Quantitative Analyse von konfigurierbaren und rekonfigurierbaren Systemen[1]

Clemens Dubslaff[2]

Abstract: Die Fülle an Konfigurationsoptionen und der daraus resultierende Reichtum an Systemvarianten stellen Entwickler von modernen Computersystemen vor großen Herausforderungen. Weitere Systemanforderungen an Adaptivität, Rekonfigurierbarkeit und an quantitative Aspekte wie Zuverlässigkeit, Energieverbrauch oder Latenz kommen erschwerend hinzu. Formale Analysen sind daher unabdingbar, um die Auswirkungen von Konfigurationsoptionen und deren Interaktionen einzuschätzen und fehlerfreie Systeme zu garantieren. Die vorgestellte Dissertation führt ein kompositionelles Modellierungs- und Analyseframework ein, welches alle genannten Herausforderungen adressiert und effektive Lösungen bietet, formale quantitative Analysen auch für bisher unmöglich große konfigurierbare Systeme durchzuführen. An real existierenden Systemen wird dessen Anwendbarkeit demonstriert und mit neuen Methoden zu kausalen Erklärungen von Analyseresultaten ergänzt.

1 Einleitung

Fast jedes Softwaresystem ist heutzutage konfigurierbar. Angefangen auf der Compilerebene, in der z.B. durch `#ifdef`-Optionen spezielle Optimierungen für die verwendete Hardware aktiviert werden können, bis hin zur Produktebene, bei der Kunden verschiedene Varianten der gleichen Software mit unterschiedlichsten Funktionalitäten zur Auswahl stehen. Die Zahl der möglichen Systemvarianten ist hierbei häufig exponentiell in der Anzahl der Konfigurationsoptionen. Dies stellt insbesondere Software- und Systementwickler vor große Herausforderungen, denn sie müssen für die Vorhersage, Erkennung, Beschreibung und das Beseitigen von Fehlern und ungewolltem Verhalten eine Vielzahl von Kombinationen von Konfigurationen betrachten. Das Verändern von Konfigurationen zur Laufzeit des Systems, sogenannte *Rekonfigurationen*, erhöhen die Komplexität nochmals. Diese sind z.B. in modernen Softwaresystemen durch Updates, Einflüsse von Nutzern und deren Einstellungen, sowie durch Selbstadaptivität und sich verändernden Umgebungen in natürlicher Weise gegeben. Neben der Konfigurierbarkeit und Rekonfigurierbarkeit von Software stellt deren Einbettung in cyber-physische Systeme (CPS) eine weitere Quelle von Komplexität dar. Das Verhalten von CPS hängt von *quantitativen Aspekten* wie dem Energieverbrauch, der Bandbreite oder Fehlerwahrscheinlichkeiten ab, welche allesamt in neuen Technologien wie 5G-Netzwerken, dem taktile Internet, sowie dem autonomen Fahren eine zentrale Rolle spielen. Um diesen Quellen der Komplexität zu begegnen und fehlerfreie Systeme

[1] Englischer Titel der Dissertation: "Quantitative Analysis of Configurable and Reconfigurable Systems"

[2] Technische Universität Dresden, Institut für Theoretische Informatik, Nöthnitzer Straße 46, 01187 Dresden, Deutschland; email: clemens.dubslaff@tu-dresden.de

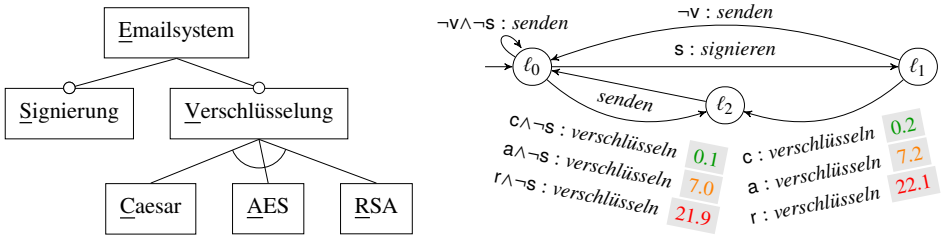


Abb. 1: Featurediagramm (links) und featured Transitionssystem mit Kostenannotationen (rechts)

zu entwickeln, werden verschiedenste Analysemethoden eingesetzt. Simulative Ansätze und Testmethoden können zeigen, dass Fehler oder Systemeigenschaften beim Ausführen des Systems möglich sind. Dahingehend zeigen Verifikationstechniken die Abwesenheit von Fehlern und geben Garantien an Systemeigenschaften, sind aber meist nur für Kernkomponenten und kleine Systeme praktikabel. Eine weit verbreitete Verifikationsmethode ist *Model Checking* [z.B. BK08], bei der das System und dessen Anforderungen formal spezifiziert und erschöpfend analysiert wird. *Probabilistisches Model Checking (PMC)* unterstützt zudem die Betrachtung von Wahrscheinlichkeiten und quantitativen Größen und wird erfolgreich zur quantitativen Analyse von einer Vielzahl von Systemen angewendet. Symbolische Methoden basierend auf *binären Entscheidungsdiagrammen (BDDs)* können Model Checking auch für reale Computersysteme ermöglichen [BK08].

Um konfigurierbare Systeme zu spezifizieren und analysieren, stellen *Features* ein weit verbreitetes Konzept dar. Sie kapseln optionale oder inkrementelle Funktionalitäten und wurden bisher in erster Linie für den Entwurf und Entwicklung von Software-Produktlinien (SPLs) verwendet [Ap13]. Systemvarianten ergeben sich durch die Auswahl von Features und korrespondieren somit zu Konfigurationen als Mengen von Features. Üblicher Weise werden feature-orientierte Systeme mittels eines zweistufigen Ansatzes spezifiziert:

(1) Basierend auf einer Domänenanalyse werden Features isoliert und deren Beziehungen in einem *Variabilitätsmodell* ausgedrückt.
(2) Das *Verhalten von Features* wird spezifiziert, z.B. in einer geeigneten Programmiersprache welche operationelle Abhängigkeiten zwischen Features beschreiben kann.

Ein gängiges Variabilitätsmodell stellen Featurediagramme dar [Ka90]. In solchen Diagrammen werden die konfigurationsrelevante Abhängigkeiten von Features in einer hierarchischen Baumstruktur modelliert, bei denen Kindfeatures die Auswahl von Elternfeatures implizieren. In Abb. 1 ist ein Featurediagramm für ein konfigurierbares Emailsystem dargestellt. Dieses System enthält optionale Signierungs- und Verschlüsslungsfeatures (modelliert durch einen Kreis oberhalb des Features), sowie Verschlüsslungsarten Caesar, AES und RSA, von denen bei ausgewähltem Verschlüsslungsfeature genau eine Art ausgewählt werden muss (modelliert durch die verbundene Verzweigung zwischen den Verschlüsslungsarten). Durch das Featurediagramm wird eine Systemfamilie mit acht Systemvarianten beschrieben: ohne Verschlüsslung oder mit einer der drei Verschlüsslungsarten, je mit Signierungsfeature oder

ohne. Ein Beispiel ist die Systemvariante ohne Signierung und mit AES Verschlüsslung, welche durch die Konfiguration $\{\mathsf{e}, \mathsf{v}, \mathsf{a}\}$ als Menge von Features formalisiert wird.
Für die Spezifikation von Verhalten konfigurierbarer Systeme unterscheidet man zwischen *annotativen* und *kompositionellen* Ansätzen [KA08]. Beide Ansätze haben komplementäre Vor- und Nachteile bezüglich Granularität, Erweiterbarkeit und Analyse. Mittels annotativer Methoden werden Verhalten mit aussagenlogischen Ausdrücken über Features versehen. Diese Verhalten sind nur in jenen Konfigurationen aktiv, welche den annotierten Featureausdruck erfüllen. Beispiele hierfür sind *featured Transitionssysteme (FTS)* [Cl13], bei denen Zustandsübergänge mit Featureausdrücken annotiert werden, und `#ifdef`-Optionen in C-Programmen. In Abb. 1 ist auf der rechten Seite ein FTS für das konfigurierbare Emailsystem dargestellt. Startend im Zustand ℓ_0 kann eine Email nur dann direkt versendet werden, wenn weder das Signierungs- noch das Verschlüsselungsfeature ausgewählt wurde (formalisiert durch den Featureausdruck $\neg\mathsf{v}\wedge\neg\mathsf{s}$). Ansonsten muss eine Email zunächst signiert oder verschlüsselt werden (Transitionen von ℓ_0 nach ℓ_1 oder ℓ_2). Annotative Ansätze bieten feingranulare Spezifikationsmöglichkeiten und erlauben eine *familienbasierte Analyse*, in der alle Systemvarianten in einem Analyseschritt betrachtet werden und symbolische Methoden Gemeinsamkeiten zwischen Systemvarianten ausnutzen können. Kompositionelle Modellierungsansätze für konfigurierbare Systeme beschreiben das Verhalten jedes Features isoliert als *Featuremodul*. Das Gesamtverhalten einer Systemvariante entsteht durch die Komposition der Featuremodule von ausgewählten Features mittels eines Kompositionsoperators. Diese Methode wird hauptsächlich in der Entwicklung von SPLs angewendet [Ap13], wobei als Kompositionsoperator die *Superimposition* eingesetzt wird. Superimposition beschreibt wie ein Featuremodul das Verhalten eines Basissystems verändert. Im FTS Beispiel von Abb. 1 würde durch das Signierungsfeature z.B. das direkte Versenden einer Mail in ℓ_0 durch eine Transition nach ℓ_1 ersetzt werden. Kompositionelle Ansätze haben Vorteile in der Trennung von Zuständigkeiten, Modularisierung und der daraus resultierenden Wart- und Erweiterbarkeit des konfigurierbaren Systems.

Aufgrund der komplementären Vor- und Nachteile annotativer und kompositioneller Methoden, stellten Kästner und Apel die Frage, ob ein *hybrider* Ansatz möglich ist, der die Vorteile beider Methoden vereint [KA08]. Die Dissertation [Du21] liefert mehrere fundamentale Beiträge zur Spezifikation und (quantitativen) Analyse von konfigurierbaren Systemen, indem u.a. ein hybriden Ansatz vorgestellt wird und dessen Vorteile formal bewiesen werden. Beiträge sind unter anderem:

(1) Ein *annotativ kompositionelles Framework* zur Spezifikation und Analyse von konfigurierbaren Systemen, u.a. für quantitative Erweiterungen von FTS.
(2) *Kompositionelle Familienmodelle*, welche eine familienbasierte Analyse auch für kompositionelle Ansätze ermöglichen.
(3) Modellierung und Analyse *rekonfigurierbarer Systeme* mit Hilfe von Familienmodellen.
(4) *Reduktion von Analyseproblemen* für konfigurierbare Systeme auf Standardanalyseprobleme und deren algorithmische Lösung.
(5) Neue Methoden zur *Reduktion von Modellgrößen*, insbesondere für Familienmodelle.

(6) Neuartige Auswertung von Analyseresultaten in konfigurierbaren Systemen durch *kausale Analyse* auf der Abstraktionsebene der Features.
(7) Anwendung und Evaluation auf reale (Hardware-)Systeme.

2 Konfigurierbare probabilistische Systeme

Kern der Dissertation [Du21] bildet die Formalisierung eines hybriden Ansatzes zur Spezifikation und Analyse konfigurierbarer Systeme mit quantitativen Aspekten.

Definition 1 *Ein* annotatives kompositionelles System (ACS) *ist ein Tupel* $\mathsf{Acs} = (F, \mathcal{V}, \mathfrak{M}, \phi, \prec, \circ)$ *über eine Menge von Features F, einem Variabilitätsmodell $\mathcal{V}$, einer Menge von annotierten Featuremodulen $\mathfrak{M}$, einer Zuordnung von Featuremodulen zu Features über eine Funktion $\phi\colon F \to \mathfrak{M}$, sowie eine totale Kompositionsordnung über Features $\prec$ und eine Kompositionsoperation $\circ$.*

Die Dissertation betrachtet hauptsächlich Featuremodule als annotierte Programme in einer feature-orientierten und probabilistischen Variante von Dijkstras *Guarded Commands*. Das präsentierte Framework ist jedoch generisch und die erbrachten Ergebnisse lassen sich auf eine Vielzahl von Formalismen übertragen. Für diese Zusammenfassung reicht es, sich Featuremodule als quantitative Varianten von FTS vorzustellen [DBK15; DKB14]. Ein Beispiel ist das FTS in Abb. 1, bei dem Transitionen mit zu den erwartenden Verschlüsselungszeiten annotiert sind. Als Kompositionsoperationen fokussieren wir uns auf feature-orientierte Varianten der parallelen Komposition $\|$ und Superimposition $\bullet$ [Du19a].

Das Verhalten einer Systemvariante ergibt sich für ACS mittels Komposition und Projektion. Werden Features $X \subseteq F$ ausgewählt, so ist die korrespondierende Systemvariante $\phi(X)\restriction_X$, wobei $\phi(X) = \phi(x_1) \circ \phi(x_2) \circ \ldots \circ \phi(x_k)$ für $X = \{x_1, x_2, \ldots, x_k\}$ mit $x_1 \prec x_2 \prec \ldots \prec x_k$ die Komposition von Featuremodulen für die Features in X gemäß der Featureordnung entspricht und die Projektion $\phi(X)\restriction_X$ all jene Verhalten in $\phi(X)$ entfernt, deren Featureausdrücke nicht in X erfüllt sind. Demnach ist der ACS Ansatz im Kern kompositionell, erlaubt aber auch feingranulare Bedingungen durch annotative Elemente.

2.1 Kompositionelle Familienmodelle

Waren Familienmodelle bisher nur für annotative Verfahren bekannt, werden diese auch für kompositionelle Ansätze in der Dissertation eingeführt. Zugrunde liegt eine erstaunlich einfache Beobachtung: eine Spezifikation, welche die Verhalten aller Featuremodule beinhalten soll, benötigt die Komposition aller Featuremodule für den gesamten Konfigurationsraum.

Definition 2 *Ein ACS* $\mathsf{Acs} = (F, \mathcal{V}, \mathfrak{M}, \phi, \prec, \circ)$ *ist ein* kompositionelles Familienmodell *wenn $\phi(F)\restriction_X = \phi(X)\restriction_X$ für alle validen Konfigurationen $X \subseteq F$.*

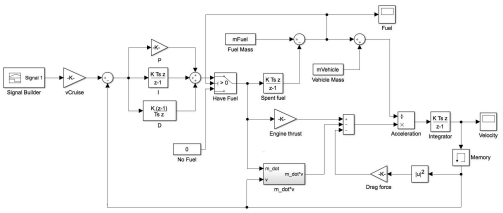
Während in annotativen Ansätzen generell von einem Familienmodell ausgegangen wird, ist nicht jedes ACS a priori ein kompositionelles Familienmodell. Grund sind mögliche Interaktionen zwischen Features, die in den validen Systemvarianten nicht auftauchen, jedoch aber bei einer Komposition aller Featuremodule. Im Emailbeispiel von Abb. 1 könnte es z.B. geschehen, dass unvorhergesehen mehrere Verschlüsselungsmethoden auch nach Projektion ausführbar sind, weil Featuremodule unabhängig voneinander entwickelt wurden.

Theorem 1 *Für jedes* $\circ$*-ACS mit* $\circ \in \{\|, \bullet\}$ *kann in polynomieller Zeit ein kompositionelles Familienmodell mit gleichen Systemvarianten konstruiert werden.*

Familienmodelle sind insbesondere für die familienbasierte Analyse unter Verwendung von symbolischen Methoden von Vorteil [Th14]. Symbolische Methoden, z.B. mittels BDDs, nutzen Gemeinsamkeiten zwischen Systemvarianten aus und können die exponentiell vielen Systemvarianten komprimiert darstellen und analysieren. Neben dieser bekannten Rolle von Familienmodellen stellt die Dissertation einen neuen Vorteil vor. Rekonfigurationen können in Familienmodellen elegant modelliert werden, da das Modell alle Verhalten und somit auch die Verhalten nach den Rekonfigurationen beinhaltet. Die Grundidee hierbei ist, die Rekonfigurationen im Variabilitätsmodell mit einzubeziehen: Zustände in diesem Variabilitätsmodell stehen für valide Konfigurationen und Transitionen beschreiben Rekonfigurationen. Ein solches Modell deckt den statischen Fall ebenfalls ab, indem valide Konfigurationen initial und keine Rekonfigurationen modelliert sind. Für ein ACS Acs löst ein spezieller (paralleler) Operator $\bowtie$ in Verbindung mit dem Familienmodell $\phi(F)$ Featureausdrücke auf und führt zur *Semantik* $\phi(F) \bowtie \mathcal{V}$ von Acs. Diese Semantik ist ein klassischen Programm, welches die Systemkonfigurationen mit in den Zuständen kodiert. Damit reduziert sich die funktionale und quantitative Analyse von (re)konfigurierbaren Systemen auf Standardmethoden, welche direkt auf $\phi(F) \bowtie \mathcal{V}$ angewendet werden können. Dies komplementiert Methoden der Literatur, welche Featureannotationen speziell in die Analysemethoden mit einbeziehen [Cl13; Th14] und somit für jede neuartige Analysemethode wiederholt auf konfigurierbare Systeme angepasst werden müssen. Zudem waren bisher keine familienbasierte Analysemethoden für rekonfigurierbare Systeme bekannt und existierende Modellierungsmethoden benötigten die Spezifikation spezieller Regeln für die Aktivierung und Deaktivierung von Features.

2.2 Zwischen Kompositionswelten

Während Parallelkompositionen $\|$ hauptsächlich in Beschreibungssprachen zur formalen Analyse verwendet werden, stellt die Superimposition $\bullet$ die Standardkomposition für feature-orientierte Softwareentwicklung dar. In der Dissertation werden beide Kompositionsoperatoren im Zusammenhang mit ACS betrachtet. Hierbei stellt sich die natürliche Frage, ob ACS mit unterschiedlichen Kompositionsoperatoren ineinander überführbar sind, d.h., in ein ACS mit gleicher Semantik $\phi(F) \bowtie \mathcal{V}$. Solche Transformationen hätten den Vorteil, feature-orientierte Programme der Softwareentwicklung in Programme zur



ϑ_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
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

Abb. 2: SIMULINK Geschwindigkeitsregler (links) und Fehlerschuldgrade der Komponenten (rechts)

formalen Analyse umzuwandeln und umgekehrt, Prototypen und Gerüste von konfigurierbarer Software mittels formaler Sprache zu entwickeln und zu verifizieren, bevor diese in feature-orientierte Software umgewandelt und verfeinert werden.

Theorem 2 *Für jedes $\|$-ACS gibt es ein exponentiell großes $\bullet$-ACS mit der gleichen Semantik. Umgekehrt gibt es $\bullet$-ACS für die es kein $\|$-ACS mit der gleichen Semantik gibt. Für jedes $\bullet$-ACS gibt es jedoch ein exponentiell großes $\|$-ACS mit den gleichem Verhalten.*

3 Zuverlässigkeitsanalyse in Redundanzsystemen

Wir haben in Abschnitt 2 gesehen, wie die familienbasierte Analyse von (re)konfigurierbaren Systemen auf Standardmethoden zurück geführt werden kann. Dies ermöglicht die Anwendung moderner Analysemethoden, w.z.B. Kosten-Nutzen-Analysen mittels PMC [BDK14] oder PMC für wissensintensive Systeme [DKT19]. Hierbei sei auch auf PROFEAT [Ch18], die Implementierung des in Abschnitt 2 vorgestellten Frameworks verwiesen, welche für $\|$-ACS die nötigen Transformationen automatisiert durchführt. Um die Vorteile des Frameworks zu demonstrieren, wurden in der Dissertation [Du21] verschiedenste Fallstudien u.a. an realen Systemen durchgeführt. Hierbei war auch der Fokus auf die familienbasierte quantitative Analyse von *Hardwaresystemen*. Während für Softwaresysteme die Vorteile einer familienbasierten Analyse bekannt waren, wurde Hardware bisher nur marginal untersucht. Neben der Parametersynthese und Analyse in rekonfigurierbaren Netzwerksystemen [BD18; DBK15; DKB14] wurden insbesondere SIMULINK-*Redundanzsysteme* eingeführt und untersucht [Du19b; Du20a; Du20b]. In solchen Systemen treten Komponenten redundant auf, um die Fehlertoleranz des Gesamtsystems zu erhöhen. Am Bekanntesten ist der *TMR-Mechanismus*, welcher die Ergebnisse von drei redundanten Komponenten mittels einer Majoritätsfunktion vergleicht. Aufgefasst als konfigurierbares System steht die Einführung von TMR für ein Feature. Die geringste Fehlerwahrscheinlichkeit kann natürlich erreicht werden, indem alle TMR Features aktiv sind. Jedoch führt die Einführung von TMR zu größeren Ausführungszeiten, höheren Energieverbrauch, sowie größeren Chips in Hardwareimplementierungen. Ein zentrales Ziel bei der Entwicklung von Redundanzsystemen ist es somit, unter gegebenen Kostenanforderungen jene Konfiguration zu ermitteln, welche

die höchste Zuverlässigkeit garantiert. In der Dissertation [Du21] wurde insbesondere eine Methode entwickelt, SimuLink Modelle in das feature-orientierte Framework von Abschnitt 2 zu übersetzen. Hierbei wurde auch ein Flugzeuggeschwindigkeitsregler, wie in Abb. 2 links dargestellt, und weitere Redundanzmechanismen untersucht. Fortschrittliches symbolisches PMC ermöglichte die Berechnung von Fehlerwahrscheinlichkeiten für mehr als 10^{10} Konfigurationen innerhalb von weniger als zwei Stunden auf einem handelsüblichen Rechner. Mit mehr als $2,5 \cdot 10^{13}$ Zuständen stellt dieses Modell auch eines der größten erfolgreich verifizierter realer Systeme dar und wurde in die MARS Benchmark Datenbank aufgenommen [Du20a].
Für die Konstruktion und Analyse des Reglermodells waren neuartige Reduktionstechniken nötig, welche in der Dissertation [Du21] eingeführt werden. *Iterative Variablenumordnung* [Du20a] ordnet Variablen in BDDs so um, dass sie Gemeinsamkeiten im Familienmodell besser komprimieren. Hierbei werden sukzessive Features zur Familie hinzugefügt, kombiniert mit klassischen Umordnungstechniken [Kl18]. Inspiriert von Techniken der Compileroptimierung verringern *Zurücksetzungs- und Registerallokationsreduktion* [Du20b] die Größe des Zustandsraums des Familienmodells.

4 Kausalität in konfigurierbaren Systemen

Die potenziell exponentielle Anzahl von Systemvarianten und der daraus folgenden großen Anzahl von Analyseergebnissen oder Fehlerberichten erfordert besondere Techniken für deren sinnvolle und praktikable Beschreibung. In der Dissertation [Du21] wurden hierzu grundlegende Konzepte und Methoden mittels *kausalem Schließen auf der Ebene von Features* präsentiert [Ba21; Du22]. Der neue Begriff der *Featuregründe* beschreibt hierbei jene Teilkonfigurationen von Features, die ein Grund für *Effekte* in Form von emergentem Systemverhalten sind. Dieser stützt sich auf die prominente kontrafaktische Definition von Kausalität von Halpern und Pearl [HP01]. Da Features meist auch eine intuitive Bedeutung den Systemfunktionalitäten zuordnen, können Erklärungen für Analyseergebnisse auf der Ebene von Features wichtige Erkenntnisse zur Systementwicklung geben [Ap13]. Effekte werden hierbei als Mengen von validen Konfigurationen beschrieben, wobei sowohl funktionale als auch quantitative Eigenschaften Effekte erzielen können. Im vorherigen Emailbeispiel könnte der Effekt “lange Verschlüsselungszeit” z.B. mit einer Menge $\mathsf{Eff}_{>5} \subseteq 2^F$ von validen Konfigurationen beschrieben werden, bei denen die erwartete Verschlüsselung länger als 5 Zeiteinheiten benötigt. Gemäß Abb. 1 ist $\mathsf{Eff}_{>5} = \big\{\{\mathsf{e},\mathsf{v},\mathsf{a}\}, \{\mathsf{e},\mathsf{v},\mathsf{r}\}, \{\mathsf{e},\mathsf{v},\mathsf{s},\mathsf{a}\}, \{\mathsf{e},\mathsf{v},\mathsf{s},\mathsf{r}\}\big\}$, da nur bei Caesar Verschlüsslung nicht mehr als 5 Zeiteinheiten benötigt werden.

Definition 3 *Ein* Featuregrund *für valide Konfigurationen* $\mathsf{Eff} \subseteq 2^F$ *ist ein Paar* $G = (G_0, G_1)$ *von inaktiven Features* $G_0 \subseteq F$ *und aktiven Features* $G_1 \subseteq F$*, sodass*

(1) $\varnothing \neq [\![G]\!] \subseteq \mathsf{Eff}$ *für die Menge* $[\![G]\!]$ *aller validen Konfigurationen bei denen alle Features aus* G_0 *nicht ausgewählt und alle Features aus* G_1 *ausgewählt sind, und*

(2) G minimal ist, d.h., Entfernen eines Features aus G_0 *oder* G_1 *führt zu* G' *mit* $[\![G']\!] \not\subseteq \mathsf{Eff}$.

Die erste Bedingung (1) formalisiert, dass alle vom Grund induzierten validen Konfigurationen auch den Effekt zeigen, während (2) die kontrafaktische Schlussfolgerung enthält: sollte eine vom Featuregrund gestellte Bedingung an ein Feature wegfallen, so ist es möglich, dass der Effekt nicht mehr eintritt. Im Emailbeispiel erhält man so drei Featuregründe für $\mathsf{Eff}_{>5}$: $(\varnothing, \{\mathsf{a}\})$, $(\varnothing, \{\mathsf{r}\})$, und $(\{\mathsf{c}\}, \{\mathsf{v}\})$. Die ersten beiden Gründe sind intuitiv und offensichtlich, denn AES oder RSA Verschlüsselung sind verantwortlich für lange Verschlüsselungszeiten. Der dritte Grund, der explizit eine Verschlüsslung fordert, aber die Caesar-Methode ausschließt, zeigt, dass manche Gründe nicht sofort offensichtlich sind aber sogar mehrere valide Konfigurationen abdecken können. In der Dissertation wird bewiesen, dass Featuregründe mit hinreichenden Primimplikanten der Effekt- und nicht-validen Konfigurationsmenge übereinstimmen, woraus sich direkt ein Algorithmus zu deren Berechnung ergibt.

Theorem 3 *Die Menge aller Featuregründe ist in polynomieller Zeit in der Anzahl der Systemkonfigurationen berechenbar.*

Da die Anzahl der Systemkonfigurationen jedoch exponentiell in der Anzahl der Features ist und es auch exponentiell viele Featuregründe geben kann, sind weitere Verfahren nützlich, die Featuregründe zur Erklärung von Effekten weiter verarbeiten. In der Dissertation werden neue Methoden zur Reduktion von aussagenlogischen Formeln für die Featuregrundmenge, als auch Effektüberdeckungen, *Verantwortlichkeiten* und *Schuldgrade* [CH04], sowie Featureinteraktionen vorgestellt. Zudem wurde mittels mehrerer Experimente aus dem Bereich der Analyse von konfigurierbaren Systemen gezeigt, dass Featuregründe beim Aufspüren von Fehlern und zum Erklären von Effekten nützlich sind.
Exemplarisch betrachten wir hier Schuldgrade, welche auch zur Erklärung der Verlässlichkeit des Reglers von Abb. 2 verwendet wurden. Die Verantwortlichkeit eines Features ist bezüglich einer gegebenen Effektkonfiguration, dem Kontext, definiert. Sie steht für den Anteil an Features, deren Belegung zusätzlich zu diesem Feature geändert werden müssen, um ein kontrafaktisches Beispiel zu erzeugen, d.h., den Effekt nicht mehr zu zeigen. Ein Wert von 1 steht somit für volle Verantwortlichkeit des Features, während ein Wert von 0 für keine Verantwortlichkeit steht. Schuldgrade nehmen eine globale Sicht auf Effektkonfigurationen ein und geben die zu erwartende Verantwortlichkeit eines Features wider. Die Tabelle in Abb. 2 listet die Schuldgrade für die Komponenten des in Abschnitt 3 analysierten Reglers für verschiedenste Zuverlässigkeitsschranken ϑ_R auf. So ist die Einführung von TMR für “Integrator” und “Acceleration”-Komponenten jeweils voll verantwortlich für den Effekt einer Fehlerwahrscheinlichkeit von weniger als $2,5\%$, was durch Standardmethoden der quantitative Analyse nicht direkt ableitbar ist. Solche Erkenntnisse können verwendet werden, um Faustregeln für die Konfiguration von Systemen abzuleiten. Im Reglerbeispiel sollten Systementwickler für geringe Fehlerwahrscheinlichkeiten TMR-Konfigurationen mit “Integrator” und “Acceleration”-Komponenten verwenden.

Literatur

[Ap13] Apel, S.; Batory, D. S.; Kästner, C.; Saake, G.: Feature-Oriented Software Product Lines - Concepts and Implementation. Springer, 2013.

[Ba21] Baier, C.; Dubslaff, C.; Funke, F.; Jantsch, S.; Majumdar, R.; Piribauer, J.; Ziemek, R.: From Verification to Causality-Based Explications. In: Proc. of the 48th International Colloquium on Automata, Languages, and Programming (ICALP). Bd. LIPIcs:198, Leibniz-Zentrum für Informatik, S. 1–20, 2021.

[BD18] Baier, C.; Dubslaff, C.: From Verification to Synthesis under Cost-Utility Constraints. ACM SIGLOG News 5/4, S. 26–46, 2018.

[BDK14] Baier, C.; Dubslaff, C.; Klüppelholz, S.: Trade-off Analysis Meets Probabilistic Model Checking. In: Proc. of the 23rd Conf. on Computer Sci. Logic and the 29th Symp. on Logic In Computer Sci. (CSL-LICS). ACM, S. 1–10, 2014.

[BK08] Baier, C.; Katoen, J.-P.: Principles of Model Checking. The MIT Press, 2008.

[CH04] Chockler, H.; Halpern, J. Y.: Responsibility and Blame: A Structural-Model Approach. Artificial Intelligence Research 22/, S. 93–115, 2004.

[Ch18] Chrszon, P.; Dubslaff, C.; Klüppelholz, S.; Baier, C.: ProFeat: feature-oriented engineering for family-based probabilistic model checking. Formal Aspects of Computing 30/, S. 45–75, 2018.

[Cl13] Classen, A.; Cordy, M.; Schobbens, P.-Y.; Heymans, P.; Legay, A.; Raskin, J.-F.: Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. Transactions on Software Engineering 39/, S. 1069–1089, 2013.

[DBK15] Dubslaff, C.; Baier, C.; Klüppelholz, S.: Probabilistic Model Checking for Feature-Oriented Systems. Transactions on Aspect-Oriented Software Development LNCS:8989/, S. 180–220, 2015.

[DKB14] Dubslaff, C.; Klüppelholz, S.; Baier, C.: Probabilistic Model Checking for Energy Analysis in Software Product Lines. In: Proc. of the 13th Conf. on Modularity (MODULARITY). ACM, S. 169–180, 2014.

[DKT19] Dubslaff, C.; Koopmann, P.; Turhan, A.-Y.: Ontology-Mediated Probabilistic Model Checking. In: Proceedings of the 15th Conference on integrated Formal Methods (iFM). Bd. LNCS:11918, Springer, S. 194–211, 2019.

[Du19a] Dubslaff, C.: Compositional Feature-Oriented Systems. In: Proceedings of the 17th Conference on Software Engineering and Formal Methods (SEFM). Bd. LNCS:12226, Springer, S. 162–180, 2019.

[Du19b] Dubslaff, C.; Ding, K.; Morozov, A.; Baier, C.; Janschek, K.: Breaking the Limits of Redundancy Systems Analysis. In: Proc. of the 29th European Safety and Reliability Conf. (ESREL). S. 2317–2325, 2019.

[Du20a] Dubslaff, C.; Morozov, A.; Baier, C.; Janschek, K.: Iterative Variable Reordering: Taming Huge System Families. In: Proc. of the 4th Workshop on Models for Formal Analysis of Real Systems (MARS). Bd. EPTCS:316, S. 121–133, 2020.

[Du20b] Dubslaff, C.; Morozov, A.; Baier, C.; Janschek, K.: Reduction Methods on Error-Propagation Graphs for Quantitative Systems Reliability Analysis. In: Proc. of the 30th European Safety and Reliability Conf. and 15th Probabilistic Safety Assessment and Management Conf. (ESREL-PSAM). 2020.

[Du21] Dubslaff, C.: Quantitative Analysis of Configurable and Reconfigurable Systems, Diss., TU Dresden, Institute for Theoretical Computer Science, 2021.

[Du22] Dubslaff, C.; Weis, K.; Baier, C.; Apel, S.: Causality in Configurable Software Systems. In: Proc. of the 44th Intern. Conf. on Software Engineering (ICSE). ACM, Pittsburgh, Pennsylvania, S. 325–337, 2022.

[HP01] Halpern, J. Y.; Pearl, J.: Causes and Explanations: A Structural-Model Approach - Part I: Causes. In: Proc. of the 17th Conf. in Uncertainty in Artificial Intelligence (UAI). Morgan Kaufmann, S. 194–202, 2001.

[KA08] Kästner, C.; Apel, S.: Integrating compositional and annotative approaches for product line engineering. In: Proc. GPCE Workshop on Modularization, Comp. and Generative Techniques for Product Line Engineering. S. 35–40, 2008.

[Ka90] Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, Techn. Ber., Carnegie-Mellon University Software Engineering Institute, 1990.

[Kl18] Klein, J.; Baier, C.; Chrszon, P.; Daum, M.; Dubslaff, C.; Klüppelholz, S.; Märcker, S.; Müller, D.: Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata. Software Tools for Technology Transfer 20/, S. 179–194, 2018.

[Th14] Thüm, T.; Apel, S.; Kästner, C.; Schaefer, I.; Saake, G.: A Classification and Survey of Analysis Strategies for Software Product Lines. Computing Surveys 47/, 6:1–6:45, 2014.

Clemens Dubslaff studierte Mathematik und Informatik an der TU Dresden, bevor er u.a. an der Neuen Universität Lissabon (Portugal) einen Master in “Computational Logic” ablegte. Nach einem Auslandsaufenthalt am NICTA in Sydney (Australien) kehrte er für seine Promotion an die TU Dresden zurück. Er ist nun an der TU Eindhoven (Niederlande) im Bereich der formalen Systemanalyse tätig. Seine wissenschaftliche Arbeit ist breit gefächert und umfasst sowohl rein theoretische, als auch interdisziplinäre und softwaretechnologische Beiträge.