicants. Let $\mathcal{P}, \mathcal{P}_0, \mathcal{P}_1 \subseteq \Delta(X)$ be three sets of partial interpretations. We say that $\mathcal{P}_1$ is *covered* by $\mathcal{P}$ (or alternatively: $\mathcal{P}$ is a *cover of/covers* $\mathcal{P}_1$) iff $[\![\mathcal{P}_1]\!] \subseteq [\![\mathcal{P}]\!]$. Further, $\mathcal{P}$ is a $*$*-cover of* $\mathcal{P}_1$ *relative to* $\mathcal{P}_0$ if $\mathcal{P}$ is a cover of $\mathcal{P}_1$ that is covered by $\Theta(X) \backslash [\![\mathcal{P}_0]\!]$, i.e., $[\![\mathcal{P}_1]\!] \subseteq [\![\mathcal{P}]\!] \subseteq \Theta(X) \backslash [\![\mathcal{P}_0]\!]$. In $*$-covers, $[\![\mathcal{P}_1]\!]$ and $[\![\mathcal{P}_0]\!]$ intuitively stand for the sets of interpretations for which a Boolean function is known to be $\texttt{true}$ and $\texttt{false}$, respectively. The set of all other interpretations $\Theta(X) \backslash [\![\mathcal{P}_0 \cup \mathcal{P}_1]\!]$ is the $*$*-set*, constituting the set of unknown or "don't care" interpretations. Note that $\mathcal{P}$ is a cover of $\mathcal{P}_1$ iff $\mathcal{P}$ is a $*$-cover of $\mathcal{P}_1$ relative to $\varnothing$ and hence, $*$-covers can be seen as a more general form of covers. A $*$-cover $\mathcal{P}$ of $\mathcal{P}_1$ relative to $\mathcal{P}_0$ is *minimal* if there is no $*$-cover $\mathcal{P}'$ of $\mathcal{P}_1$ relative to $\mathcal{P}_0$ where $|\mathcal{P}'| < |\mathcal{P}|$. Note that minimal $*$-covers are not uniquely defined and there can be multiple minimal $*$-covers $\mathcal{P}$ for a given $\mathcal{P}_0$ and $\mathcal{P}_1$. Our notions also extend to $\mathcal{P}$, $\mathcal{P}_0$, or $\mathcal{P}_1$ being singletons (or sets of total interpretations), e.g., a partial configuration $\partial \in \Delta(X)$ covers a set of total interpretations $\mathcal{T} \subseteq \Theta(X)$ iff $\mathcal{T} \subseteq [\![\partial]\!]$. We call a partial interpretation $\partial \in \Delta(X)$ a *prime implicant* of $\mathcal{P}$ iff $\mathcal{P}$ covers ∂ and $[\![\partial\!\uparrow_x]\!] \nsubseteq \mathcal{P}$ for all $x \in \mathsf{supp}(\partial)$. A *prime* $*$*-cover* of $\mathcal{P}_1$ relative to $\mathcal{P}_0$ is a cover of $\mathcal{P}_1$ that only comprises prime implicants of $\Theta(X) \backslash [\![\mathcal{P}_0]\!]$. We denote by $\mathbb{P}(\mathcal{P}_1, \mathcal{P}_0)$ the set of *prime* $*$*-covers* of $\mathcal{P}_1$ relative to $\mathcal{P}_0$ and by $m\mathbb{P}(\mathcal{P}_1, \mathcal{P}_0)$ the *minimal prime* $*$*-covers* of $\mathcal{P}_1$ relative to $\mathcal{P}_0$.

Propositional logics. A *propositional logic formula* over a set X is an expression defined by the grammar $\phi \;=\; \texttt{true} \mid \texttt{false} \mid x \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$ where x ranges over X. The *length* $|\phi|$ of a formula ϕ is recursively defined by $|\texttt{true}| = |\texttt{false}| = |x| = 1$, $|\neg\phi| = |\phi| + 1$, and $|\phi_0 \wedge \phi_1| = |\phi_0 \vee \phi_1| = |\phi_0| + |\phi_1| + 1$. For a partial interpretation $\partial \in \Delta(X)$, we write $\partial \models \phi$ if either $\phi = \texttt{true}$, $\phi = x \in \mathsf{supp}(\partial)$ and $\partial(x) = \texttt{true}$, $\phi = \neg\psi$ and not $\partial \models \psi$, $\phi = \phi_0 \wedge \phi_1$ and $\partial \models \phi_0$ and $\partial \models \phi_1$, and $\phi = \phi_0 \vee \phi_1$ and $\partial \models \phi_0$ or $\partial \models \phi_1$. The semantics of ϕ is the set of all satisfying total interpretations $[\![\phi]\!] = \{\theta \in \Theta(X) : \theta \models \phi\}$.

Configurable systems. A widely adopted concept to model configurable systems is by means of features (Apel et al., 2013). *Features* encapsulate optional or incremental units of functionality (Zave, 2001) and describe commonalities and variabilities of whole families of systems. At an abstract level, we identify Boolean configuration options with features of the system and fix a set of features F. We call a total interpretation $\theta \in \Theta(F)$ over F a *configuration*, which we usually describe by listing the selected features, i.e., the features $x \in F$ where $\theta(x) = \texttt{true}$. The set of *valid configurations* $\mathsf{Valid} \subseteq \Theta(F)$ comprises those configurations for which there exists a corresponding system implementation. A partial interpretation over F is called *partial configuration*.

Example 1. As the running example, consider a simple email system over features $F = \{m, s, e, c, a, r\}$, formalizing the base e<u>m</u>ail system functionality, optional features for <u>s</u>igning and <u>e</u>ncryption, and encryption methods <u>C</u>aesar, <u>A</u>ES, and <u>R</u>SA. Valid configurations are usually specified through *feature diagrams* (Kang et al., 1990). These are hierarchical structures over features describing the constraints imposed on configurations to render them valid. An example of such a diagram for our running example is provided in Fig. 1. In essence, the feature of the root of the diagram has always to be included in any valid configuration, here "m". If some son is active in a configuration, then also its parent has to be included. The other way around, if no $\circ$ is drawn at the top of a node (indicating "optional features"), then also the activation of a parent feature imposes activation of the son. Non-connected branches stand for logical conjunctions and connected branches for exclusive disjunctions over the connected sons. Thus, for the encryption features, we assume that exactly one can be selected. The described variability constraints for the email system specified in the feature diagram of Fig. 1 lead to valid configurations $\mathsf{Valid} = \{\, m, mec, mea, mer, ms, msec, msea, mser \,\}$. $\diamond$

3. Feature causality

The notion of *causality* has been extensively studied in philosophy, social sciences, and artificial intelligence (Good, 1959; Eells, 1991; Pearl, 2009; Williamson, 2009). We focus here on *actual causality*, describing binary causal relationships between cause events C and effect events E. Halpern and Pearl formalized actual causality based on the concept of *counterfactual dependencies* (Lewis, 1973) using a structural-equation approach (Halpern, 2015; Halpern and Pearl, 2001a,b). The idea of counterfactual reasoning (Wachter et al., 2017) relies on the assumption that E would not have happened if C had not happened before, which corresponds to the "but-for" test used in law (Spellman and Kincannon, 2001; Wachter et al., 2017).

In this section, we take inspiration of the definition by (Halpern and Pearl, 2001a; Halpern, 2015) to establish a notion of causality at the level of features. Here, we interpret the selection of features as events considered for actual causality. The basic reasoning task we address then amounts to *determine those feature selections that cause a given effect property*. Examples for effect properties are "the execution time is longer than five minutes" or "the system crashes".

We assume to have described the effect properties as *effect set* $\mathsf{Effect} \subseteq \mathsf{Valid}$ of valid configurations for which an effect property can be observed. Elements of Effect are called *effect instances*. All other valid configurations in $\mathsf{Valid} \backslash \mathsf{Effect}$ are assumed not to exhibit the effect (called *non-effect instances*). Feature selections are naturally specified by partial configurations. Clearly, a partial configuration γ can only be a cause of the effect if γ ensures the effect to emerge, i.e., all valid configurations that are covered by γ are effect instances. Furthermore, following counterfactual reasoning, we require for γ being a cause that, if we would select features of γ differently, there might be a configuration for which the effect does not emerge. These two intuitive conditions on causality are reflected in our formal definition of causes of Effect w.r.t. Valid:

Definition 1. A *feature cause* of an effect Effect w.r.t. valid configurations Valid is a partial configuration $\gamma \in \Delta(F)$ where

(FC1) $\varnothing \neq [\![\gamma]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect}$ and
(FC2) $[\![\gamma\!\uparrow_x]\!] \cap \mathsf{Valid} \nsubseteq \mathsf{Effect}$ for all $x \in \mathsf{supp}(\gamma)$.

$\mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$ denotes the set of all causes for Effect w.r.t. Valid.

In case **(FC1)** holds for a partial configuration γ, we say that γ is *sufficient* for Effect w.r.t. Valid (Garvin and Cohen, 2011). This sufficiency is considered in the scope of Valid, since for configurations not contained in Valid it usually cannot be decided in practice whether an effect emerges or not. The counterfactual nature of **(FC2)** ensures that for every feature cause γ and $x \in \mathsf{supp}(\gamma)$ there is a *counterfactual witness* $\overline{\eta} \in [\![\gamma\!\uparrow_x]\!] \cap (\mathsf{Valid} \backslash \mathsf{Effect})$. That is, a valid feature configuration where the effect does not emerge but where changing one feature selection may yield an effect instance. Note that **(FC2)** ensures minimality of the feature cause w.r.t. its support, i.e., dropping conditions on interpretations of features necessarily leads to a partial configuration that is not sufficient for the effect anymore. In the formal definition of Halpern and Pearl causality (Halpern, 2015), counterfactuality and minimality are stated in two distinct conditions (see also Section 3.2).

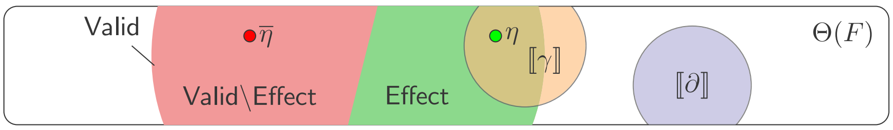

Fig. 2. Configuration sets for feature causality.

We usually denote configurations in Effect by η, counterfactual witnesses in Valid\Effect by $\bar{\eta}$, and feature causes by γ. Fig. 2 depicts the relation between valid configurations, effects, causes, and counterfactual witnesses.

Example 2. Let us continue our running example of the configurable email system introduced in Example 1. We consider an effect property reflecting "long decipher time", e.g., that it takes in average more than three months for an attacker to decrypt an email. Assume that this effect property can be observed by configurations in which AES or RSA are selected, i.e., $\mathsf{Effect} = \{mea, mer, msea, mser\}$. Conversely, in all valid configurations in which AES and RSA are not selected, the effect does not emerge. In this setting, the encryption features AES and RSA are both causes since all valid configurations with either feature show the effect. Considered in isolation, AES and RSA are not necessary for the effect, as one can choose the other encryption feature (RSA or AES, respectively) to ensure the effect. The sign feature does not trigger the effect and is not a cause.

Interestingly, a further cause is given by selecting the encryption feature and explicitly deselecting the Caesar feature, illustrating that also explicitly not selecting features might be a cause of some effect. This hints at the fact that causes can be represented in different ways, addressed later in the paper.

Formalizing this intuition, we check whether the three partial configurations γ_a, γ_r, and $\gamma_{e\bar{c}}$ given by

(i) $\mathsf{supp}(\gamma_a) = \{a\}$ with $\gamma_a(a) = \mathtt{true}$,
(ii) $\mathsf{supp}(\gamma_r) = \{r\}$ with $\gamma_r(r) = \mathtt{true}$, and
(iii) $\mathsf{supp}(\gamma_{e\bar{c}}) = \{e, c\}$ with $\gamma_{e\bar{c}}(e) = \mathtt{true}$ and $\gamma_{e\bar{c}}(c) = \mathtt{false}$

are feature causes of Effect w.r.t. Valid according to Definition 1: First, note that $[\![\gamma_{e\bar{c}}]\!] \cap \mathsf{Valid} = ([\![\gamma_a]\!] \cup [\![\gamma_r]\!]) \cap \mathsf{Valid} = \mathsf{Effect}$ and both, $[\![\gamma_a]\!] \cap \mathsf{Valid}$ and $[\![\gamma_r]\!] \cap \mathsf{Valid}$, are non-empty. Hence, **(FC1)** is fulfilled for $\gamma_{e\bar{c}}$, γ_a, and γ_r. To check **(FC2)**, we observe that $[\![\gamma_a {\uparrow}_a]\!] = [\![\gamma_r {\uparrow}_r]\!] = \Theta(F)$ and

$$[\![\gamma_{e\bar{c}} {\uparrow}_e]\!] \cap \mathsf{Valid} \ = \ \mathsf{Effect} \cup \{m, ms\}, \text{ and}$$
$$[\![\gamma_{e\bar{c}} {\uparrow}_c]\!] \cap \mathsf{Valid} \ = \ \mathsf{Effect} \cup \{mec, msec\}.$$

Hence, m is a counterfactual witness for γ_a, γ_r, and $\gamma_{e\bar{c}}$ w.r.t. a, r, and e, respectively, while mec can serve as such for $\gamma_{e\bar{c}}$ and c. It is easy to check that there are no further feature causes since for all other partial configurations sufficient for Effect w.r.t. Valid (see **(FC1)**) there are expansions towards γ_a, γ_r, or $\gamma_{e\bar{c}}$ (thus, violating **(FC2)**). ⋄

3.1. Effect properties and effect sets

Our definition of feature causality relies on a given effect set, which is assumed to comprise all those valid configurations where the effect property holds. We now elaborate more on how to obtain effect sets from analyzing configurable systems. In fact, our generic definition supports a multitude of effect properties for which the only assumption is that there is an effective method to determine all configurations in which the effect property holds. Such methods include variability-aware white-box analyses (Weber et al., 2021; Velez et al., 2021) or formal analysis through model checking (Cordy et al., 2013a; Chrszon et al., 2018) where the source code or operational behavior of system variants is accessible. Further, also black-box analyses are eligible to specify effect sets, where only the behavior of the system can be observed without knowledge about the innerworkings, relying on testing or sampling (Guo et al., 2018; Kaltenecker et al., 2019). In the following paragraphs, we exemplify how to obtain effect sets from analysis results. The discussed effect properties reflect the instances of the experimental evaluation section (see Section 5) and do not claim to be exhaustive.

Functional properties. To reason about causality w.r.t. functional properties, the effect set can be determined by variability-aware static analysis (ter Beek et al., 2019; Rhein et al., 2018) or model checking (Plath and Ryan, 2001; Classen et al., 2013; Apel et al., 2013). In the latter case, effect properties can be formalized, e.g., in a temporal logic such as LTL (Pnueli, 1977) or CTL (Clarke et al., 1986). Model checking configurable systems against LTL and CTL properties has broad tool support (e.g. Classen et al., 2012; Cordy et al., 2013a). Given a formula φ that specifies the effect property, these tools return all the valid configurations $\theta \in \mathsf{Valid}$ whose corresponding system variants satisfy φ, i.e., the *effect set*

$$\mathsf{Effect}_{\varphi} \ = \ \{\theta \in \mathsf{Valid} : \theta \models \varphi\}$$

Since model checking is based on an exhaustive analysis, an analysis also exposes those valid configurations for which the effect property does not hold. The same is possible for variability-aware static analysis (Rhein et al., 2018; Bodden et al., 2013).

Non-functional properties. Besides functional properties, also non-functional properties of configurable systems can serve as effect property and give rise to an effect set. Let $\rho : \mathsf{Valid} \to \mathbb{R}$ be a function that results from a quantitative analysis of the configurable system in question, providing a quantitative measure for all valid configurations. Values $\rho(\theta)$ for a valid configuration $\theta \in \mathsf{Valid}$ may stand for the performance achieved, the probability of failure, or the energy consumed in the system variant that corresponds to θ. To obtain ρ for real-world systems, Siegmund et al. (2015), Siegmund et al. (2012) presented a black-box method to generate linear-equation models for performance measures by multivariable linear regression on sampled configurations. Other black-box approaches rely on regression trees (Guo et al., 2018), Fourier learning (Zhang et al., 2015), or probabilistic programming (Dorn et al., 2020). Related white-box approaches use insights of local measurements and taint analysis information (Velez et al., 2021) or profiling information (Weber et al., 2021).

An orthogonal formal white-box analysis on operational models with quantitative information (such as probabilities, costs, etc.) is provided through variability-aware probabilistic model checking (Dubslaff et al., 2015; ter Beek et al., 2016). Effect properties for such approaches are specified in quantitative variants of temporal logic such as probabilistic CTL (Hansson and Jonsson, 1994). These approaches have been implemented in the tools like ProFeat (Chrszon et al., 2018) and QFLan (Vandin et al., 2018) and have shown practical applicability in various experimental studies.

Given ρ that results from one of the analysis approaches mentioned above, an effect set can be specified by imposing a threshold $\tau \in \mathbb{R}$ combined with a comparison relation $\sim$ towards *threshold effect sets*

$$\mathsf{Effect}_{\rho\sim\tau} \ = \ \{\theta \in \mathsf{Valid} : \rho(\theta) \sim \tau\}.$$

Example 3. In Example 2, we informally specified the effect of a "long decipher time" as taking more than three months to decrypt an email without having the encryption key available. By a variability-aware quantitative analysis on the email system, we may obtain a function ρ that, for a configuration θ, returns the minimal time in years to decipher an email sent with the system variant corresponding to θ. Analysis results could be, e.g., $\rho(\theta)=0$ with no encryption, $\rho(\theta)=10^{-7}$ with Caesar, $\rho(\theta)=1$ with AES, and $\rho(\theta)=2$ with RSA selected in θ, respectively. Then, $\mathsf{Effect}_{\rho>0.25}$ provides the effect set Effect of Example 2. ⋄

On computing effect sets. The effect set and the set of valid configurations can be of exponential size in the number of features. An efficient computation of these sets depends on the analysis techniques used and are independent from our causal framework. However, specifically tailored variability-aware analysis techniques can tackle the exponential blowup, e.g., through symbolic representation of family models (Thüm et al., 2014; Dubslaff, 2019).

3.2. Relation to HP causality

The original definition of actual causality by (Halpern, 2015) relies on a structural-equation approach and comprises three conditions: effectiveness, counterfactuality, and minimality. Compared to our definition of feature causes presented in Definition 1, their definition supports non-Boolean evaluation of variables and the distinction between endogenous and exogenous variables, specifying variables inherently contained in the system and those that can be subject of external influences, respectively. Directly transferring their notion of actual causality to the setting of configurable systems leads to the following definition of *HP feature causality*:

Definition 2. An *HP feature cause* of an effect Effect w.r.t. valid configurations Valid is a partial configuration $\gamma \in \Delta(F)$ where

(FCa) $[\![\gamma]\!] \cap \mathsf{Effect} \neq \varnothing$ and $[\![\gamma]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) = \varnothing$,
(FCb) there is a partial configuration $\partial \in \Delta(F)$ with $\mathsf{supp}(\gamma)=\mathsf{supp}(\partial)$ such that $[\![\partial]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) \neq \varnothing$, and
(FCc) γ is minimal; for all $\varnothing \neq S \subseteq \mathsf{supp}(\gamma)$ either **(FCa)** or **(FCb)** is not satisfied for $\gamma{\uparrow}_S$.

In the following, we draw the connection between HP feature causes and our definition of feature causes provided in Definition 1. First, observe that for the counterfactual **(FCb)** of HP feature causes, a single counterfactual witness suffices:

Lemma 1. $\mathsf{Valid}\backslash\mathsf{Effect} \neq \varnothing$ *iff* **(FCb)** *for some* $\gamma \in \Delta(F)$.

Proof. **(⇒):** Let $\theta \in \mathsf{Valid}\backslash\mathsf{Effect}$, which exists by assumption and let $\gamma \in \Delta(F)$. For $\partial = \theta{\uparrow}_{F\backslash\mathsf{supp}(\gamma)}$ we then have $\theta \in [\![\partial]\!]$ and hence, $\theta \in [\![\partial]\!] \cap \mathsf{Valid}\backslash\mathsf{Effect}$. Thus, **(FCb)** is fulfilled for γ.

(⇐): Let $\gamma \in \Delta(F)$. By **(FCb)** there is $\partial \in \Delta(F)$ with $\mathsf{supp}(\gamma)=\mathsf{supp}(\partial)$ and $[\![\partial]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) \neq \varnothing$. Then there is $\theta \in [\![\partial]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect})$ and thus $\theta \in \mathsf{Valid}\backslash\mathsf{Effect}$, leading to $\mathsf{Valid}\backslash\mathsf{Effect} \neq \varnothing$. □

Then, we can show that minimality condition **(FCc)** agrees with **(FC2)** under the condition of **(FCa)** and **(FCb)** to hold:

Lemma 2. *Let* $\gamma \in \Delta(F)$ *for which* **(FCa)** *and* **(FCb)** *hold. Then,* **(FCc)** *holds for* γ *iff* **(FC2)** *holds for* γ.

Proof. First observe that due to Lemma 1 we have that for any $S \subseteq \mathsf{supp}(\gamma)$ we have **(FCb)** to hold for $\gamma{\uparrow}_S$ iff **(FCb)** holds for γ.

(⇒): Let $S=\{x\}$ for some $x \in \mathsf{supp}(\gamma)$, then $\gamma{\uparrow}_x$ validates **(FCb)**. Since γ validates **(FCc)**, **(FCa)** is not valid for $\gamma{\uparrow}_x$. Further, $[\![\gamma]\!] \cap \mathsf{Effect} \neq \varnothing$ due to **(FCa)** and thus also $[\![\gamma{\uparrow}_x]\!] \cap \mathsf{Effect} \neq \varnothing$. Hence, $[\![\gamma{\uparrow}_x]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) \neq \varnothing$ and thus, $[\![\gamma{\uparrow}_x]\!] \cap \mathsf{Valid} \nsubseteq \mathsf{Effect}$, i.e., **(FC2)** holds.

(⇐): Due to **(FC2)** we have $[\![\gamma{\uparrow}_x]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) \neq \varnothing$ for all $x \in \mathsf{supp}(\gamma)$. By **(FCa)**, $|\mathsf{supp}(\gamma)|>0$, hence there is also an $x \in \mathsf{supp}(\gamma)$ such that $[\![\gamma{\uparrow}_x]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) \neq \varnothing$. Let $S \subseteq \mathsf{supp}(\gamma)$ with $|S| \geq 1$. Due to the remark at the beginning of this proof **(FCb)** holds for $\gamma{\uparrow}_S$. However, since $[\![\gamma{\uparrow}_x]\!] \subseteq [\![\gamma{\uparrow}_S]\!]$ for all $x \in S$ we have $[\![\gamma{\uparrow}_x]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) \neq \varnothing$, contradicting **(FCa)** for $\gamma{\uparrow}_S$. Hence, **(FCc)**. □

Finally, we obtain the following proposition providing the direct connection of actual causes by Halpern and Pearl in the setting of configurable systems and feature causality:

Proposition 1. *For* $\mathsf{Valid}\backslash\mathsf{Effect} \neq \varnothing$*, any HP feature cause for* Effect *w.r.t.* Valid *is a feature cause for* Effect *w.r.t.* Valid *and vice versa. Otherwise, i.e., if* $\mathsf{Valid}\backslash\mathsf{Effect} = \varnothing$*, then*

- *there is no HP feature cause for* Effect *w.r.t.* Valid*, and*
- *if* $\mathsf{Valid} \neq \varnothing$*, then* $\mathsf{Causes}(\mathsf{Effect},\mathsf{Valid}) = \{\alpha\}$ *with* $\mathsf{supp}(\alpha)=\varnothing$.

Proof. First, let $\mathsf{Valid}\backslash\mathsf{Effect} \neq \varnothing$. **(FCa)** coincides with **(FC1)**. Due to Lemma 1 we also have **(FCb)**. The claim then directly follows in combination with Lemma 2. If $\mathsf{Valid}\backslash\mathsf{Effect} = \varnothing$, then **(FCb)** is violated (see Lemma 1) and hence, there is no HP feature cause for Effect w.r.t. Valid. Further, **(FC2)** can only be satisfied for a $\gamma \in \Delta(F)$ if $\mathsf{supp}(\gamma)=\varnothing$ and **(FC1)** can only be satisfied if $\mathsf{Valid} \neq \varnothing$. Thus, we have $\mathsf{Causes}(\mathsf{Effect},\mathsf{Valid}) = \{\alpha\}$ with $\mathsf{supp}(\alpha)=\varnothing$. □

It is merely a philosophical discussion whether one allows for empty causes, i.e., feature causes with empty support and covering all valid feature configurations. We defined feature causes allowing for an empty support to distinguish between those cases where no causal dependencies arise, which is the case of an empty effect set. In the latter case, one could not distinguish between no causal dependencies due to all valid configurations showing the effect or none.

From effect configurations to effect sets. Our definition of HP feature causes can be embedded into HP causality on binary domains, allowing for structural equations on Boolean variables for feature selection (Halpern, 2015). Given an effect property as a propositional formula, actual causes are then considered w.r.t. a *contingency* that has a similar role as a single effect configuration in our setting. Causes w.r.t. a contingency then can be easily extended to causes w.r.t. an effect set.

3.3. Computation of feature causes

For a given effect set Effect and a set of valid configurations Valid along with a partial configuration ∂, Definition 1 directly provides a polynomial-time algorithm to decide whether ∂ is a cause of Effect w.r.t. Valid by checking **(FC1)** and **(FC2)**. From this, we obtain a simple approach to compute the set $\mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$ by successively checking expansions for sets of features applied on elements in Effect as candidates for causes. Since there might be exponentially many such expansions, this approach easily renders infeasible already within a small number of features.

We now present a practical algorithm to compute the set of causes, which relies on a connection to the notion of *prime implicants* (see Section 2). In the setting of features, a prime implicant of a set of partial configurations $\mathcal{P} \subseteq \Delta(F)$ is a partial configuration $\partial \in \Delta(F)$ where $\mathcal{P}$ covers ∂ and $[\![\partial{\uparrow}_x]\!] \nsubseteq \mathcal{P}$ for all $x \in \mathsf{supp}(\partial)$. Here, observe the similarity to **(FC2)** of Definition 1. Further, this connection is not immediately visible when considering HP causality (see Definition 2). Towards establishing the feature cause computation algorithm, we first require a technical lemma:

Lemma 3. *For any partial configuration* $\partial \in \Delta(F)$

$$[\![\partial]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect} \quad \textit{iff} \quad [\![\partial]\!] \subseteq \big(\Theta(F)\backslash\mathsf{Valid}\big) \cup \mathsf{Effect}.$$

Proof. Let $\theta \in [\![\partial]\!]$.

($\Rightarrow$): If $\theta \in \mathsf{Valid}$, then $\theta \in \mathsf{Effect}$ and thus, $\theta \in \big(\Theta(F)\backslash\mathsf{Valid}\big) \cup \mathsf{Effect}$. Otherwise, $\theta \in \Theta(F)\backslash\mathsf{Valid}$ and thus also $\theta \in \big(\Theta(F)\backslash\mathsf{Valid}\big) \cup \mathsf{Effect}$.

($\Leftarrow$): If $\theta \in \mathsf{Effect}$, then the statement is clear due to $\mathsf{Effect} \subseteq \mathsf{Valid}$ and thus $\theta \in [\![\partial]\!] \cap \mathsf{Valid}$. Otherwise, $\theta \notin \mathsf{Effect}$ and thus, $\theta \in \Theta(F)\backslash\mathsf{Valid}$. Hence $\theta \notin \mathsf{Valid}$, such that $\theta \notin [\![\partial]\!] \cap \mathsf{Valid}$. □

Following this lemma, every cause of Effect w.r.t. Valid is also an implicant of $\big(\Theta(F)\backslash\mathsf{Valid}\big)\cup\mathsf{Effect}$ due to **(FC1)** and even a prime implicant due to **(FC2)**. Conversely, every prime implicant ∂ of $\big(\Theta(F)\backslash\mathsf{Valid}\big) \cup \mathsf{Effect}$ for which $[\![\partial]\!] \cap \mathsf{Effect} \neq \emptyset$ is a cause due to **(FC1)**. This directly suggests an algorithm to compute causes via prime implicants: Algorithm 1 first generates prime implicants as cause candidates and then removes those candidates that are not sufficient for Effect w.r.t. Valid. Fig. 2 reflects this situation where γ and ∂ are prime implicants with γ being a cause and ∂ not: at least one effect instance is covered by γ, while this is not the case for ∂ and hence would be removed by Algorithm 1.

Algorithm 1: Computation of feature causes

```
    input  : $\mathsf{Effect}, \mathsf{Valid} \subseteq \Theta(F)$ with $\mathsf{Effect} \subseteq \mathsf{Valid}$
    output: $\mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$
1 $\mathcal{P} := \textsc{Compute-Primes}\big((\Theta(F)\backslash\mathsf{Valid}) \cup \mathsf{Effect}\big)$
2 forall $\partial \in \mathcal{P}$ where $[\![\partial]\!] \cap \mathsf{Effect} = \emptyset$ do $\mathcal{P} := \mathcal{P}\backslash\{\partial\}$
3 return $\mathcal{P}$
```

Prime implicants of a set of configurations can be computed in polynomial time in the size of the input set (Strzemecki, 1992), which directly leads to:

Theorem 1. *Given valid configurations* $\mathsf{Valid} \subseteq \Theta(F)$ *and effect set* $\mathsf{Effect} \subseteq \mathsf{Valid}$*, Algorithm 1 computes* $\mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$*, the set of feature causes for* Effect *w.r.t.* Valid*, in polynomial time in* $|\Theta(F)|$.

Proof. By Lemma 3 and Definition 1, we directly obtain that for all $\gamma \in \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$ we have that γ is a prime implicant of $(\Theta(F)\backslash\mathsf{Valid})\cup\mathsf{Effect}$ where $[\![\gamma]\!]\cap\mathsf{Effect} \neq \emptyset$. To this end, Algorithm 1 is complete. For soundness, let $\partial \in \mathcal{P}$ where $\mathcal{P}$ is returned by Algorithm 1. Then ∂ is a prime implicant of $(\Theta(F)\backslash\mathsf{Valid})\cup\mathsf{Effect}$ and by Lemma 3 we have that **(FC2)** is satisfied. Further, by the definition of implicants we obtain that $[\![\partial]\!] \subseteq (\Theta(F)\backslash\mathsf{Valid})\cup\mathsf{Effect}$ and thus $[\![\partial]\!]\cap\mathsf{Valid} \subseteq \mathsf{Effect}$, which in combination with $[\![\partial]\!]\cap\mathsf{Effect} \neq \emptyset$ by Line 2 directly leads to **(FC1)**. Hence, for all $\partial \in \mathcal{P}$ we have $\partial \in \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$ and thus, Algorithm 1 is sound.

Let us now turn to the complexity of Algorithm 1. It is well known that the computation of prime implicants can be done in polynomial time (Strzemecki, 1992) in the size of the number of interpretations. Hence, $|\mathcal{P}|$ is polynomial in $|(\Theta(F)\backslash\mathsf{Valid}) \cup \mathsf{Effect}|$, which is smaller than $|\Theta(F)|$. Furthermore, since the emptiness check in Line 2 can be done by simply iterating over all elements of Effect, the whole algorithm runs in polynomial time in $|\Theta(F)|$. □

Note that the set of valid and effect configurations can be both exponential in the number of features and there might be exponentially many prime implicants (Chandra and Markowsky, 1978) in the worst case. Hence, Algorithm 1 is exponential in the number of features.

Example 4. Let us illustrate the computation of feature causes of Example 2 by Algorithm 1. First notice that

$$\big(\Theta(F) \setminus \mathsf{Valid}\big) \cup \mathsf{Effect} \;=\; \Theta(F) \setminus \{\, m, mec, ms, msec \,\}$$

comprising 60 feature configurations. The prime implicants for this set are computed in Line 1, which yields

$$\mathcal{P} \;=\; \{\, \gamma_{\bar{m}}, \gamma_a, \gamma_r, \gamma_{e\bar{c}}, \gamma_{\bar{e}c} \,\}.$$

Here, we used notations as in Example 2, e.g., $\mathsf{supp}(\gamma_{\bar{e}c}) = \{e, c\}$, $\gamma_{\bar{e}c}(e) = \mathtt{false}$, and $\gamma_{\bar{e}c}(c) = \mathtt{true}$. Clearly, all configurations covered by $\gamma_{\bar{m}}$ or $\gamma_{\bar{e}c}$ are not valid and hence also no effects. Thus, they are removed in Line 2, leading to $\mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid}) = \{\gamma_a, \gamma_r, \gamma_{e\bar{c}}\}$. ⋄

3.4. Effect uncertainty and feature precauses

For our notion of feature causality introduced at the beginning of this section (see Definition 1), we assumed full information about the set of valid configurations Valid and its partition into configurations Effect that show an effect and those Valid \ Effect that do not. While this assumption is reasonable when applied in the context of variability-aware exhaustive analysis (see Section 3.1), there are situations where effect and non-effect configurations cannot be ultimately separated:

(1) The set of valid configurations is unknown, irrelevant, or cannot be explicitly constructed.
(2) There is incomplete information about the set of effect configurations due to non-exhaustive analysis methods, e.g., variability-aware testing or limits on analysis resources such as runtime.
(3) The chosen analysis method inherently incorporates uncertainties, e.g., relies on approximations.

The first case arises, e.g., when the configuration spaces are of such sizes that formal reasoning about valid configurations is infeasible (Thüm, 2020). The second case boils down to not having information about all effect configurations at hand. This also covers practical relevant situations where the effect is partly described, e.g., through bug reports where users report the same bug within different configurations (see Section 3.1). The third case is in particular relevant for non-functional effect properties, e.g., when variability-aware probabilistic model checking or approximative regression methods are chosen as analysis methods to investigate threshold effect sets (see Section 3.1). Then, an exact decision whether the given threshold is met cannot be made due to noise: those quantities close to the threshold cannot be decided up to the precision of the analysis method.

While feature causality cannot directly be used to pinpoint those configuration options that are the reasons for such *effects with uncertainty*, we can still follow the very same counterfactual reasoning to establish *candidates* for causes given the limited access to information, which we call *feature precauses*.

Definition 3. A *feature precause* for effect configurations $\mathsf{Effect}^* \subseteq \Theta(F)$ w.r.t. non-effect configurations $\mathsf{NEffect}^* \subseteq \Theta(F)$ where $\mathsf{Effect}^* \cap \mathsf{NEffect}^* = \emptyset$ is a partial configuration $\gamma \in \Delta(F)$ where

(FPC1) $[\![\gamma]\!] \cap \mathsf{Effect}^* \neq \emptyset$, $[\![\gamma]\!] \cap \mathsf{NEffect}^* = \emptyset$, and
(FPC2) $[\![\gamma{\uparrow}_x]\!] \cap \mathsf{NEffect}^* \neq \emptyset$ for all $x \in \mathsf{supp}(\gamma)$.

We denote by $\mathsf{PCauses}(\mathsf{Effect}^*, \mathsf{NEffect}^*)$ the set of all precauses for effect configurations Effect^* w.r.t. non-effect configurations $\mathsf{NEffect}^*$.

Note the close relation of the effectiveness condition in HP feature causes (see **(FCa)** in Definition 2) and **(FPC1)**. Stated in words, precauses provide minimal conditions on feature selection or deselection (see **(FPC2)**) such that it is possible to show the effect and they do not cover any configuration where it is sure the effect is not emerging (see **(FPC1)**). Our definition of precauses also implements counterfactual reasoning by **(FPC2)**, since relaxing one of the feature selection conditions directly leads to also allow for a possible non-effect configuration, serving as counterfactual witness.

Feature precauses indeed constitute an extension of feature causes, i.e., in case the set of valid configurations can be partitioned into effect and non-effect configurations, precauses coincide with causes. Even more, if the set of valid configurations is known to be Valid, every precause for an effect Effect^* w.r.t. non-effects $\mathsf{NEffect}^*$ is a cause for the effect $\mathsf{Valid}\backslash\mathsf{NEffect}^*$:

Lemma 4. *Let* $\mathsf{Valid} \subseteq \Theta(F)$ *be a set of valid configurations containing effect configurations* $\mathsf{Effect}^* \subseteq \mathsf{Valid}$ *and non-effect configurations* $\mathsf{NEffect}^* \subseteq \mathsf{Valid}$ *such that* $\mathsf{Effect}^* \cap \mathsf{NEffect}^* = \varnothing$. *Further, let* $\mathsf{Effect} = \mathsf{Valid} \backslash \mathsf{NEffect}^*$. *Then*

$$\mathsf{PCauses}(\mathsf{Effect}^*, \mathsf{NEffect}^*) \subseteq \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$$

with meeting equality in case $\mathsf{Effect}^* = \mathsf{Effect}$.

Proof. If $\mathsf{Effect}^* = \varnothing$, then $\mathsf{PCauses}(\mathsf{Effect}^*, \mathsf{NEffect}^*) = \varnothing$. Otherwise $\mathsf{Effect}^* \neq \varnothing$, $\mathsf{NEffect}^* \neq \varnothing$, and $\mathsf{Effect}^* \subseteq \mathsf{Effect}$. Let now $\gamma \in \mathsf{PCauses}(\mathsf{Effect}^*, \mathsf{NEffect}^*)$. For **(FC1)**, **(FPC1)** with $[\![\gamma]\!] \cap \mathsf{Effect}^* \neq \varnothing$ leads to $[\![\gamma]\!] \cap \mathsf{Valid} \neq \varnothing$ and $[\![\gamma]\!] \cap \mathsf{NEffect}^* = \varnothing$ leads to $[\![\gamma]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect}$. Due to **(FPC2)** for all $x \in \mathsf{supp}(\gamma)$ there is $\overline{\eta} \in [\![\gamma\!\uparrow_x]\!] \cap \mathsf{NEffect}^*$ such that $\overline{\eta} \in [\![\gamma\!\uparrow_x]\!] \cap \mathsf{Valid}$ but $\overline{\eta} \notin \mathsf{Effect}$, directly leading to **(FC2)**. Now we show that precauses agree with causes if $\mathsf{Effect}^* = \mathsf{Effect}$. Let $\gamma \in \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$. **(FC1)** with $[\![\gamma]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect} = \mathsf{Valid} \backslash \mathsf{NEffect}^*$ leads to $[\![\gamma]\!] \cap \mathsf{NEffect}^* = \varnothing$ and further $[\![\gamma]\!] \cap \mathsf{Valid} \neq \varnothing$ leads to $[\![\gamma]\!] \cap \mathsf{Effect} = [\![\gamma]\!] \cap \mathsf{Effect}^* \neq \varnothing$, implying **(FPC1)**. Due to **(FC2)** for all $x \in \mathsf{supp}(\gamma)$ there is $\overline{\eta} \in [\![\gamma\!\uparrow_x]\!] \cap \mathsf{Valid}$ such that $\overline{\eta} \notin \mathsf{Effect} = \mathsf{Valid} \setminus \mathsf{NEffect}^*$, leading to $\overline{\eta} \in [\![\gamma\!\uparrow_x]\!] \cap \mathsf{NEffect}^*$ and hence **(FPC2)**. □

Similar to Lemma 3, which opened the door towards effective feature cause computations, we observe that precauses are exactly those prime implicants ∂ of $\Theta(F) \setminus \mathsf{NEffect}^*$ that cover an effect witness, i.e., $[\![\partial]\!] \cap \mathsf{Effect}^* \neq \varnothing$. This directly leads to an algorithm for computing precauses, provided in Algorithm 2.

Algorithm 2: Computation of feature precauses

```
input  : $\mathsf{Effect}^*, \mathsf{NEffect}^* \subseteq \Theta(F)$ with $\mathsf{Effect}^* \cap \mathsf{NEffect}^* = \varnothing$
output: $\mathsf{PCauses}(\mathsf{Effect}^*, \mathsf{NEffect}^*)$
1 $\mathcal{P} := \textsc{Compute-Primes}\big(\Theta(F) \backslash \mathsf{NEffect}^*\big)$
2 forall $\partial \in \mathcal{P}$ where $[\![\partial]\!] \cap \mathsf{Effect}^* = \varnothing$ do $\mathcal{P} := \mathcal{P} \backslash \{\partial\}$
3 return $\mathcal{P}$
```

Proposition 2. *Given effect configurations* $\mathsf{Effect}^* \subseteq \Theta(F)$ *and non-effect configurations* $\mathsf{NEffect}^* \subseteq \Theta(F)$, *Algorithm* 2 *computes* $\mathsf{PCauses}(\mathsf{Effect}^*, \mathsf{NEffect}^*)$, *the set of feature precauses for* Effect^* *w.r.t.* $\mathsf{NEffect}^*$, *in polynomial time in* $|\Theta(F)|$.

4. Causal explications

Since the number of feature causes (and precauses) can be exponential in the number of features, a mere listing of all causes is neither feasible nor expedient for real-world software systems. This holds for both, humans that have to evaluate causal relationships in configurable systems, e.g., during software development, and machines that might use feature causes for further processing and reasoning.

In this section, we present and discuss several methods to compute *causal explications*, i.e., mathematical or computational constructs that arise from processing feature causes to provide useful causal representations and measures (Baier et al., 2021). Explications are closely related to *explanations*, by which we mean human-understandable objects employed within an integrated system, e.g., in feature-oriented software development or in production-level deployments.

Our methods for computing explications rely on techniques from propositional logic and circuit optimization (Paul, 1975; McGeer et al., 1993), responsibility and blame (Chockler and Halpern, 2004), and feature interactions (Garvin and Cohen, 2011). They all take a global perspective on sets of feature causes rather than only considering single feature causes in isolation. In the following, we fix sets of valid configurations $\mathsf{Valid} \subseteq \Theta(F)$ and effects $\mathsf{Effect} \subseteq \mathsf{Valid}$.

4.1. Propositional logic formulas

A rather natural explication for a set of causes $C \subseteq \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$ is to represent C as propositional logic formula, e.g., as the *characteristic formula* $\chi(C)$ defined in disjunctive normal form (DNF)

$$\chi(C) = \bigvee_{\partial \in C} \Big(\bigwedge_{\substack{x \in \mathsf{supp}(\partial) \\ \partial(x) = \mathtt{true}}} x \wedge \bigwedge_{\substack{x \in \mathsf{supp}(\partial) \\ \partial(x) = \mathtt{false}}} \neg x \Big).$$

Clearly, $\chi(C)$ has the same size as C and its representation does not exhibit any advantage compared to C. Methods to minimize propositional logic formulas (McCluskey, 1956; McGeer et al., 1993; Hemaspaandra and Schnoor, 2011) could be used to yield small formulas φ covering the same configurations as C, i.e., where $[\![\varphi]\!] = [\![C]\!]$. While beneficial for related problems in configurable systems analysis, e.g., for presence condition simplification (von Rhein et al., 2015), such methods vanish causal information, i.e., the set of causes C cannot be reconstructed from the reduced formula φ. To provide a small formula that maintains the causal information of C, we use a simple yet effective reduction method, which we call *distributive law simplification (DLS)*. The basic idea is to factorize common feature selections in a DNF formula step by step, exploiting the n-ary distributive law $[\![\bigvee_{i=1}^{n}(\phi \wedge \psi_i)]\!] = [\![\phi \wedge \bigvee_{i=1}^{n} \psi_i]\!]$. Each factorization leads to a length reduction of $(n-1) \cdot |\phi|$, where $|\phi|$ is the length of the propositional formula factored out. Obviously, these transformations are reversible, such that the original DNF $\chi(C)$ and hence the set of causes C can be reconstructed. The final formula length depends on the formulas factored out, the subformulas, and the factorization order. Determining a formula through DLS that has minimal size is close to global optimization problems for propositional logic formulas and thus computationally hard. For practical applications, we hence employ a heuristics that reduces a given formula φ in DNF by stepwise factoring out literals that have maximal number of occurrences in DNF subformulas. We denote the reduced formula obtained by this heuristics by $\mathsf{DLS}(\varphi)$, where $\mathsf{DLS}(\cdot)$ is provided by Algorithm 3.

Algorithm 3: Distributive law simplification $\mathsf{DLS}(\cdot)$

```
input  : $\phi = \bigvee_{i \in I} \bigwedge_{j \in J_i} \ell_{ij}$ in DNF
output: propositional logic formula $\mathsf{DLS}(\phi)$
1 if $|I| \leq 1$ then return $\phi$
2 forall $i \in I$ do $L_i := \{\ell_{ij} : j \in J_i\}$
3 forall $\ell \in \bigcup_{i \in I} L_i$ do
      $N_\ell := |\ \{(i,j) \in I \times J_i \ :\ |J_i| > 1, \ell_{ij} = \ell\}\ |$
4 $\alpha = argmax_{\ell \in \bigcup_{i \in I} L_i} N_\ell$
5 $\phi_0 = \bigvee_{\substack{i \in I \\ |L_i| \leq 1 \vee \alpha \notin L_i}} \bigwedge_{j \in J_i} \ell_{ij}$
6 $\phi_1 = \bigvee_{\substack{i \in I \\ |L_i| > 1 \wedge \alpha \in L_i}} \bigwedge_{\substack{j \in J_i \\ \ell_{ij} \neq \alpha}} \ell_{ij}$
7 if $[\![\phi_0]\!] = \varnothing$ then return $\alpha \wedge \mathsf{DLS}(\phi_1)$
8 return $\mathsf{DLS}(\phi_0) \vee \big(\alpha \wedge \mathsf{DLS}(\phi_1)\big)$
```

Note that Line 4 involves a non-deterministic choice of a literal with maximal occurrence. This leaves some degree of freedom when implementing this algorithm, enabling heuristics depending on the practical need, e.g., prioritizing important or user-specific features for explication.

Extended boolean connectives. Besides the standard Boolean connectives $\wedge$ and $\vee$ apparent in our DLS, common patterns of propositional logic formulas such as $\oplus$ (XOR), $\leftrightarrow$ (equivalence), $\neq$ (non-equivalence), and $\rightarrow$ (implication) could be used for explication and providing a short (yet reversible) representation.

Example 5. In the feature diagram of Fig. 1, the encryption features are connected through an exclusive disjunction (XOR) and the set of valid feature configurations could then be described by

$$m \wedge (c \vee a \vee r \to e) \wedge (e \to c \oplus a \oplus r).$$

⋄

Such extended Boolean connectives could well be included into our DLS scheme, possibly also exploiting *anti-distributivity* of →.

Numeric features. Another possibility to further provide reversible and concise representations could be achieved by exploiting properties of multi-features and attributes, i.e., features that might not only be active or inactive but are configurable by a numerical value (Classen et al., 2011; Cordy et al., 2013b). Formally, a *numeric feature configuration* is a function $f : F \to \mathbb{D}$ over a finite domain $\mathbb{D} \subseteq \mathbb{N}$. It is well known that feature attributes can be modeled by Boolean features by extending the feature space, e.g., introducing features $\ell_0, \ldots, \ell_9$ in the case a feature attribute ℓ can take numeric values in $\mathbb{D} = \{0, \ldots, 9\}$. To this end, our Boolean framework for feature causality also covers reasoning about configurable systems with feature attributes. Towards a meaningful explication, we might however revert this encoding after our feature causality analysis and replace parts of a propositional logic formula for the feature causes by corresponding arithmetic expressions. For instance, $\ell_0 \vee \ell_1 \vee \ell_2$ could be replaced by an expression $\ell \leq 2$.

4.2. Cause–effect covers

The complete set of causes may contain several candidates to describe reasons for the effect emerging in a single system variant. If not interested in all causes but in a set of causes that covers all effects (i.e., that contains, at least, one cause for every system variant) we might ask for a preferably small set of causes covering all effect configurations. Formally, a *cause–effect cover* of Effect is a set of causes $C \subseteq$ Causes(Effect, Valid) that covers Effect, i.e., where Effect $\subseteq [\![C]\!]$. Note that Causes(Effect, Valid) is a cause–effect cover of Effect

Lemma 5. Effect $\subseteq [\![$Causes(Effect, Valid)$]\!]$.

Proof. Let us assume the opposite, i.e., there is a configuration $\eta \in$ Effect such there is no $\gamma \in$ Causes(Effect, Valid) with $\eta \in [\![\gamma]\!]$. Since **(FC1)** is fulfilled for η, **(FC2)** cannot be true for η since otherwise $\eta \in$ Causes(Effect, Valid) and thus, $\gamma = \eta$ could serve as counterexample of our statement. Hence, there is an $x \in F$ such that $[\![\eta\!\uparrow_x]\!] \cap$ Valid $\subseteq$ Effect, which yields **(FC1)** to hold for $\eta\!\uparrow_x$. By an inductive argument, there is $\varnothing \neq X \subseteq F$ such that $\eta\!\uparrow_X \in$ Causes(Effect, Valid) but $\eta \in [\![\eta\!\uparrow_X]\!]$. □

We say that a cause–effect cover C is *minimal* if there is no cause–effect cover C' of Effect where $|C'| < |C|$. Note that this notion of minimality is similar to the standard definition for sets of partial interpretations, but ranges not over all sets of partial configurations but over causes only.

Example 6. For the email system in Example 2 we directly see that $\{\gamma_{e\bar{c}}\}$ and $\{\gamma_a, \gamma_r\}$ are the only cause–effect covers of Effect. Thus, $\{\gamma_{e\bar{c}}\}$ is a minimal cause–effect cover of Effect as $|\{\gamma_a, \gamma_r\}| > |\{\gamma_{e\bar{c}}\}|$.⋄

It is well known that computing minimal covers is expensive as the decision problem whether there is a cover of a given set of configurations with at most $k \in \mathbb{N}$ elements is NP-complete (e.g. Umans et al., 2006). The same holds for computing minimal prime $*$-covers and hence minimal cause–effect covers (Paul, 1975). Thus, for practical applicability, heuristics that lead to nearly minimal cause–effect covers are of interest to concisely explicate causal candidates covering all effect configurations. In the following, we establish such a heuristic by a greedy scheme involving *most general causes*.

Definition 4. The binary relation $\trianglelefteq \subseteq \Delta(F) \times \Delta(F)$, where $\partial \trianglelefteq \partial'$ stands for ∂' to be *at least as general as* ∂ w.r.t. Effect, is defined by

$$\partial \trianglelefteq \partial' \quad \text{iff} \quad [\![\partial]\!] \cap \text{Effect} \subseteq [\![\partial']\!] \cap \text{Effect}.$$

The set of *most general causes* MaxCauses(Effect, Valid) comprises those causes for Effect w.r.t. Valid that are $\trianglelefteq$-maximal in Causes(Effect, Valid).

In the next paragraphs, we formally prove relationships of Causes(Effect, Valid) and the set of most general causes MaxCauses(Effect, Valid) to prime $*$-covers and minimal prime $*$-covers. From the above definition, we then directly establish an algorithm to compute MaxCauses(Effect, Valid) by computing $\trianglelefteq$ in quadratic time and by selecting elements maximal w.r.t. $\trianglelefteq$. In particular, we will see that Causes(Effect, Valid) and MaxCauses(Effect, Valid) both provide cause–effect covers of Effect. Note that $\trianglelefteq$ is not antisymmetric and hence there might be different most general causes that cover the same set of effect instances. Towards nearly minimal cause–effect covers, we thus might pick only one of those candidates from MaxCauses(Effect, Valid) (e.g., with minimal support) to obtain even more concise representatives for feature causality.

Feature causality and prime covers. Lemma 5 and the section above showed the close connection of feature causes of Effect w.r.t. Valid to prime covers. However, it is still open whether Causes(Effect, Valid) is actually a prime $*$-cover of Effect relative to Valid∖Effect. In the corner cases, e.g., if Effect $= \varnothing$, then Causes(Effect, Valid) $= \varnothing$ due to **(FC1)** and the statement holds since $\varnothing$ is the trivial $*$-cover of $\varnothing$ relative to any set. Further observe that if Effect = Valid, the statement is also clear since the uniquely defined prime $*$-cover of Effect relative to Valid∖Effect is $\{\alpha\}$ with $\alpha \in \Delta(F)$ where supp$(\alpha) = \varnothing$. Then, **(FC2)** is trivially fulfilled and since α covers all configurations, also **(FC1)** holds.

For the following proposition, recall that for sets of partial configurations $\mathcal{P}_0$ and $\mathcal{P}_1$, $\mathbb{P}(\mathcal{P}_1, \mathcal{P}_0)$ denotes the set of *prime $*$-covers* of $\mathcal{P}_1$ relative to $\mathcal{P}_0$ and $m\mathbb{P}(\mathcal{P}_1, \mathcal{P}_0)$ denotes the set of *minimal prime $*$-covers* of $\mathcal{P}_1$ relative to $\mathcal{P}_0$.

Proposition 3.

(P1) Causes(Effect, Valid) $\in \mathbb{P}$(Effect, Valid∖Effect)
(P2) $\mathcal{M} \subseteq$ Causes(Effect, Valid) *for all* $\mathcal{M} \in m\mathbb{P}$(Effect, Valid∖Effect)

Proof. Ad **(P1)**: Due to Definition 1 and Lemma 3, all causes are prime implicants of $\big(\Theta(F)\setminus$Valid$\big) \cup$ Effect. Further, $\big(\Theta(F)\setminus$Valid$\big) \cup$ Effect $= \Theta(F)\setminus\big($Valid∖Effect$\big)$ and thus, it is left to show that Causes(Effect, Valid) covers Effect. Assume the opposite, i.e., there is $\eta \in$ Effect such that $\eta \notin [\![$Causes(Effect, Valid)$]\!]$. Then there must be a prime implicant $\partial \in \Delta(F)$ of $\big(\Theta(F)\setminus$Valid$\big) \cup$ Effect with $\eta \in [\![\partial]\!]$. By Lemma 3, and $\eta \in [\![\partial]\!] \cap$ Effect, **(FC1)** is satisfied. Further, ∂ being a prime implicant directly leads to $[\![\partial\!\uparrow_x]\!] \cap$ (Valid∖Effect) $\neq \varnothing$ for all $x \in$ supp(∂) and thus, **(FC2)** and $\partial \in$ Causes(Effect, Valid). This contradiction yields that Causes(Effect, Valid) is a prime $*$-cover of Effect relative to Valid∖Effect.

Ad **(P2)**: First consider the case where $\mathcal{M} = \varnothing$, then this is the only minimal prime $*$-cover and Effect $= \varnothing$, leading to Causes(Effect, Valid) $= \varnothing$ and $\mathcal{M} \subseteq$ Causes(Effect, Valid). Now let $\partial \in \mathcal{M}$, then $[\![\partial]\!] \cap$(Valid∖Effect) $= \varnothing$ since ∂ is a prime implicant of $\big(\Theta(F)\setminus$Valid$\big) \cup$ Effect $= \Theta(F)\setminus\big($Valid∖Effect$\big)$, which also leads to **(FC2)** being fulfilled for ∂. Further, since $\mathcal{M}$ is a minimal cover, $\mathcal{M}\setminus\{\partial\}$ is not a cover of Effect and thus, $[\![\partial]\!] \cap$ Effect $\neq \varnothing$. In combination with $[\![\partial]\!] \cap$ (Valid∖Effect) $= \varnothing$, we obtain $[\![\partial]\!] \cap$ Valid $\subseteq$ Effect and thus, **(FC1)** is fulfilled for ∂. Hence, $\partial \in$ Causes(Effect, Valid). □

Prime covers and most general causes. Due to the close connection between feature causes and prime implicants (see Proposition 3 and Lemma 3) and the global maximization definition w.r.t. the semantics of partial feature configurations, one might think about a connection of minimal prime $*$-covers of Effect relative to Valid∖Effect and MaxCauses(Effect, Valid). However, there could be most general feature causes that are not contained in any minimal $*$-cover Effect relative to Valid∖Effect:

Example 7. Let $F = \{a, b, c\}$, Effect $= \{abc, ab, ac, c\}$, and Valid $= \Theta(F)$. Then there are three feature causes Causes(Effect, Valid) $= \{\gamma_{ab}, \gamma_{ac}, \gamma_{\bar{b}c}\}$ with γ_{ab} assigning `true` to a and b, γ_{ac} assigning `true` to a and c, and $\gamma_{\bar{b}c}$ assigning `false` to b and `true` to c. Note that all these causes are most general according to Definition 4. However, $[\![\gamma_{ac}]\!] \subseteq [\![\gamma_{ab}]\!] \cup [\![\gamma_{\bar{b}c}]\!]$, i.e., γ_{ac} is covered by γ_{ab} and $\gamma_{\bar{b}c}$. Hence,

$$\mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid}) \not\subseteq \bigcup_{\mathcal{M} \in m\mathbb{P}(\mathsf{Effect}, \mathsf{Valid} \setminus \mathsf{Effect})} \mathcal{M}$$

Nevertheless, all most general feature causes are contained in any minimal $*$-cover of Effect relative to $\mathsf{Valid} \setminus \mathsf{Effect}$.

Proposition 4.

(M1) $\mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid}) \in \mathbb{P}(\mathsf{Effect}, \mathsf{Valid} \setminus \mathsf{Effect})$

(M2) *There is* $\mathcal{M} \in m\mathbb{P}(\mathsf{Effect}, \mathsf{Valid} \setminus \mathsf{Effect})$ *with* $\mathcal{M} \subseteq \mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid})$.

Proof. Ad **(M1)**: Following Definition 4, we have that $\mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid}) \subseteq \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$ but $[\![\mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid})]\!] = [\![\mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})]\!]$.

The statement is then a direct consequence of the definition of covers, **(P1)** of Proposition 3, and $\mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid}) \subseteq \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$.

Ad **(M2)**: Due to Proposition 3**(P2)** we have $\mathcal{M}' \subseteq \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$ for all $\mathcal{M}' \in m\mathbb{P}(\mathsf{Effect}, \mathsf{Valid} \setminus \mathsf{Effect})$. Let $\mathcal{M}$ arise from $\mathcal{M}'$ by replacing every $\partial' \in \mathcal{M}'$ with $\partial' \notin \mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid})$ by a $\partial \in \mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid})$ such that $\partial' \trianglelefteq \partial$. Then $\mathcal{M} \in m\mathbb{P}$ $(\mathsf{Effect}, \mathsf{Valid} \setminus \mathsf{Effect})$ and $\mathcal{M} \subseteq \mathsf{MaxCauses}(\mathsf{Effect}, \mathsf{Valid})$. □

To this end, sets of most general causes can serve as prime $*$-covers of Effect relative to $\mathsf{Valid} \setminus \mathsf{Effect}$ and thus provide cause–effect covers as concise representation of cause candidates for each effect instance.

Effect uncertainty and precause–effect covers. Following Definition 3 and Proposition 2, counterfactual reasoning by using prime implicant computations can be also used to provide causal candidates for underspecified effect sets. However, the number of precauses is usually much higher than actual causes, due to the uncertainty of configurations being possible to serve as effect or non-effect instances. To this end, precause–effect covers are important to reduce the number of precauses eligible to describe the (sure) effect instances and respect counterfactual reasoning. It seems reasonable to aim for precause–effect covers that minimize the number of precauses and the covered underspecified valid configurations (or configurations in case the set of valid configurations is unknown). We propose to first compute most general precauses following Definition 4 and stepwise select those towards a precause–effect cover that have the highest ratio between covered effect instances from Effect^* and *additional* covered underspecified valid configurations. For the latter, we assume configurations covered by previously selected most-general precauses as not underspecified anymore (with complete information, they would turn into effect instances for feature causes following **(FC1)** in Definition 1).

4.3. Responsibility and blame

To measure the influence of causes on effects, Chockler and Halpern (2004) introduced degrees of *responsibility* and *blame*, ranging from zero to one for “no” to “full” responsibility and blame, respectively. *Responsibility* measures how relevant a single cause is for an effect in a specific context. *Blame* denotes the expected overall responsibility according to a given probability distribution on all contexts where the effect emerges. We take inspiration from these measures and present corresponding notions for feature causality. In short, the degree of responsibility is the maximal share of features contributing to the effect, i.e., features that would have to be reconfigured to provide a counterfactual witness.

Example 8. We rephrase the *majority example* by Chockler and Halpern (2004) in our setting. Consider 11 features whose configurations are all valid. We are interested in responsibilities for the effect that the majority of features is active. If all eleven features are active, each feature has a responsibility of $1/6$, since six features share the responsibility for the effect: Besides the feature of interest, further five features have to be reconfigured towards a majority of inactive features. In a configuration where six features are active and five not, each of the six active ones is fully responsible for the effect: If this feature would be reconfigured, more features would be inactive than active. We then assign responsibility of one to each of the six active features.⋄

In what follows, we formalize degrees of responsibility and blame for single features as in the example above. An extension of this notion to partial configurations to explicate feature interactions is provided in Section 4.4.

Feature responsibility. Intuitively, the degree of responsibility of a single feature $x \in F$ is defined as the maximal share to contribute to causing the effect in an effect instance $\eta \in \mathsf{Effect}$. In the case that x does not appear in the support of any cause γ covering η, feature x does not contribute to causing the effect in η and thus has no responsibility. Otherwise, x shares its responsibility with, at least, a minimal number of other features, whose switch of its interpretation in η would lead to a counterfactual witness $\overline{\eta}$, i.e., $\overline{\eta} \in \mathsf{Valid} \setminus \mathsf{Effect}$. In the following, Y denotes the set of features that have to be switched in a configuration (including the feature x the responsibility is determined for) such that a counterfactual witness is reached.

Definition 5. The *degree of responsibility* of a feature $x \in F$ in the context $\eta \in \mathsf{Effect}$ for which there is $\gamma \in \mathsf{Causes}(\mathsf{Effect}, \mathsf{Valid})$ with $\eta \in [\![\gamma]\!]$ and $x \in \mathsf{supp}(\gamma)$ is defined as

$$\mathsf{resp}(x, \eta) = \frac{1}{\min\{|Y| : Y \subseteq F, x \in Y, \mathsf{switch}(Y, \eta) \in \mathsf{Valid} \setminus \mathsf{Effect}\}}$$

and as $\mathsf{resp}(x, \eta) = 0$ otherwise.

Note that due to **(FC2)** there exists at least one counterfactual witness in the case x appears in a cause covering η and hence, the denominator of the above fraction is finite and greater than zero.

Example 9. Continuing the email example (see Example 2), the mail feature m and sign feature s do not have any responsibility for a long decipher time in any configuration as they do not appear in any of the causes γ_a, γ_r, and $\gamma_{e\bar{c}}$. Also the AES feature a has no responsibility in configurations mer or $mers$, since the only covering causes are $\gamma_{e\bar{c}}$ and γ_r, not containing a in their support. Besides the analogous case for the RSA feature r, other degrees of responsibility are $1/2$: switching a feature x in η usually requires one further feature to switch towards a configuration of $\mathsf{Valid} \setminus \mathsf{Effect}$. For example, selecting the Caesar feature c in $\eta = mea$ requires also to deselect the AES feature a, leading to $\overline{\eta} = mec \in \mathsf{Valid} \setminus \mathsf{Effect}$. The table below shows $\mathsf{resp}(x, \eta)$ for $x \in \{e, c, a, r\}$.

η	$x = e$	$x = c$	$x = a$	$x = r$
mea	1/2	1/2	1/2	0
mer	1/2	1/2	0	1/2
$msea$	1/2	1/2	1/2	0
$mser$	1/2	1/2	0	1/2

Example 10. Let us take a more formal view on Example 8, using notations to specify effect sets (see Section 3.1). Let $F = \{x_0, \ldots, x_{10}\}$ define the set of features and consider all configurations valid, i.e., $\mathsf{Valid} = \Theta(F)$. Further, let $\rho : \mathsf{Valid} \to \mathbb{R}$ count the number of active features in a configuration. The majority effect set is then specified through $\mathsf{Effect}_{\rho>5} = \{\theta \in \mathsf{Valid} : \rho(\theta) > 5\}$. In any effect configuration $\eta \in \mathsf{Effect}_{\rho>5}$, the responsibility $\mathsf{resp}(x, \eta)$ of any feature $x \in F$ is zero if $\eta(x) = \mathtt{false}$ and $\frac{1}{\rho(\eta)-5}$ otherwise. For instance, if $\rho(\eta) = 11$, then all

features are selected and thus, each of the features have responsibility of $1/6$ since in addition to the feature, at least five other features have to swap its vote before not winning the majority vote. In case $\rho(\eta) = 6$, the responsibility of each of the features x where $\eta(x) = \texttt{true}$ is one, since only swapping the vote of x suffices to turn down the majority vote.⋄

Blame. Degrees of responsibility are locally defined w.r.t. a context, while one is surely also interested in a global measure of responsibility of a feature or partial configuration w.r.t. all possible contexts, i.e., effect configurations. The *degree of blame* is defined as the expected degree of responsibility on a probability distribution $\pi : \mathsf{Valid} \to [0,1]$ over valid configurations Valid, i.e., where $\sum_{\theta\in\mathsf{Valid}} \pi(\theta) = 1$.

Definition 6. The *degree of blame* of a feature $x \in F$ w.r.t. a distribution π over Valid is defined as

$$\mathsf{blame}(x,\pi) \;=\; \sum_{\eta\in\mathsf{Effect}} \pi(\eta)\cdot \mathsf{resp}(x,\eta).$$

Example 11. To illustrate the degree of blame, continue Example 9, where we assume a uniform distribution π over effect configurations, i.e., $\pi(\eta) = 1/|\mathsf{Effect}| = 1/4$ for all $\eta \in \mathsf{Effect}$ and 0 otherwise. Then, $\mathsf{blame}(x,\pi) = 1/4 \sum_{\eta\in\mathsf{Effect}} \mathsf{resp}(x,\eta)$ and thus, $\mathsf{blame}(x,\pi)$ is $1/2$ for $x \in \{e,c\}$, $3/8$ for $x \in \{a,r\}$, and 0 for $x \in \{m,s\}$. Note that responsibility and blame values are independent from the polarities of feature, i.e., whether they have to be active or inactive for being causally relevant. For instance, features e and c have both the highest blame values, originating from the cause $\gamma_{e\bar{c}}$ where e is active and c is inactive. ⋄

On the choice of blame distributions. The distribution π models the frequency of valid configurations to occur, for which there are several scenarios that lead to a reasonable definition of π. One natural distribution may model the frequency of users choosing a configuration in the configurable system. But also the frequency of effect configurations is useful, e.g., to model the frequency a certain bug is reported by users when the effect corresponds to the property of a malfunction. In the case such statistics are not at hand or one is interested in the degree of blame from a developer's perspective, uniform distributions over valid configurations or effects are canonical candidates for π.

4.4. Feature interactions

A notorious problem in configurable systems is the presence of (inadvertent) *feature interactions* (Calder et al., 2003; Apel et al., 2014), which describe system behaviors that emerge due to a combination of multiple features not easily deducible from the features' individual behaviors. The detection, isolation, and resolution of feature interactions play a central role in the development of configurable systems and beyond (Zave, 2001; Apel et al., 2013). On the one hand, feature interactions may be desired to integrate and coordination multiple independently developed features. On the other hand, due to the exponential number of possibilities of how features may interact, undesired feature interactions can easily be overseen by developers. The problem of detecting unintended feature interactions rises in severity with the number of features. This led to a crisis in the area of telecommunication systems already in the early 1980s. Back then, an increasing number of undesired behaviors between features of complex telecommunication systems have been observed (Calder et al., 2003). A multitude of approaches have been proposed and are still developed to address the problem of detecting (undesired) feature interactions, for example, to discover feature interaction faults (Garvin and Cohen, 2011). We now show how our black-box causal analysis at the level of features (see Section 3.1) can be used for the detection and isolation of feature interactions. These can provide the basis for fine-grained, white-box feature-interaction resolution as also proposed by Garvin and Cohen (2011).

Detection. The first problem we address is to detect the *necessity* of feature interactions for an effect to emerge. Garvin and Cohen presented a formal definition of *feature interaction faults* to capture faults in configurable systems that necessarily arise from the interplay between multiple features (Garvin and Cohen, 2011). Notably, their characterization is also at the abstraction level of features and relies on black-box testing of faults, similar to our perspective on effects. We transfer their definition to our setting, but covering arbitrary effects instead of faults only. Recall that a partial configuration $\omega \in \Delta(F)$ is *sufficient* for Effect w.r.t. Valid if $\varnothing \neq [\![\omega]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect}$ (see **(FC1)**).

Definition 7. A partial configuration $\omega \in \Delta(F)$ is a *t-way interaction witness* for Effect w.r.t. Valid if

(FI1) ω sufficient for Effect w.r.t. Valid with $|\mathsf{supp}(\omega)| = t$ and
(FI2) there is no $\hat{\omega}$ sufficient for Effect w.r.t. Valid with $|\mathsf{supp}(\hat{\omega})| = t-1$.

Basically, there is a one-to-one correspondence between interaction witnesses and feature causes with minimal support:

Theorem 2. *For any partial configuration $\gamma \in \Delta(F)$ with $t = |\mathsf{supp}(\gamma)|$ we have that*

(1) if γ is a t-way interaction witness for Effect *w.r.t.* Valid*, then $\gamma \in \mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$, and*

(2) if $\gamma \in \mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$ and there is no $\hat{\gamma} \in \mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$ with $|\mathsf{supp}(\hat{\gamma})|<t$, then γ is a t-way interaction witness for Effect *w.r.t.* Valid.

Proof. Ad (1): Let γ be a t-way interaction witness for Effect w.r.t. Valid. Since $\varnothing \neq [\![\gamma]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect}$, also $[\![\gamma]\!] \cap \mathsf{Effect} \neq \varnothing$, directly leading to **(FC1)**. Further, $[\![\gamma]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) = \varnothing$ due to $[\![\gamma]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect}$ and $[\![\gamma{\uparrow_x}]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) \neq \varnothing$ for all $x \in \mathsf{supp}(\gamma)$ since otherwise $\gamma' = \gamma{\uparrow_x}$ would be a $t-1$-way interaction witness for $t > 1$ or $[\![\gamma{\uparrow_x}]\!] = \Theta(F)$ for $t = 1$ (recall that $\mathsf{Valid}\backslash\mathsf{Effect} = \mathsf{Valid} \setminus \mathsf{Effect}$). Hence, also **(FC2)** is fulfilled for γ.

Ad (2): Let $\gamma \in \mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$ where there is no $\hat{\gamma} \in \mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$ with $|\mathsf{supp}(\hat{\gamma})| < |\mathsf{supp}(\gamma)|$. Then $[\![\gamma]\!] \cap \mathsf{Effect} \neq \varnothing$ due to **(FC1)** and $[\![\gamma]\!] \cap (\mathsf{Valid}\backslash\mathsf{Effect}) = \varnothing$ due to **(FC2)**. Thus, $[\![\gamma]\!] \cap \mathsf{Valid} \subseteq \mathsf{Effect}$ and thus, **(FI1)** holds. Note that this holds for all causes, independent from the minimality condition. Thus, since there is no $\hat{\gamma} \in \mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$ with $|\mathsf{supp}(\hat{\gamma})| < |\mathsf{supp}(\gamma)|$, there is no $\hat{\gamma} \in \mathsf{Causes}(\mathsf{Effect},\mathsf{Valid})$ with $|\mathsf{supp}(\hat{\gamma})| = |\mathsf{supp}(\gamma)| - 1$ and hence, also **(FI2)** is fulfilled. □

To this end, Algorithm 1, in combination with a projection to feature causes with minimal support, can be used to decide whether the effect emerges necessarily from feature interactions: A necessary feature interaction takes place in the case these minimal feature causes all have a supports that involve, at least, two features.

Example 12. Returning to Example 2, there are exactly two causes that are both 1-way interaction witnesses: γ_a and γ_r. The cause $\gamma_{e\bar{c}}$ does not witness a necessary 2-way feature interaction since although having support size of two, γ_a and γ_r have support size one (cf. Theorem 2(2)). Hence, the effect describing long decipher time is not necessarily related to a feature interaction.⋄

Isolation. The second problem we address is to pinpoint features responsible for feature interactions. For this, observe that Definition 7 is similar to Definition 1, but with a different notion of minimality: While **(FI2)** ensures global minimality over all partial configurations, **(FC2)** ensures local minimality through expansions, taking the individual selection of features into account. To pinpoint those features that actually interact towards the effect, feature causes can be interpreted as a form of *interaction witnesses* at the local level instead of the global perspective taken for t-way interaction witnesses: Switching some feature a cause does not ensure the effect to emerge anymore — hence, the switched feature is necessary for the effect. For instance, in Example 12, the interaction between encryption and Caesar being disabled is witnessed by the feature cause $\gamma_{e\bar{c}}$. Feature causes thus also provide a criterion for feature interactions at the operational level and can be used to guide a more in-depth white-box feature interaction analysis, possibly reducing the naive exponentially-sized feature-interaction search space.

Feature interaction responsibility and blame. Any subset of the support of a feature cause that contains more than two features provides a candidate for an actual interaction between those features. Since both, the number of feature causes and their expansions, can be exponential in the number of features, feature interactions isolated via a causal analysis might still be difficult to interpret by developers. Based on the degree of responsibility for single features, we now provide a variant to measure responsibility and blame of feature interactions, where high values indicate strong relevance of the interaction and low values weak relevance. Both measures are defined as the share of features $Y \subseteq F$ to be switched including at least one feature $x \in \mathsf{supp}(\partial) \cap Y$ from the interaction support to obtain a counterfactual witness. This definition covers the single feature case (Definition 5), where the latter feature x coincides with the feature of interest.

Definition 8. The *degree of responsibility* $\mathsf{resp}(\partial, \eta)$ of a partial configuration $\partial \in \Delta(F)$ in the context $\eta \in \mathtt{Effect}$ for which there is $\gamma \in \mathsf{Causes}(\mathtt{Effect}, \mathsf{Valid})$ with $\eta \in [\![\gamma]\!]$ and $\mathsf{supp}(\partial) \subseteq \mathsf{supp}(\gamma)$ and $\partial(x) = \gamma(x)$ for all $x \in \mathsf{supp}(\partial)$ is defined as

$$\frac{1}{\min\{|Y| \,:\, Y \subseteq F, \mathsf{supp}(\partial) \cap Y \neq \emptyset, \mathsf{switch}(Y, \eta) \in \mathsf{Valid}\backslash\mathtt{Effect}\}}$$

and as $\mathsf{resp}(\partial, \eta) = 0$ otherwise.

Note that, for $\partial(x) = \eta(x)$ and $\mathsf{supp}(\partial) = \{x\}$, Definition 8 agrees with Definition 5. The degree of responsibility is non-zero in the case of a single feature if the feature appears in some cause, whereas the degree of feature interaction responsibility is non-zero if some cause is an expansion of the potential feature interaction. *Feature interaction blame* is extended similarly from the single feature case (Definition 6) by replacing single features $x \in F$ by partial configurations $\partial \in \Delta(F)$ that stand for the potential feature interaction of interest:

Definition 9. The *degree of blame* of a partial configuration $\partial \in \Delta(F)$ w.r.t. a distribution π over Valid is defined as

$$\mathsf{blame}(\partial, \pi) = \sum_{\eta \in \mathtt{Effect}} \pi(\eta) \cdot \mathsf{resp}(\partial, \eta).$$

Example 13. We formalize the majority voting example of Example 8 by features $F = \{x_1, \dots, x_{11}\}$ and a function $\rho \colon \mathsf{Valid} \to \mathbb{N}$ counting active features in valid configurations such that $\mathtt{Effect} = \{\theta \in \mathsf{Valid} : \rho(\theta) > 5\}$. Then, the degree of responsibility of a part $X = \{x_1, x_2, x_3\}$ in the context of $\eta \in \mathtt{Effect}$ where $\rho(\eta) = 11$, i.e., all features are active. Then, the joint responsibility of X in η is $1/4$, since besides the features of X, further three features have to swap to yield a non-winning configuration. ⋄

5. Experiment setup

To evaluate our causal analysis and explication methods, we conducted a number of experiments comprising many analyses on community benchmarks and real-world examples from the area of configurable software systems.

Our evaluation is driven by the five research questions stated in the introduction that address the key issue of whether and how the notion of feature causality facilitates identifying root causes, estimating the effects of features, and detecting feature interactions in controlled and practical settings.

5.1. Implementation

We implemented our algorithms to compute feature causes and explications in the prototypical tool FeatCause. Written in Python, our tool relies on the engines for logical expressions and binary decision diagrams (BDDs) of PyEDA, a library for electronic design automation (Drake, 2015). The tool takes the sets of valid feature configurations Valid and effects Effect as input. FeatCause supports different input formats for these sets, e.g., by Boolean expressions in DNF or CNF. We implemented Algorithm 1, which uses prime implicants to efficiently determine feature causes (cf. Section 3). Internally, sets of (partial) feature configurations are efficiently represented as reduced ordered BDDs (Bryant, 1986). In addition to their compact and hence space-efficient representation, we chose BDDs because they provide an efficient method to check satisfiability (required, e.g., for Line 2 in Algorithm 1). Note that even when using BDDs, a naive algorithm that directly checks the conditions **(FC1)** and **(FC2)** for all partial configurations is not feasible, since it would need to construct and operate on exponentially many BDDs for each of the possible partial configurations. Instead, in our evaluation we determine feature causes using prime implicant computations (cf. Section 3). To compute prime implicants, we used the tool Espresso (McGeer et al., 1993), well known from circuit optimization, through an interface that mediates between our BDD representations and the DNF representations in Espresso's PLA format. This interface is also used to provide minimal and nearly minimal cause–effect covers through Espresso-Signature and Espresso, respectively, which can then be compared with our heuristic cause–effect cover by most general causes (see Section 4.2). While it is well known that the length of DNFs can be exponential in the size of the BDD representing the same Boolean function, generating DNFs from our BDD representations did not face any significant blowup and required negligible time in all our experiments. For resolving the non-deterministic choice in Line 4 of Algorithm 3 (DLS) in our implementation, we picked the first element of the list of literals ordered according to their occurrences, i.e., left the resolution of the non-determinism open to be resolved by the sorting algorithm implemented in Python. We also modified the algorithm towards a global minimization by exhaustively iterating over all literals and returning the factorization with minimal length. Our conducted experiments with a global minimization resolution of this non-determinism unsurprisingly led to massive performance drawbacks. The impact on the resulting formula sizes turned out to be negligible. The reduction by the algorithm is reversible and hence the set of causes can be easily reconstructed from the reduced formula. Besides the core tool, we have implemented several conversion scripts to generate valid feature configuration sets from TVL (Classen et al., 2011) and effect sets from analysis results returned by variability-aware analysis tools such as ProVeLines (Cordy et al., 2013a) and ProFeat (Chrszon et al., 2018) (see also Section 3.1) and the data sets from Siegmund et al. (2015) and Kaltenecker et al. (2019).

5.2. Subject systems

We selected a diverse set of subject systems to approach our research questions, ranging from popular community benchmarks to more involved systems with non-functional properties and from real-world settings.

From Cordy et al. (2013a), we use CFDP, Elevator, and Minepump systems and analyzed them against the accompanied LTL properties φ using the variability-aware model checker ProVeLines. Furthermore, we took the Email and Elevator from von Rhein et al. (2015) and analyzed multiple defects provided as propositional logic formula generated by SPLVerifier (Apel et al., 2013).

For quantitative properties, we generated effect sets from configurable system analysis results as illustrated in Section 3.1 for three classes of systems.

Table 1
Statistics of feature causality experiments.

Property	System	#	|Valid|	$\|F\|$	Time	Average size of			
[type]					[s]	Effect	C	$\trianglelefteq$–C	DLS
φ	CFDP	10	56	13	0.21	28	7.30	2.60	58%
	ELEVATOR $_1$	36	256	9	0.09	128	0.58	0.58	26%
	MINEPUMP	82	128	11	0.29	64	1.77	0.91	43%
ε_B	LINKEDLIST	42	204	19	18.97	102	47.40	8.36	54%
	LINUX	21	16 777 216	25	5.88	22	22.14	22.14	80%
	PKJAB	28	72	12	0.42	36	8.43	4.86	72%
	PREVAYLAR	42	24	7	0.20	12	3.57	2.17	93%
	SNW	42	3 240	36	2 751.32	1 620	2 110.31	18.90	37%
	ZIPME $_1$	42	64	9	0.19	32	4.19	2.69	93%
ε_M	CURL	28	768	14	83.07	384	96.75	67.39	59%
	H264	28	1 152	17	355.37	576	229.86	113.93	53%
	SQLITE	21	3 932 160	40	34 826.86	1 310 729	1 881.67	436.48	27%
	WGET	28	5 120	17	777.26	2 560	298.11	201.79	51%
ε_T	APACHE	28	192	10	2.85	96	28.79	22.96	69%
	ELEVATOR $_2$	28	10	6	0.14	5	3.21	2.25	80%
	EMAIL	28	40	10	1.01	20	22.32	8.21	69%
	H264	56	1 152	17	32.24	576	68.98	22.71	51%
	ZIPME $_2$	42	640	16	1.73	320	11.07	9.98	72%
τ_R	BSN	4	298	11	0.63	149	21.00	8.75	68%
	VCL	22	2 097 152	21	60 403.78	1 048 576	3 718.05	3 718.05	36%
τ_T	BERKELEYDB	36	2 560	19	5.71	1 280	13.03	4.28	72%
	DUNE	46	2 304	32	3 950.15	1 152	724.87	46.00	46%
	LLVM	30	1 024	12	16.21	512	53.57	53.57	55%
	LRZIP	48	432	20	5.71	216	13.85	1.90	69%
	X264	48	1 152	17	2.38	576	7.50	3.75	78%

We list for each type of effect property the considered subject systems, the numbers of experiments (#), valid configurations (|Valid|), features ($|F|$), and the overall time in seconds to compute feature causes and most general causes. The second part of the table lists the average sizes of the effect sets (Effect), feature causes (C = Causes(Effect, Valid)), cause–effect covers by most general causes ($\trianglelefteq$–C = MaxCauses(Effect, Valid)), and DLS formulas relative to the characteristic formula of causes (DLS=|DLS($\chi(C)$)|/|$\chi(C)$| in percent).

First, we constructed effect sets for several systems from studies on performance prediction of non-functional properties (Siegmund et al., 2013a; Siegmund et al., 2012; Siegmund et al., 2013b), such as APACHE, LINUX, SQLITE, and WGET. For these, we used the black-box approach by Siegmund et al. (2012), which uses multivariable linear regression methods to generate variability-aware performance models. Our thresholds ε_B, ε_M, and ε_T for constructing effect sets are imposed on the prediction accuracy of the three different non-functional properties of binary size, memory footprint, and runtime, respectively.

Second, we considered configurable systems modeled for the variability-aware probabilistic model checker PROFEAT (Chrszon et al., 2018), comprising a body sensor network (BSN) model (Rodrigues et al., 2015) and a velocity control loop (VCL) model of an aircraft (Dubslaff et al., 2019a, 2020b). In both systems, the reliability of the system is analyzed in terms of the probability of failure of sensors and control components, respectively. Effect sets are generated by imposing a threshold τ_R on the analysis results.

Third, we generated effect sets from performance measurements of real-world configurable software systems that have been used to evaluate performance modeling techniques (Siegmund et al., 2015; Kaltenecker et al., 2019). In particular, we selected five systems from different domains: a compiler framework (LLVM), a database-system (BERKELEYDB), a compression tool (LRZIP), a video encoder (X264), and a toolbox for solving partial differential equations (DUNE). For these systems, we chose thresholds τ_T on the runtime of the system executions.

5.3. Operationalization

To address **(RQ1)**, we compute both sets Causes(Effect, Valid) and their dual Causes(Valid\Effect, Valid), i.e., the feature causes of the effect property and its negation. Based on these feature causes, we further compute the following causal explications, e.g., to answer **(RQ2)**:

- most-general cause–effect covers MaxCauses(Effect, Valid) and their dual set MaxCauses(Valid\Effect, Valid),
- nearly minimal cause–effect covers by $*$-covers of Effect w.r.t. Valid\Effect provided by ESPRESSO (and their dual case),
- distributive law simplification (DLS) on causes Causes(Effect, Valid) and cause–effect covers MaxCauses(Effect, Valid) (and their dual case),
- feature precauses PCauses(Effect*, NEffect*) and precause–effect covers for samples Effect* and NEffect* from Effect and Valid\Effect, respectively, of increasing size,
- responsibility and blame values for single features and causes, and
- feature interaction blames for pairs of features.

For blame computations, we assume a uniform distribution over all effects, due to the absence of further statistical information and taking a developers' perspective (cf. Section 4.3). For **(RQ3)**, we compute single feature blames based on a uniform effect distribution to measure the influence of individual features onto the effect. We compute feature interaction blames on pairs of features to address **(RQ4)**, again assuming a uniform effect distribution. Table 1 provides key statistics about our experiments, focusing on model characteristics and the time to compute feature causes and most general feature causes. All experiments were conducted on an AMD Ryzen 7 3800X 8-Core system with 32 GB of RAM running Debian 10 and Python 3.7.3.

6. Results

We discuss our results w.r.t. the different kinds of causal explications of Section 4. First, we discuss statistics on our experiments, also quantitatively analyzing the potential of causal explications by most general causes and DLS-reduced formulas. Then, we address our research questions in more depth by means of three representative subject systems. Here, we focus on properties not detectable by classical causal white-box analysis methods (Zeller, 2002; Johnson et al., 2020).

Table 2
Minepump – (most general) feature causes.

Blame	Feature cause (characteristic formula)
0.29	High ∧ Command ∧ ¬Stop ∧ MethaneAlarm
0.57	High ∧ Start ∧ MethaneAlarm
0.14	High ∧ Command ∧ ¬Stop ∧ MethaneSensor ∧ ¬MethaneQuery
0.57	High ∧ Start ∧ Stop
0.57	High ∧ Low ∧ Start
0.29	High ∧ Start ∧ MethaneSensor ∧ ¬MethaneQuery
0.29	High ∧ Low ∧ Command ∧ ¬Stop

6.1. Descriptive statistics ((RQ1) and (RQ2))

The computability of feature causes is of major interest for our evaluation. Table 1 provides an overview of the subject systems for which we generated effect sets and applied our feature causality analysis. We see that our algorithms compute feature causes in reasonable time, within a few seconds for most subject systems. The effect sets have been precomputed by various variability-aware analysis methods for effect properties as described in Section 3.1: φ stands for LTL properties; ε_B, ε_M, and ε_T for thresholds on the accuracy of a prediction model for binary size, memory footprint, and runtime; and τ_R and τ_T for reliability and runtime thresholds, respectively. This variety of properties already illustrates the wide range of applications and potential use of feature causality. Note that the time to compute the effect sets is not included in our experiments, since they were partly taken from existing benchmark sets. The sizes of valid configuration and effect sets crucially influence the time for computing feature causes, which is as expected since the complexity of Algorithm 1 is dominated by the computation of prime implicants of $(\Theta(F)\backslash\mathsf{Valid})\cup\mathsf{Effect}$. Since our implementation relies on BDDs for the representation of valid configuration and effect sets, it is however well possible that within similar sizes, computation times can significantly differ. This is mainly due to the fact that BDD sizes highly depend on the specific nature of the represented Boolean functions and the variable order chosen (Bryant, 1986). For instance, while the experiments on DUNE and BerkeleyDB (see Table 1) have similar sized sets Valid and Effect, their runtimes differ in two orders of magnitude. We see that the number of most general feature causes is often way smaller than the overall number of feature causes, which renders the creation of cause–effect covers by most general causes sensible to support concise explications. In the same vein, the application of DLS leads to great reductions of logical representations of feature causes, e.g., on average by almost 3/4 in the Elevator$_1$ subject system (see Table 1).

For (RQ1) and (RQ2), we conclude that feature causes are computable in reasonable time. A substantial reduction of the set of feature causes and cause–effect covers can be performed with DLS formulas and most general feature causes, respectively.

6.2. Feature cause explications (RQ2)

We discuss the explications of feature causes that we have generated by the example of the Minepump system (Classen et al., 2013), which is frequently used in the configurable systems' analysis community. This system models a water pump of a mine with $|F| = 11$ features on which requirements expressed in LTL are imposed (see Section 3.1). One of these requirements addresses system stabilization, formalized by the LTL formula

$$\varphi \;=\; \Diamond\Box\neg\text{pumpOn} \vee \Diamond\Box\text{pumpOn},$$

i.e., that from some point on the pump stays on or off forever. An analysis of the stabilization property using ProVeLines returned $|\mathsf{Effect}| =$ 28 configurations where the property holds and $|\mathsf{Valid}\backslash\mathsf{Effect}| = 100$ configurations where it does not hold.

The assessment which features are important for this effect property is not obvious, possibly requiring a careful investigation of all $|\mathsf{Valid}| = 128$ configurations, evaluating both, effect and non-effect configurations. Our causal analysis returned seven feature causes in an automated way, listed with their degree of blame in Table 2. They already provide hints which features are responsible for the property. Among the feature causes, three are most general, highlighted in Table 2. They have the highest degree of partial configuration blame while the most lengthy cause has the smallest degree. Our DLS heuristics on most general causes yields

$$\text{High} \wedge \text{Start} \wedge (\text{Stop} \vee \text{Low} \vee \text{MethaneAlarm})$$

providing a concise representation of feature cause candidates explicating the effect property. That is, selecting features High, Start, and one of the three features Stop, Low, or MethaneAlarm covers all causally relevant configurations for the Minepump system to stabilize. On other subject systems, explications are also effective, but with less drastic reductions than for the Minepump example.

Answering (RQ2), feature causes are of reasonable size compared to the complete analysis results, i.e., most general feature causes and DLS provide concise explications for feature causes. Responsibility and blame reflect the impact of feature causes.

6.3. Causality-guided configuration (RQ3)

Feature blame provides a quantitative measure on the causal impact of feature selections w.r.t. a set of configurations. This measure can be used to support configuration decisions, e.g., by prioritizing features with high blame values in case the effect property is desirable. We investigate such a causality-guided configuration on the velocity control loop (VCL) subject system (Dubslaff et al., 2020a,b). The VCL models an aircraft velocity controller in SimuLink for which its reliability in terms of probability of failure is of interest. A common principle to increase the reliability of a system is by *triple modular redundancy (TMR)* where system components are triplicated and their output are combined via a majority vote. Dubslaff et al. (2019a) suggested to model and analyze systems with such *protection mechanisms* using family-based methods from configurable systems' analysis. To each component they assign a *protection feature* that specifies whether a component is triplicated or not. Comprising 21 components eligible for protection, the VCL model has $|\mathsf{Valid}| = 2^{21} = 2\,097\,152$ valid feature configurations. Clearly, the highest reliability is achieved by protecting all components. However, each protection comes at its costs in terms of execution time, energy costs, and packaging size. While it is known how to determine protection configurations with optimal reliability–cost tradeoff (Dubslaff et al., 2019a), reasons for why a protection configuration is optimal or why a component was selected for protection are typically unclear. We address this issue exploiting our causal analysis methods. Using the variability-aware probabilistic model checking tool ProFeat (Chrszon et al., 2018), we generated effect sets $\mathsf{Effect}_{\rho<\tau_R}$ w.r.t. ρ mapping to the probability of failure of the VCL within two control-loop executions and reliability thresholds τ_R between 0.019 and 0.064.[1] Table 3 shows the degree of feature blame for 18 protection features of the 21 components of VCL (cf. Section 4.3). The three components not shown in the table are input components, having zero degree of blame and hence do not contribute to the systems' reliability. With the lowest threshold τ_R =

[1] To ensure timely analysis results, real-world failure probability measures were increased by a factor 100 (Dubslaff et al., 2020a). Resulting higher values might seem unrealistic but arguably do not affect the causal analysis measuring the impact of protections.

Table 3
Degree of feature blame for the VCL redundancy system.

τ_R	MFunc	Sum1, Sum4	Memory	MOne, P, vCruise	SpentFuel, I	D	Prod	DragForce, EThrust	Integrator	Sum5	Sum3	Sum2	Acceleration
0.019	0.63	1.00	0.63	1.00	1.00	1.00	0.63	1.00	1.00	1.00	1.00	1.00	1.00
0.020	0.70	0.80	0.77	0.82	0.94	1.00	0.70	0.82	1.00	0.80	0.80	0.80	1.00
0.022	0.49	0.55	0.55	0.61	0.75	0.95	0.49	0.61	1.00	0.55	0.55	0.55	1.00
0.025	0.35	0.38	0.36	0.40	0.46	0.65	0.35	0.40	1.00	0.38	0.38	0.38	1.00
0.029	0.28	0.29	0.28	0.29	0.31	0.33	0.28	0.29	0.91	0.29	0.29	0.29	1.00
0.034	0.28	0.29	0.30	0.30	0.32	0.38	0.28	0.30	0.57	0.29	0.30	0.30	0.97
0.040	0.26	0.27	0.27	0.27	0.28	0.31	0.26	0.27	0.37	0.27	0.34	0.34	0.86
0.047	0.24	0.25	0.25	0.25	0.25	0.26	0.25	0.25	0.30	0.25	0.36	0.36	0.78
0.055	0.24	0.24	0.24	0.24	0.25	0.26	0.24	0.25	0.27	0.26	0.37	0.39	0.68
0.064	0.23	0.23	0.23	0.23	0.24	0.24	0.24	0.25	0.27	0.27	0.35	0.41	0.61

Table 4
Degree of feature interaction blame for Lrzip.

	Gzip		Lrzip				Zpaq	
τ_T [s]	8	9	4–6	7	8	9	4–7	8–9
200	0.002	0.009						
300			0.033	0.033	0.033	0.033		
400				0.042	0.042	0.042		
500				0.042	0.042	0.042		
600		1-way		0.042	0.042	0.042		
700		2-way		0.056	0.056	0.056	0.056	0.056
800				0.056	0.056	0.056	0.056	0.056
900				0.063	0.063		0.063	0.063
1 000				0.063	0.063		0.063	0.063
1 100					0.071		0.071	0.071
1 200							0.083	0.083
⋮								
2 200							0.083	0.083
2 300								0.250

0.019, the effect set contains only 32 out of the 2^{21} feature configurations; almost all protections of components are responsible for the low probability of failure. Increasing the threshold lowers the degree of blame since the effect set increases, leading to less counterfactual witnesses. Using our blame analysis, we can directly deduce advice for engineers about components protections towards high reliability: With tight reliability constraints, one should protect the "Acceleration" component, followed by the "Integrator" component, as their blames are significantly higher than for other components (cf. upper rows of Table 3). When higher failure rates are acceptable, one should prefer to protect the components "Sum2" and "Sum3" instead of the "Integrator" component due to their higher impact on reliability (cf. lower rows of Table 3).

For **(RQ3)**, we conclude that feature causes and degrees of blame reveal and quantify the impact of features on the desired effect and, this way, are able to guide the feature configuration process.

6.4. *Feature interactions* (RQ4)

Our theoretical results from Section 4.4 provide a new connection of causal reasoning to feature interactions in configurable systems. In particular, all methods to explicate feature causes and precauses, as well as cause–effect covers with causes of minimal support can also be used to explicate feature interactions. For illustration of how to detect and isolate feature interactions through causal reasoning, we perform causal analysis on the Lrzip subject system, modeling a compression system for which runtime characteristics of the compression algorithms are of interest. Since the number of feature causes and their expansions can be both exponential in the number of features, a direct evaluation of the runtimes and causal analysis results is difficult. We hence investigate feature interactions through their degrees of blame as described at the end of Section 4.3. The subject effect sets $\mathtt{Effect}_{\rho>\tau_T}$ depend on the runtime $\rho : \mathsf{Valid} \to \mathbb{R}$ in seconds for a configuration compressing a file that is obtained as by Siegmund et al. (2015), Kaltenecker et al. (2019) and a runtime threshold τ_T (see end of Section 3.1). We then focus on 2-way interactions by investigating potential feature interactions between the compression algorithm and compression level responsible to have high runtimes. For this we compute degrees of feature interaction blame for partial configurations ∂ where $\mathsf{supp}(\partial) \in \{Gzip, Lrzip, Zpaq\} \times \{1, \dots, 9\}$ and $\partial(x) = \mathtt{true}$ for each $x \in \mathsf{supp}(\partial)$. In the columns of Table 4, we show the degrees of feature interaction blames for thresholds τ_T ranging from $\tau_T = 200$s to $\tau_T = 2\,300$s. Empty cells correspond to combinations of compression algorithm and level that do not appear in any cause and thus have zero blame. In these cases, we can conclude that no feature interaction takes place. Higher blame values indicate that the combined responsibility of the compression algorithm and level has a greater causal impact on runtime. Notably, we observe that, with an increasing threshold, the level of compression is increasingly responsible for longer runtime. Certain compression algorithms always have runtimes above the threshold independently of the compression level. This leads to a configuration blame of zero at any compression level, e.g., thresholds $\tau_T \leq 600$s for the *Zpaq* algorithm shown in the upper right of Table 4. Note that, in these cases, *Zpaq* serves as 1-way interaction witness according to Definition 7. All greater thresholds for *Lrzip* and *Zpaq* do not have 1-way but 2-way interaction witnesses. That is, being above the runtime threshold is a result of a feature interaction between the compression algorithm and those compression levels not showing zero blame. Notice that the sums of the given feature interaction blames for $\tau_T \geq 700$s that contain algorithms *Lrzip* or *Zpaq* add up to $1/2$. That is, no other features are to be blamed for exceeding the runtime threshold.

Features have a dedicated meaning and one would hence expect higher runtimes for higher compression levels. To this end, it seems odd that the feature interaction of *Lrzip* and compression level 9 is less to blame for higher runtimes than for levels 7 and 8. This indicates an anomaly of the feature interaction between *Lrzip* and the compression levels 7, 8, and 9. Further investigations on analysis results and feature causes support these findings: averaged over all measurements of *Lrzip* configurations, we observe runtimes of 1 064.9s at compression level 7 (standard deviation 4.1s), 1 181.7s at level 8 (standard deviation 3.2s), and only 830.5s at level 9 (standard deviation 2.6s). Hence, *Lrzip* at level 9 is not causally relevant for exceeding the execution time threshold of 900 s, as the compression level 9 feature is not contained in any cause with *Lrzip*. However, this insight is difficult to obtain relying purely on the performance influence model given by ρ, as this would require handcrafted analysis of all 432 analysis results (see Table 1).

For **(RQ4)** we conclude that feature causes can provide hints for feature interactions and anomalies arising from them. Blame measures render themselves promising to quantify the influence of feature interactions that contribute to certain effects.

6.5. Minimal $*$-Covers **(RQ2)**

Minimal prime $*$-covers have, due to its different purpose minimizing the number of prime implicants in the cover, no direct correspondence to most general causes. But as detailed in Section 4.2, they share certain commonalities. While determining minimal $*$-covers involves to solve an NP-complete problem, many heuristics exist to compute nearly-minimal $*$-covers (McGeer et al., 1993). We discovered that in all our experiments the nearly minimal $*$-covers returned by Espresso are contained in the set of $*$-covers of most general feature causes that have minimal support size. This has three implications: First, the highly optimized heuristics of two-level logic minimizers can provide a first impression about feature causality. Second, as feature interactions correspond to feature causes with minimal supports, these methods provide also first insights about feature interactions. Third, once feature causes are computed, our heuristics towards cause–effect cover with most general causes provides an efficient method for nearly minimal $*$-covers.

Concerning **(RQ2)**, we can conclude that most general feature causes in combination with distributive law reduction provides smaller representations for $*$-covers of Effect relative to Valid\Effect, which could also lead to meaningful and concise presence conditions von Rhein et al. (2015).

6.6. Effect uncertainty and precause–effect covers **(RQ5)**

To investigate the impact of uncertainty effect sets, we conducted two experiments, (1) addressing effect set underspecification and (2) accounting approximative analysis methods and threshold effect sets.

For (1), we chose the Minepump example and stepwise uniformly sampled effects Effect_i^* and non-effects $\mathsf{NEffect}_i^*$ for $i = 1, \dots, 128$ from the 128 valid configurations in Effect and Valid $\setminus$ Effect, respectively, such that $\mathsf{Effect}_{128}^* = \mathsf{Effect}$ and $\mathsf{NEffect}_{128}^* = \mathsf{Valid} \setminus \mathsf{Effect}$. During sampling, we closely maintain the ratio between effects and non-effects, i.e., $\frac{|\mathsf{Effect}_i^*|}{|\mathsf{NEffect}_i^*|} \approx \frac{|\mathsf{Effect}|}{|\mathsf{Valid}\setminus\mathsf{Effect}|}$ for all $i \in \{1, \dots, 128\}$. To evaluate the quality of the precause–effect covers $\mathcal{P}_i$ of Effect_i^* with relation to the (true) effects Effect, we computed the *fscore* in each step i, i.e.,

$$\mathrm{fscore}(i) = \frac{2 \cdot TP}{2 \cdot TP + TN + FN}$$

where $TP = |[\![\mathcal{P}_i]\!] \cap \mathsf{Effect}|$ are the true positives, $TN = |[\![\mathcal{P}_i]\!] \cap (\mathsf{Valid} \setminus \mathsf{Effect})|$ the true negatives, and $FN = |(\mathsf{Valid} \setminus [\![\mathcal{P}_i]\!]) \cap \mathsf{Effect}|$ the false negatives.

The results of these scores are shown for the 20 functional effect properties of Minepump with non-empty sets of causes in Fig. 3. Here, we averaged the score for each step of 500 sample runs. One can see that already with around 30% of knowledge about a configuration being either effect or non-effect, in all cases an fscore of more than 0.8 shows great quality of the causal description of effects from precauses. Hence, our notion of feature causality is also meaningful for effect sets that are underspecified.

To account for the approximative nature of performance models obtained by regression methods, we investigated Lrzip with the same thresholds as in Section 6.4 but with introducing a 5% uncertainty around thresholds τ_T, i.e.,

$$\begin{aligned}\mathsf{Effect}^* &= \{\theta \in \mathsf{Effect}_{<\tau_T} : \rho(\theta) \le \tau_T \cdot 0.95\}\\ \mathsf{NEffect}^* &= \{\theta \in (\mathsf{Valid} \setminus \mathsf{Effect}_{<\tau_T}) : \rho(\theta) \ge \tau_T \cdot 1.05\}.\end{aligned}$$

This introduced noise led to only slight changes of the results in feature interaction blames of Table 4. Specifically, the uncertainty leads to not being sure about the interaction between Gzip and compression level 8 at the threshold $\tau_T = 200$s, i.e., showing a blame of 0 in the leftmost upper cell of Table 4. Starting with 10% uncertainty only changes the determined causal influences also from the other compression methods, but not affecting the overall picture. Hence, we can conclude that our causal reasoning is also robust on small deviations and introducing noise in effect sets.

Concerning **(RQ5)**, we can conclude that precause–effect covers already show great performance for causal explications also when considering reasonably underspecified effect sets.

7. Discussion

In this section, we discuss potential threats to validity of our experiments and relate our findings to existing work from the literature.

7.1. Threats to validity

A threat to internal validity arises from the correctness of the analysis results from which we generated the effect sets. While for functional properties this threat is not crucial due to exact model-checking techniques used in our experiments, for non-functional properties the results have been partly established using machine learning. To mitigate this threat, we carefully chose effect-set thresholds such that the effect sets remain stable also within small threshold variations. Note that the choice of the effect set has no influence on the applicability of our causality definitions but only on to what extent causality can serve as an explication. For blame computations in our experiments, we assumed a uniform distribution over all effects, taking a developers' perspective where frequencies on how often an effect occurs in a real-world setting are not yet accessible. Other distributions could change our quantitative results, it is unlikely that they would alter our conclusions about causal influences of features and feature interactions. To increase the internal validity of our prototype, we implemented and evaluated several methods to compute causes. These include a naive brute-force approach and two additional methods to generate prime implicants, independent from the tool Espresso.

Naturally, the choice of subject systems threatens external validity, which includes the kinds of effect sets on which we evaluate causality. To alleviate this threat, we included a wide variety of systems with multiple properties from different areas to our evaluation. They comprise several real-world software systems often used to evaluate sampling strategies and performance-modeling approaches. We further added several community benchmarks from the feature-oriented model-checking community as well as a large-scale redundancy system from reliability engineering.

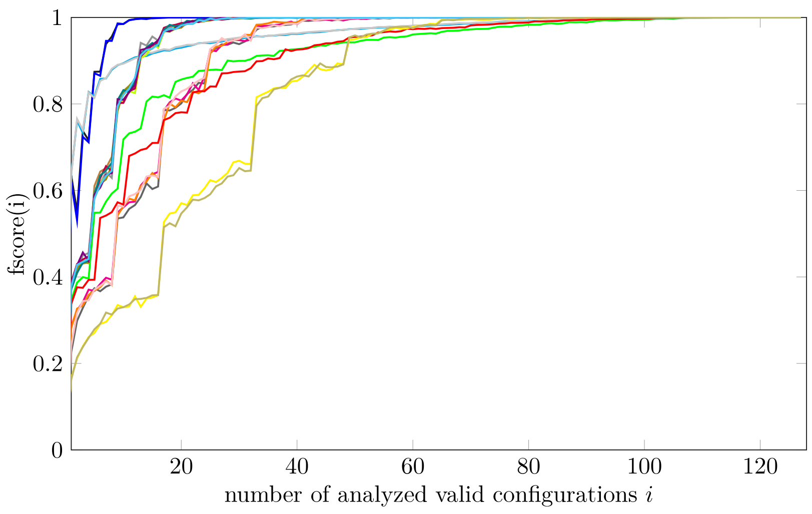

Fig. 3. Fscores from precauses for effect sets sampled with increasing size.

7.2. Related work

Various techniques for software defect detection have been proposed in the literature, ranging from testing (Myers, 2004) and static code analysis (Nielson et al., 2010) to model checking (Baier and Katoen, 2008). These techniques have been also extended for analyzing configurable systems to tackle huge configuration spaces (Thüm et al., 2014). While such methods are able to identify defects and their location, the challenge of finding root causes for defects remains. A methodology to identify causes of defects during software development is provided through *root cause analysis* (Risk and Division, 1999; Rooney and Heuvel, 2004), which can be supported by a multitude of techniques for causal reasoning (Pearl, 2009; Peters et al., 2017). To the best of our knowledge, the foundations for a combination of configurable systems analysis and causal reasoning as we presented in this paper have not yet been addressed in the literature. In the following, we discuss related work in the fields of configurable systems analysis and causal reasoning.

Configurable systems analysis and explications. For analyzing configurable software systems, many approaches have been presented in the last two decades (Post and Sinz, 2008; Thüm et al., 2014). There is broad tool support for variability-aware testing and sampling (Siegmund et al., 2012; Guo et al., 2018; Kaltenecker et al., 2019; ter Beek et al., 2019; Kaltenecker et al., 2020), static analysis (Bodden et al., 2013; Rhein et al., 2018; Weber et al., 2021; Velez et al., 2020, 2021), and model checking (Plath and Ryan, 2001; Cordy et al., 2013a; Classen et al., 2013; Apel et al., 2013; Chrszon et al., 2018; Vandin et al., 2018).

There is a substantial corpus of work on determining those features in a configurable system that are responsible for emerging effects (e.g. Kuhn et al., 2004; Yilmaz et al., 2006; Qu et al., 2008). The focus has been mainly on detecting feature interactions (Calder et al., 2003; Calder and Miller, 2006; Apel et al., 2014). Siegmund et al. (2012) and Kolesnikov et al. (2019) describe non-functional feature interactions as interactions where the composed non-functional property diverges from the aggregation of the individual contributions of the single features. Garvin and Cohen (2011) provided a formal definition of feature interaction faults based on black-box analysis to guide white-box isolation of interaction faults.

An incremental software configuration approach to optimize non-functional properties has been presented by Nair et al. (2020), complementing our causality-guided configuration exemplified in Section 6.3.

To reduce the size of propositional logic formulas in configurable systems, von Rhein et al. (2015) proposed to exclude information about valid configurations and use two-level logic minimization, e.g., by the Espresso heuristics (McCluskey, 1956; McGeer et al., 1993). Our DLS method differs from this approach by prioritizing causal information over reduction.

Causal reasoning. Algorithmic reasoning about actual causes following the approach by Halpern and Pearl (2001a), Halpern (2015) on structural equations is computationally hard in the general case (Eiter and Lukasiewicz, 2002; Aleksandrowicz et al., 2017). However, tractable instances such as the Boolean case have been identified by Eiter and Lukasiewicz (2006). For deciding whether a partial interpretation is an actual cause in the Boolean case, Ibrahim and Pretschner presented an approach based on SAT solving (Ibrahim and Pretschner, 2020). To compute all causes, their implementation relies on checking causality for all possible partial interpretations, suffering from an additional exponential blowup in the number of variables, which we avoid within our approach using prime implicant computations.

Using test generation methods relying on program trace information, program locations that are the origin of the defect can be identified (Johnson et al., 2020; Rößler et al., 2012). Analyzing differences between program states of sampled failing and passing executions, *delta debugging* identifies code positions relevant for an emerging failure (Cleve and Zeller, 2005). Similarly, causes for detects can be determined by analyzing counterexample traces (Groce and Visser, 2003; Beer et al., 2012). Faults can be also located by causal inference on graphs constructed from statement and test coverage data (Baah et al., 2010).

Iqbal et al. present a static technique to generate causal models of a given configurable system using causal interference and statistical counterfactual reasoning (Iqbal et al., 2021). This model is used to detect performance bugs and provide hints for their resolution. While we focused on actual causality and rigorous analysis, they are interested in *type causality* to answer more generic questions.

8. Concluding remarks

We introduced a formal definition and algorithms to identify causes in configurable systems that relies on counterfactual reasoning and connections to classical problems of propositional logics and circuit optimization. We demonstrated their potential by analyzing several subject systems, including real-world software systems and popular community benchmarks. To prepare for explanations of causes and their impact onto effects, we proposed explication techniques to concisely represent causes and quantify the causal impact of features. We showed that our explications are meaningful and can support the development of configurable software systems by causality-guided configuration and isolating feature interactions. With our prototypical implementation, we showed that our algorithms are effective on real-world systems of varying sizes and run in reasonable time.

While already shown to be effective, our implementation could be enhanced by directly integrating feature cause computations into optimized algorithms to compute prime implicants, e.g., relying on prime implicant computations at the level of BDDs (Coudert and Madre, 1992). Combined with state-of-the-art BDD libraries such as CUDD (Somenzi, 1997), the computation of causes might become feasible for even larger systems than considered in this paper.

Another direction is by enhancing our analysis of feature interaction blames (see Theorem 2) with in-depth white-box analyses (Garvin and Cohen, 2011) to pinpoint root causes in source code for a great variety of effect properties using feature causality.

Further applications could be imagined for *context-aware systems* where feature-oriented formalisms have been shown great applicability (Mauro et al., 2016; Dubslaff et al., 2019b; Chrszon et al., 2020). Here, our causal framework could reason about contexts responsible for certain effects, e.g., in self-adaptive systems (Aßmann et al., 2021). In this vein, *dynamic configurable systems* (Gomaa and Hussein, 2003; Dubslaff, 2021) are also an interesting direction to be considered. In such systems, features can be activated or deactivated during runtime, e.g., to model upgrade and downgrade of systems. The detection and isolation of feature interactions in dynamic configurable systems is a well-known challenge (Liu and Meier, 2009). It is a promising avenue of further work to extend our causal framework to determine root causes and identify feature interactions in the dynamic setting (Beer et al., 2012). Results of such extension could also lead to causal analysis of evolving software product lines, where the feature space (assumed to be fixed in our static setting) changes over time. Another important instance are *quantitative feature interactions* (Siegmund et al., 2012; Chrszon et al., 2020), which take into account how performance measures and quality of service (QoS) change when features interact. One possibility would be to identify feature causes with those that have the greatest impact on a quantitative measure, e.g., which features have to be active or inactive for ensuring the best increase of reliability in a fault-tolerant system. This approach would be similar to the probability-raising principle in probabilistic operational systems (Baier et al., 2021).

CRediT authorship contribution statement

Clemens Dubslaff: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition. **Kallistos Weis:** Software, Validation, Investigation, Data curation, Writing – review & editing, Visualization. **Christel Baier:** Validation, Formal analysis, Writing – review & editing, Funding acquisition. **Sven Apel:** Validation, Writing – review & editing, Funding acquisition, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The research data and code is publicly available.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jss.2023.111915.

References

Abal, I., Melo, J., Stănciulescu, Ş., Brabrand, C., Ribeiro, M., Wąsowski, A., 2018. Variability bugs in highly configurable systems: A qualitative analysis. Trans. Softw. Eng. Methodol. 26.

Aleksandrowicz, G., Chockler, H., Halpern, J.Y., Ivrii, A., 2017. The computational complexity of structure-based causality. Artif. Intell. Res. 58, 431–451.

Apel, S., Atlee, J.M., Baresi, L., Zave, P., 2014. Feature interactions: The next generation (Dagstuhl Seminar 14281). Dagstuhl Rep. 4, 1–24.

Apel, S., Batory, D.S., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer.

Apel, S., von Rhein, A., Wendler, P., Größlinger, A., Beyer, D., 2013. Strategies for product-line verification: Case studies and experiments. In: Proceedings of the 35th International Conference on Software Engineering. ICSE, IEEE, pp. 482–491.

Aßmann, U., Baier, C., Dubslaff, C., Grzelak, D., Hanisch, S., Hartono, A.P.P., Köpsell, S., Lin, T., Strufe, T., 2021. Tactile computing: Essential building blocks for the tactile internet. In: Tactile Internet with Human-in-the-Loop. Academic Press, pp. 301–326.

Baah, G.K., Podgurski, A., Harrold, M.J., 2010. Causal inference for statistical fault localization. In: Proceedings of the 19th International Symposium on Software Testing and Analysis. ISSTA, ACM, pp. 73–84.

Baier, C., Dubslaff, C., Funke, F., Jantsch, S., Majumdar, R., Piribauer, J., Ziemek, R., 2021. From verification to causality-based explications. In: Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, Vol. LIPIcs:198. ICALP, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 1–20.

Baier, C., Katoen, J.-P., 2008. Principles of Model Checking. MIT Press.

ter Beek, M.H., Damiani, F., Lienhardt, M., Mazzanti, F., Paolini, L., 2019. Static analysis of featured transition systems. In: Proceedings of the 23rd Systems and Software Product Line Conference. SPLC, ACM, pp. 39–51.

Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.J., 2012. Explaining counterexamples using causality. Form. Methods Syst. Des. 40, 20–40.

Bodden, E., Tolêdo, T., Ribeiro, M., Brabrand, C., Borba, P., Mezini, M., 2013. SPL$^{\text{LIFT}}$: statically analyzing software product lines in minutes instead of years. In: Proceedings of the 34th Conference on Programming Language Design and Implementation. PLDI, ACM, pp. 355–364.

Bryant, R.E., 1986. Graph-based algorithms for Boolean function manipulation. Trans. Comput. 35, 677–691.

Calder, M., Kolberg, M., Magill, E.H., Reiff-Marganiec, S., 2003. Feature interaction: a critical review and considered forecast. Comput. Netw. 41, 115–141.

Calder, M., Miller, A., 2006. Feature interaction detection by pairwise analysis of LTL properties—A case study. Form. Methods Syst. Des. 28, 213–261.

Chandra, A.K., Markowsky, G., 1978. On the number of prime implicants. Discrete Math. 24, 7–11.

Chockler, H., Halpern, J.Y., 2004. Responsibility and blame: A structural-model approach. Artificial Intelligence Res. 22, 93–115.

Chrszon, P., Baier, C., Dubslaff, C., Klüppelholz, S., 2020. From features to roles. In: Proceedings of the 24th Systems and Software Product Line Conference. SPLC, ACM, pp. 1–11.

Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C., 2018. ProFeat: feature-oriented engineering for family-based probabilistic model checking. Form. Asp. Comput. 30, 45–75.

Clarke, E.M., Emerson, E.A., Sistla, A.P., 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. Trans. Programm. Lang. Syst. 8, 244–263.

Classen, A., Boucher, Q., Heymans, P., 2011. A text-based approach to feature modelling: Syntax and semantics of TVL. Sci. Comput. Programm. 76, 1130–1143.

Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.-Y., 2012. Model checking software product lines with SNIP. Int. J. Softw. Tools Technol. Transfer 14, 589–612.

Classen, A., Cordy, M., Schobbens, P.-Y., Heymans, P., Legay, A., Raskin, J.-F., 2013. Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. Trans. Softw. Eng. 39, 1069–1089.

Cleve, H., Zeller, A., 2005. Locating causes of program failures. In: Proceedings of the 27th International Conference on Software Engineering. ICSE, ACM, pp. 342–351.

Cordy, M., Classen, A., Heymans, P., Schobbens, P., Legay, A., 2013a. ProVeLines: a product line of verifiers for software product lines. In: Proceedings of the 17th Systems and Software Product Line Conference. SPLC, ACM, pp. 141–146.

Cordy, M., Schobbens, P.-Y., Heymans, P., Legay, A., 2013b. Beyond boolean product-line model checking: Dealing with feature attributes and multi-features. In: Proceedings of the 35th International Conference on Software Engineering. ICSE, IEEE, pp. 472–481.

Coudert, O., Madre, J.C., 1992. Implicit and incremental computation of primes and essential primes of boolean functions. In: Proceedings of the 29th European Design Automation Conference. EURO-DAC, IEEE, pp. 36–39.

Dorn, J., Apel, S., Siegmund, N., 2020. Mastering uncertainty in performance estimations of configurable software systems. In: Proceedings of the 35th Conference on Automated Software Engineering. ASE, IEEE, pp. 684–696.

Drake, C., 2015. Pyeda: Data structures and algorithms for electronic design automation. In: Proceedings of the 14th Python in Science Conference. SciPy, pp. 26–31.

Dubslaff, C., 2019. Compositional feature-oriented systems. In: Proceedings of the 17th Conference on Software Engineering and Formal Methods, Vol. LNCS:12226. SEFM, Springer, pp. 162–180.

Dubslaff, C., 2021. Quantitative Analysis of Configurable and Reconfigurable Systems (Ph.D. thesis). TU Dresden, Institute for Theoretical Computer Science.

Dubslaff, C., Baier, C., Klüppelholz, S., 2015. Probabilistic model checking for feature-oriented systems. Trans. Aspect-Oriented Softw. Dev. LNCS:8989, 180–220.

Dubslaff, C., Ding, K., Morozov, A., Baier, C., Janschek, K., 2019a. Breaking the limits of redundancy systems analysis. In: Proceedings of the 29th European Safety and Reliability Conference. ESREL, pp. 2317–2325.

Dubslaff, C., Koopmann, P., Turhan, A.-Y., 2019b. Ontology-mediated probabilistic model checking. In: Proceedings of the 15th Conference on Integrated Formal Methods, Vol. LNCS:11918. IFM, Springer, pp. 194–211.

Dubslaff, C., Morozov, A., Baier, C., Janschek, K., 2020a. Iterative variable reordering: Taming huge system families. In: Proceedings of the 4th Workshop on Models for Formal Analysis of Real Systems, Vol. EPTCS:316. MARS, Open Publishing Association, pp. 121–133.

Dubslaff, C., Morozov, A., Baier, C., Janschek, K., 2020b. Reduction methods on error-propagation graphs for quantitative systems reliability analysis. In: Proceedings of the 30th European Safety and Reliability Conference (ESREL) and 15th Probabilistic Safety Assessment and Management Conference. PSAM.

Dubslaff, C., Weis, K., Baier, C., Apel, S., 2022. Causality in configurable software systems. In: Proceedings of the 44th International Conference on Software Engineering. ICSE'22.

Dubslaff, C., Weis, K., Baier, C., Apel, S., 2023a. FeatCause – Sources and Data. Zenodo, http://dx.doi.org/10.5281/zenodo.8350560.

Dubslaff, C., Weis, K., Baier, C., Apel, S., 2023b. FeatCause – Github. URL https://github.com/dubslaff/FeatCause.

Eells, E., 1991. Probabilistic Causality. Cambridge University Press.

Eiter, T., Lukasiewicz, T., 2002. Complexity results for structure-based causality. Artificial Intelligence 142, 53–89.

Eiter, T., Lukasiewicz, T., 2006. Causes and explanations in the structural-model approach: Tractable cases. Artif. Intell. 170 (6–7), 542–580.

Garvin, B.J., Cohen, M.B., 2011. Feature interaction faults revisited: An exploratory study. In: Proceedings of the 22nd International Symposium on Software Reliability Engineering. ISSRE, ACM, pp. 90–99.

Gomaa, H., Hussein, M., 2003. Dynamic software reconfiguration in software product families. In: Proceedings of the 5th Workshop on Software Profuct Family Engineering, Vol. LNCS:3014. PFE, pp. 435–444.

Good, I.J., 1959. A theory of causality. British J. Philos. Sci. 9, 307–310.

Groce, A., Visser, W., 2003. What went wrong: Explaining counterexamples. In: Proceedings of the 10th Workshop on Model Checking of Software, Vol. LNCS:2648. Springer, pp. 121–135.

Guo, J., Yang, D., Siegmund, N., Apel, S., Sarkar, A., Valov, P., Czarnecki, K., Wasowski, A., Yu, H., 2018. Data-efficient performance learning for configurable systems. Empir. Softw. Eng. 23, 1826–1867.

Halpern, J.Y., 2015. A modification of the Halpern-Pearl definition of causality. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence. IJCAI, AAAI, pp. 3022–3033.

Halpern, J.Y., Pearl, J., 2001a. Causes and explanations: A structural-model approach - part I: Causes. In: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence. UAI, Morgan Kaufmann, pp. 194–202.

Halpern, J.Y., Pearl, J., 2001b. Causes and explanations: A structural-model approach - part II: Explanations. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence. IJCAI, Morgan Kaufmann, pp. 27–34.

Hansson, H., Jonsson, B., 1994. A logic for reasoning about time and reliability. Form. Asp. Comput. 6, 512–535.

Hemaspaandra, E., Schnoor, H., 2011. Minimization for generalized boolean formulas. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence. IJCAI, IJCAI/AAAI, pp. 566–571.

Ibrahim, A., Pretschner, A., 2020. From checking to inference: Actual causality computations as optimization problems. In: Proceedings of the 18th Symposium Automated Technology for Verification and Analysis, Vol. LNCS:12302. ATVA, Springer, pp. 343–359.

Iqbal, M.S., Krishna, R., Javidian, M.A., Ray, B., Jamshidi, P., 2021. CADET: Debugging and fixing misconfigurations using counterfactual reasoning. arXiv:2010.06061.

Johnson, B., Brun, Y., Meliou, A., 2020. Causal testing: Understanding defects' root causes. In: Proceedings of the 42nd International Conference on Software Engineering. ICSE, ACM, pp. 87–99.

Kaltenecker, C., Grebhahn, A., Siegmund, N., Apel, S., 2020. The interplay of sampling and machine learning for software performance prediction. Software 37, 58–66.

Kaltenecker, C., Grebhahn, A., Siegmund, N., Guo, J., Apel, S., 2019. Distance-based sampling of software configuration spaces. In: Proceedings of the 41st International Conference on Software Engineering. ICSE, IEEE, pp. 1084–1094.

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep., Carnegie-Mellon University Software Engineering Institute.

Kolesnikov, S.S., Siegmund, N., Kästner, C., Grebhahn, A., Apel, S., 2019. Tradeoffs in modeling performance of highly configurable software systems. Softw. Syst. Model. 18, 2265–2283.

Kuhn, D.R., Wallace, D.R., Gallo, A.M., 2004. Software fault interactions and implications for software testing. Trans. Softw. Eng. 30, 418–421.

Lewis, D., 1973. Counterfactual theories of causation. J. Phil. 556–567.

Liu, Y., Meier, R., 2009. Resource-aware contracts for addressing feature interaction in dynamic adaptive systems. In: Proceedings of the 5th International Conference on Autonomic and Autonomous Systems. ICAS, IEEE, pp. 346–350.

Mauro, J., Nieke, M., Seidl, C., Yu, I.C., 2016. Context aware reconfiguration in software product lines. In: Proceedings of the 10th Workshop on Variability Modelling of Software-Intensive Systems. VaMoS, ACM, pp. 41–48.

McCluskey, Jr., E.J., 1956. Minimization of Boolean functions. Bell Syst. Tech. J. 35, 1417–1444.

McGeer, P., Sanghavi, J., Brayton, R., Vincentelli, A.S., 1993. Espresso-signature: A new exact minimizer for logic functions. In: Proceedings of the 30th Design Automation Conference. DAC, ACM, pp. 618–624.

Myers, G.J., 2004. The Art of Software Testing. Wiley.

Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S., 2020. Finding faster configurations using FLASH. Trans. Softw. Eng. 46, 794–811.

Nielson, F., Nielson, H.R., Hankin, C., 2010. Principles of Program Analysis. Springer.

Paul, W.J., 1975. Boolesche minimalpolynome und überdeckungsprobleme. Acta Inform. 4, 321–336.

Pearl, J., 2009. Causality: Models, Reasoning and Inference, second ed. Cambridge University Press.

Peters, J., Janzing, D., Schölkopf, B., 2017. Elements of Causal Inference: Foundations and Learning Algorithms. MIT Press.

Plath, M., Ryan, M., 2001. Feature integration using a feature construct. Sci. Comput. Programm. 41, 53–84.

Pnueli, A., 1977. The temporal logic of programs. In: Proceedings of the 18th Symposium on Foundations of Computer Science. SFCS, IEEE, pp. 46–57.

Post, H., Sinz, C., 2008. Configuration lifting: Verification meets software configuration. In: Proceedings of the 23rd Conference on Automated Software Engineering. ASE, IEEE, pp. 347–350.

Qu, X., Cohen, M.B., Rothermel, G., 2008. Configuration-aware regression testing: An empirical study of sampling and prioritization. In: Proceedings of the 17th International Symposium on Software Testing and Analysis. ISSTA, ACM, pp. 75–86.

Rhein, A.V., Liebig, J., Janker, A., Kästner, C., Apel, S., 2018. Variability-aware static analysis at scale: An empirical study. Trans. Softw. Eng. Methodol. 27.

Risk, A.G.I., Division, R., 1999. Root Cause Analysis Handbook: A Guide To Effective Incident Investigation. In: G - Reference, Information and Interdisciplinary Subjects Series, Government Institutes.

Rodrigues, G.N., Alves, V., Nunes, V., Lanna, A., Cordy, M., Schobbens, P., Sharifloo, A.M., Legay, A., 2015. Modeling and verification for probabilistic properties in software product lines. In: Proceedings of the 16th Symposium on High Assurance Systems Engineering. HASE, IEEE, pp. 173–180.

Rooney, J.J., Heuvel, L.N.V., 2004. Root cause analysis for beginners. Qual. Prog. 37, 45.

Rößler, J., Fraser, G., Zeller, A., Orso, A., 2012. Isolating failure causes through test case generation. In: Proceedings of the 2012 International Symposium on Software Testing and Analysis. ISSTA, ACM, pp. 309–319.

Siegmund, N., Grebhahn, A., Apel, S., Kästner, C., 2015. Performance-influence models for highly configurable systems. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. ESEC/FSE, ACM, pp. 284–294.

Siegmund, N., Kolesnikov, S.S., Kästner, C., Apel, S., Batory, D., Rosenmüller, M., Saake, G., 2012. Predicting performance via automated feature-interaction detection. In: Proceedings of the 34th International Conference on Software Engineering. ICSE, IEEE, pp. 167–177.

Siegmund, N., Rosenmüller, M., Kästner, C., Giarrusso, P.G., Apel, S., Kolesnikov, S.S., 2013a. Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. Inf. Softw. Technol. 55, 491–507, Special Issue on Software Reuse and Product Lines.

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G., 2012. SPL conqueror: Toward optimization of non-functional properties in software product lines. Softw. Qual. J. 20, 487–517.

Siegmund, N., Sven, A.v., Apel, 2013b. Family-based performance measurement. SIGPLAN Not. 49 (3), 95–104.

Somenzi, F., 1997. CUDD 3.0.0. URL http://vlsi.colorado.edu/~fabio/CUDD/html/.

Spellman, B.A., Kincannon, A., 2001. The relation between counterfactual ("But for") and causal reasoning: Experimental findings and implications for Jurors' decisions. Law Contemp. Probl. (ISSN: 00239186) 64 (4), 241–264, URL http://www.jstor.org/stable/1192297.

Strzemecki, T., 1992. Polynomial-time algorithms for generation of prime implicants. J. Complexity 8, 37–63.

ter Beek, M.H., Legay, A., Lluch Lafuente, A., Vandin, A., 2016. Statistical model checking for product lines. In: Proceedings of the 7th Symposium on Leveraging Applications of Formal Methods, Vol. LNCS:9952. ISoLA, Springer, pp. 114–133.

Thüm, T., 2020. A BDD for linux? The knowledge compilation challenge for variability. In: Proceedings of the 24th Systems and Software Product Line Conference. SPLC, ACM.

Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G., 2014. A classification and survey of analysis strategies for software product lines. Comput. Surv. 47, 6:1–6:45.

Umans, C., Villa, T., Sangiovanni-Vincentelli, A.L., 2006. Complexity of two-level logic minimization. Trans. Comput.-Aided Des. Integr. Circuits Syst. 25, 1230–1246.

Vandin, A., ter Beek, M.H., Legay, A., Lluch Lafuente, A., 2018. QFLan: A tool for the quantitative analysis of highly reconfigurable systems. In: Proceedings on the 22nd Symposium on Formal Methods, Vol. LNCS:10951. FM, Springer, pp. 329–337.

Velez, M., Jamshidi, P., Sattler, F., Siegmund, N., Apel, S., Kästner, C., 2020. ConfigCrusher: Towards white-box performance analysis for configurable systems. Autom. Softw. Eng. 27, 265–300.

Velez, M., Jamshidi, P., Siegmund, N., Apel, S., Kästner, C., 2021. White-box analysis over machine learning: Modeling performance of configurable systems. In: Proceedings of the 43rd International Conference on Software Engineering. ICSE, IEEE.

von Rhein, A., Grebhahn, A., Apel, S., Siegmund, N., Beyer, D., Berger, T., 2015. Presence-condition simplification in highly configurable systems. In: Proceedings of the 37th International Conference on Software Engineering. ICSE, IEEE, pp. 178–188.

Wachter, S., Mittelstadt, B.D., Russell, C., 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. Harv. J. Law Technol. 31, 841–887.

Weber, M., Apel, S., Siegmund, N., 2021. White-box performance-influence models: A profiling and learning approach. In: Proceedings of the 43rd International Conference on Software Engineering. ICSE, IEEE.

Williamson, J., 2009. Probabilistic theories of causation. Oxf. Handb. Causation 185–212.

Yilmaz, C., Cohen, M.B., Porter, A.A., 2006. Covering arrays for efficient fault characterization in complex configuration spaces. Trans. Softw. Eng. 32, 20–34.

Zave, P., 2001. Feature-oriented description, formal methods, and DFC. In: Proceedings of the Workshop on Language Constructs for Describing Features. Springer, pp. 11–26.

Zeller, A., 2002. Isolating cause-effect chains from computer programs. In: Proceedings of the 10th Symposium on Foundations of Software Engineering. FSE, ACM, pp. 1–10.

Zhang, Y., Guo, J., Blais, E., Czarnecki, K., 2015. Performance prediction of configurable software systems by Fourier learning (T). In: Proceedings of the 30th Conference on Automated Software Engineering. ASE, IEEE, pp. 365–373.