Effiziente Interaktive und Automatische Produktlinienkonfiguration[1]

Sebastian Krieter[2]

Abstract: Moderne, hochgradig konfigurierbare Systeme umfassen eine enorme Anzahl von Varianten, die aus einer gemeinsamen Codebasis erstellt und auf spezifische Benutzeranforderungen zugeschnitten werden. Die Verwaltung und Benutzung dieser hohen Variabilität stellt sowohl die Benutzer als auch die Entwickler dieser Systeme vor Herausforderungen, da es für die meisten Systeme nicht möglich ist, alle möglichen Kombinationen von Konfigurationsoptionen zu testen oder auch nur zu berücksichtigen. In dieser Arbeit betrachten wir dieses Problem, indem wir Werkzeugunterstützung für automatische und halbautomatische Konfigurationsprozesse von konfigurierbaren Systemen untersuchen. Wir führen dazu neue Datenstrukturen, Algorithmen und Metriken ein, die sowohl die Effizienz als auch die Effektivität bisherige Ansätze deutlich steigern. Damit erleichtern wir den automatischen und halbautomatischen Konfigurationsprozess, insbesondere im Bezug auf das Testens und der Analyse konfigurierbarer Systeme.

1 Einleitung und Problemstellung

Konfigurierbare Systeme, wie zum Beispiel Software-Produktlinien, ermöglichen es den Benutzern, das Verhalten eines Softwaresystems an ihre besonderen Anforderungen anzupassen. Zu diesem Zweck umfasst ein konfigurierbares System eine gemeinsame Codebasis mit mehreren Konfigurationsoptionen (*Features*), die auf entsprechende Implementierungsartefakte abgebildet werden [Ap13; CE00]. Ein Benutzer kann eine *Konfiguration* bereitstellen, die den Zustand jedes Features definiert, um die gewünschte Variante des Systems zu erstellen [Ap13; CE00]. Moderne konfigurierbare Systeme können Tausende von Features umfassen, was zu einer enormen Anzahl von verschiedenen möglichen Varianten führt [Be10; Be13; HXC12; STS20]. Typischerweise weisen die Features komplexe Abhängigkeiten auf, die sich aus ihren, in einem *Feature-Modell* definierten, Beziehungen und den Interaktionen ihrer jeweiligen Implementierungsartefakte ergeben [Kn17; Na15]. In Abb. 1 zeigen wir ein Beispiel für ein Feature-Modell mit zehn Features, das insgesamt zehn Varianten beschreibt.

Die hohe Variabilität und die komplexen Abhängigkeiten stellen sowohl die Entwickler moderner konfigurierbarer Systeme als auch deren Benutzer vor Herausforderungen [BBR06; YL05]. Die Nutzer eines konfigurierbaren Systems müssen einen Konfigurationsprozess

[1] Englischer Titel der Dissertation: “Efficient Interactive and Automated Product-Line Configuration”

[2] Otto-von-Guericke-Universität Magdeburg, Arbeitsgruppe Datenbanken und Software Engineering, Universitätsplatz 2, 39106 Magdeburg, Deutschland sebastian.krieter@uni-ulm.de

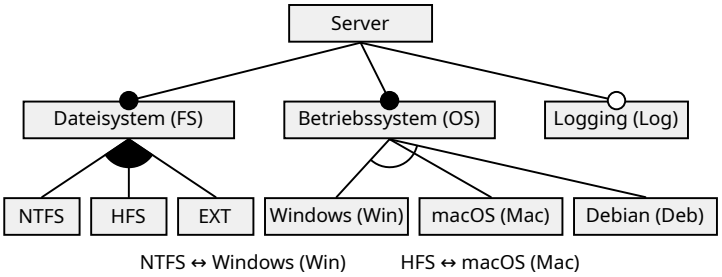


Abb. 1: Beispiel eines Feature-Modells für eine kleine Server-Produktlinie

durchführen, um eine für ihre Bedürfnisse geeignete Variante abzuleiten. Während dieses Prozesses müssen sie alle relevanten Abhängigkeiten zwischen den von ihnen gewählten Features berücksichtigen, um eine funktionierende Systemvariante abzuleiten. So können sich beispielsweise einige Features gegenseitig ausschließen oder voneinander abhängig sein. Dies kann den Konfigurationsprozess sehr mühsam und fehleranfällig machen, da einige Kombinationen von Features unmöglich sind oder unerwünschte Interaktionen auslösen können [BBR06]. Für Entwickler solcher Systeme ist dieses Problem noch komplexer, da sie sicherstellen müssen, dass sich jede konfigurierbare Variante entsprechend ihrer Spezifikationen verhält und sie keine unvorhergesehenen Interaktionen oder Fehler enthält. Eine einfache Überprüfung aller möglichen Varianten ist aufgrund der enormen Anzahl von Konfigurationen eines Systems meist nicht durchführbar. Die Lösung dieser Probleme erfordert automatisierte Analysen und Werkzeugunterstützung während der Entwicklung und des Konfigurationsprozesses.

Mit Hilfe von Werkzeugen kann ein halbautomatischer Konfigurationsprozess ermöglicht werden, der Benutzern bei ihren Entscheidungsfindungen unterstützt. So kann unter Anderem das Risiko ungültige Konfigurationen zu erstellen vermindert und der manuellen Aufwand für die Benutzer verringert werden. Ein halbautomatischer Konfigurationsprozess unterstützt Entwickler auch bei der Analyse der Variabilität eines Systems, dem Testen von Varianten und der Optimierung von Parametern. In der Arbeit, betrachten wir dabei vor allem *Decision-Propagation* [Ba05]. Diese Technik berechnet zu einer gegeben Auswahl oder Abwahl eines Features in einem Konfigurationsprozess (Decision) die Menge der Features, die durch diese *Decision* aus- oder abgewählt werden müssen, um am Ende des Prozesses eine gültige Konfiguration zu erhalten. Werkzeugunterstützung kann auch einen vollautomatischen Konfigurationsprozess ermöglichen, bei dem die Konfigurationen vollständig von einem Algorithmus auf der Grundlage einer bestimmten Eingabe generiert werden. So kann ein Algorithmus beispielsweise zufällige Konfigurationen (z.B. zu Testzwecken), Konfigurationen, die ein bestimmtes Coverage-Kriterium erfüllen (z.B. *statement* oder *T-Wise Interaction Coverage*) oder Konfigurationen, die zuvor erstellten Konfigurationen ähneln (z.B. zur Parameteroptimierung) erzeugen [Oh17].

In der Arbeit betrachten wir insbesondere die Verbesserung von *Sampling-Algorithmen*

mit dem Schwerpunkt auf Erstellung von Konfigurationen zu Testzwecken. Sampling-Algorithmen versuchen das Problem der hohen Variantenanzahl zu umgehen, indem sie eine kleine Liste von Konfigurationen erstellen (d.h. ein *Sample*), die den gültigen Variantenraum repräsentiert. Es gibt eine Reihe verschiedener Sampling-Strategien zur Erzeugung repräsentativer Samples [Va18]. Eine vielversprechende Sampling-Technik ist das *T-Wise Interaction Sampling*, das darauf abzielt, ein minimales Sample zu erzeugen, das alle möglichen Interaktionen von t Features abdeckt (z.B. alle ausgewählt, alle abgewählt, genau eins ausgewählt usw.).

Gerade für moderne konfigurierbare Systeme mit einer hohen Anzahl von Features stellen die oben genannten Werkzeuge eine notwendige Unterstützung für Entwickler und Benutzer dar. Allerdings ist gerade bei diesen Systemen die Anwendung von existieren Techniken aufgrund der enorme Anzahl von möglichen Konfigurationen schwierig. Decision-Propagation bietet zwar eine hilfreiche Unterstützung in einem interaktiven Konfigurationsprozess, ist aber auch ein rechenintensiver Algorithmus, da das zugrundeliegende Problem, gültige Belegungen für Variablen in einer Booleschen Formel zu finden (Erfüllbarkeitsproblem), NP-vollständig ist [Co71]. Aus diesem Grund versuchen wir, die Leistung von Decision-Propagation durch neuartige Datenstrukturen zu verbessern.

Ein vollautomatischer Konfigurationsprozess mittels Sampling ist eine wertvolle Technik zum Testen konfigurierbarer Systeme. Selbst für kleine Werte von t (d.h. $t \leq 3$) erzeugt T-Wise Interaction Sampling ein effektives Sample für das Testen. Bei hochgradig konfigurierbaren Systemen mit Tausenden von Features kann das Sampling jedoch selbst bei kleinen Werten von t eine unzumutbare Berechnungszeit in Anspruch nehmen und Sample mit einer zu großen Anzahl von Konfigurationen erzeugen. Daher untersuchen wir die Steigerung der Effizienz von T-Wise Interaction Sampling-Algorithmen durch die Verbesserung ihrer allgemeinen Leistung und die Einführung zusätzlicher Parameter zur Feinabstimmung eines Sampling-Prozesses für ein bestimmtes konfigurierbares System.

Ein weiteres Problem bei der Verwendung des T-Wise Interaction Sampling besteht darin, dass es ausschließlich mit den Features arbeitet, die durch das Feature-Modell eines konfigurierbaren Systems definiert sind. Das bedeutet, dass reguläres T-Wise Interaction Sampling die Implementierungsartefakte eines Systems, wie Quellcode, Modelle und Testfälle, nicht berücksichtigt. Dies kann zu unnötig großen Samples und einem unnötig hohen Berechnungsaufwand führen. Darüber hinaus können die resultierenden Samples die tatsächlichen Interaktionen der Implementierungsartefakte nicht genau wiedergeben und können dadurch weniger effektiv für das Testen eines Systems sein. Wir versuchen diese Probleme zu lösen, indem wir ein neues Coverage-Kriterium für das einen Sampling einführen, um effizientere und effektivere Samples erstellen zu können.

2 Hauptbeiträge

Das Hauptziel der Arbeit ist die Verbesserung der Werkzeugunterstützung für den Konfigurationsprozess von konfigurierbaren Systemen. Zu diesem Zweck führen wir drei neue Techniken ein, um den halb- und vollautomatischen Konfigurationsprozess zu erleichtern [Kr22]. Erstens führen wir *modale Implikationsgraphen (MIGs)* ein, um den (halb)automatischen Konfigurationsprozess zu unterstützen. MIGs sind eine Erweiterung von regulären Implikationsgraphen und ermöglichen die einfache Identifizierung paarweiser Beziehungen zwischen Features, was den Rechenaufwand zur Bestimmung der Gültigkeit einer Konfiguration stark reduzieren kann. Zur Anwendung von MIGs stellen wir einen Algorithmus für *Decision-Propagation* innerhalb eines Interaktiven Konfigurationsprozesses vor, der MIGs verwendet. Zweitens stellen wir einen neuen Sampling-Algorithmus zur effizienten Erzeugung von Samples für T-Wise Interaction Coverage vor. Unser neuer Sampling-Algorithmus *YASA* zielt darauf ab, die Sampling-Zeit und die Sample-Größe im Vergleich zu aktuellen State-of-the-Art-Algorithmen durch die Verwendung von MIGs und anderen Optimierungen zu reduzieren. Außerdem ermöglichen wir die Anpassung von YASA durch Laufzeitparameter und eine modulare Architektur. Dies führt dazu, dass der Nutzer mehr Kontrolle über die Sampling-Zeit, die Sample-Größe, sowie über andere Eigenschaften des erstellten Samples hat (z.B. über die Ähnlichkeiten zwischen Konfigurationen innerhalb eines Samples) und somit in der Lage ist, das Sample für ein bestimmtes System abzustimmen. Drittens, schlagen wir ein neues Coverage-Kriterium (*T-Wise Presence Condition Coverage*) vor, das eine bessere Abschätzung des Potenzial eines Samples zur Aufdeckung von Softwarefehlern liefern soll als das bisher genutzte T-Wise Interaction Coverage. Anstatt nur die Interaktionen zwischen Features zu betrachten, berücksichtigt unser neues Kriterium die Interaktionen zwischen den konkreten Implementierungsartefakten. Wir erweitern YASA, so dass für jedes System ein Sample mit einer 100% T-Wise Presence Condition Coverage erzeugt werden kann.

Wir stellen fünf Forschungsfragen auf, anhand derer wir die drei Hauptbeiträge unserer Arbeit evaluieren:

- RQ_1 Kann die Effizienz der Konfigurationsgenerierung in einem interaktiven und automatisierten Konfigurationsprozess mithilfe von MIGs erhöht werden?
- RQ_2 Kann den initialen Rechenaufwand zur Erstellung eines MIGs durch einen optimierten Erstellungsprozess verringert werden?
- RQ_3 Kann die Effizienz der Konfigurationserstellung in einem automatisierten Konfigurationsprozess mit YASA erhöht werden?
- RQ_4 Kann die Sample-Größe und die Sampling-Zeit von YASA durch Anpassung der Parameter sinnvoll beeinflusst werden?
- RQ_5 Kann die Testeffizienz und -effektivität durch den Einsatz von T-Wise Presence Condition Coverage erhöht werden?

2.1 Modale Implikationsgraphen

Viele der relevanten Analysen für Feature-Modelle, wie z.B. das Berechnen von Decision-Propagation sind NP-schwere Probleme, was bedeutet, dass derzeit kein Verfahren bekannt ist, das diese effizient lösen kann [Co71]. Dennoch versuchen wir, Analysen von Feature-Modellen in vertretbarer Zeit durchzuführen, indem wir uns zwei ihrer Eigenschaften zunutze machen. Erstens können die meisten Feature-Modell-Analysen gelöst werden, indem man sie auf eine oder mehrere Instanzen des *Erfüllbarkeitsproblems (SAT)* oder anderer bekannter Probleme, wie das *Constraint Satisfaction Problem (CSP)* oder das Zählproblem zum Erfüllbarkeitsproblem (#SAT) reduziert [Be07; Me09]. Zweitens sind die durch die Reduktion entstehenden SAT-Instanzen für viele reale Feature-Modelle relativ einfach zu lösen [MWC09]. Dennoch sind die meisten SAT-basierten Analysen aufwendig zu berechnen, da sie auf mehrere SAT-Instanzen reduziert werden müssen. Eine mögliche Effizienzverbesserung für diese Analysen liegt daher in der Verringerung der Anzahl der SAT-Instanzen, die zur Berechnung erforderlich sind.

Durch die Modellierung der Beziehungen zwischen Features innerhalb eines *Modalen Implikationsgraphen (MIG)* versuchen wir die Anzahl von notwendigen SAT-Instanzen innerhalb von SAT-basierten Analysen zu reduzieren. MIGs sind eine Erweiterung von regulären Implikationsgraphen und ermöglichen die einfache Identifizierung paarweiser Beziehungen zwischen Features. Pro Feature enthält ein MIG zwei Knoten, die die An- und Abwahl des Features in einer Konfiguration repräsentieren. Impliziert die An- oder Abwahl eines Feature die An- oder Abwahl eines anderen Features, so werden die entsprechenden Knoten durch eine *starke* Kante verbunden. Ist die Implikation abhängig von dem Zustand anderer Feature so werden die Knoten mit einer *schwachen* Kante verbunden. Abb. 2 zeigt einen MIG für das Feature-Modell in Abb. 1. Um einen MIG für ein beliebiges Feature-Modell zu konstruieren, haben wir einen vollständigen und einen optimierten Build-Prozess entwickelt, die einen vollständigen bzw. einen unvollständigen MIG erstellen. Weiterhin haben wir einen inkrementellen Build-Prozess entwickelt, um einen bestehenden MIG nach einer Änderung am Feature-Modell zu aktualisieren.

Zur Anwendung von MIGs stellen wir einen Algorithmus für *Decision-Propagation* innerhalb eines interaktiven Konfigurationsprozesses vor, der MIGs verwendet. Durch das Traversieren eines MIG können ausgehend von einem Startknoten alle direkt implizierten, potentiell implizierten und nicht erreichbaren Knoten identifiziert werden. Dadurch müssen nur noch SAT-Instanzen für die Menge der potentiell implizierten Knoten gelöst werden.

Als Ergebnis unserer Evaluierung von MIGs und ihres Build-Prozesses haben wir einige wichtige Erkenntnisse über das Konzept gewonnen. (1) Die Konstruktion eines MIG erfordert einen gewissen zusätzlichen Zeitaufwand, der von dem jeweiligen Build-Prozess abhängt. Mit dem optimierten Build-Prozess kann ein unvollständiger und nicht-minimaler MIG effizient erstellt werden. Der vollständige Build-Prozess erstellt einen vollständigen und minimalen MIG, erfordert jedoch deutlich mehr Zeit. Der inkrementelle Build-Prozess ist wesentlich schneller als der vollständige Build-Prozess und gleichauf mit dem optimierten.

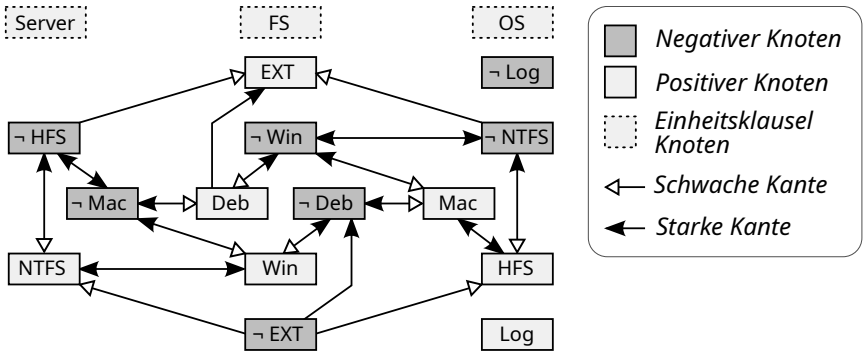


Abb. 2: Modaler Implikationsgraph für die Server-Produktlinie

Allerdings kann der Grad der Minimalität und Vollständigkeit eines inkrementell erstellten MIGs höher sein als bei einem MIG, der mit dem optimierten Build-Prozess erstellt wurde. (2) Die Verwendung eines MIGs kann die Ausführungszeit von Decision-Propagation erheblich verringern. Im Vergleich zur regulären SAT-basierten Decision-Propagation können wir hohe Effizienzsteigerung feststellen. Dies gilt sogar für unvollständige und nicht minimale MIGs, die durch den optimierten bzw. den inkrementellen Build-Prozess erstellt wurden. (3) Der Unterschied in der Effizienz von Decision-Propagation eines minimalen und vollständigen MIG im Vergleich zu einem nicht-minimalen und unvollständigen MIG relativ gering. Die Vollständigkeit eines MIGs kann die Leistung einiger Analysen erheblich steigern, aber für Decision-Propagation ist sogar ein unvollständiger MIG ausreichend. (4) Für die meisten Szenarien die Verwendung eines unvollständigen MIGs, der aus unserem optimierten oder inkrementellen Build-Prozess resultiert, am besten geeignet ist. Die Verwendung eines reinen SAT-basierten Ansatzes ohne MIG ist nur dann geeignet, wenn Decision-Propagation nur wenige Male für ein Feature-Modell durchgeführt wird. Im Gegensatz dazu ist die Verwendung eines vollständigen und minimalen MIGs nur dann geeignet, wenn die Decision-Propagation viele Male verwendet wird oder wenn die anfängliche Erstellungszeit irrelevant ist. Für die meisten Szenarien scheint also ein unvollständiger MIG den besten Kompromiss zwischen der für seine Erstellung benötigten Zeit und der während der Analyse eingesparten Zeit zu bieten.

2.2 YASA

Mit *YASA (Yet Another Sampling Algorithm)* stellen wir einen effizienten und flexiblen Algorithmus für T-Wise Interaction Sampling vor. Das Design von YASA soll möglichst schnell kleine Samples erzeugen, die an bestimmte Nutzeranforderungen angepasst werden können. Zu diesem Zweck bietet YASA eine Reihe von Parametern, wie z.B. Gruppierung von Features, Anzahl an Resamplings und Angabe eines initialen Samples, die dem Benutzer mehr Kontrolle über die Sampling-Zeit, die Sampling-Größe und andere Eigenschaften des

Samples geben. Außerdem besteht YASA aus einer modularen Architektur, die es ermöglicht, bestimmte Strategien innerhalb des Algorithmus zu ersetzen, um die Eigenschaften des berechneten Samples weiter zu beeinflussen.

Die Evaluierung von YASA zeigt einige wichtige Erkenntnisse auf. (1) YASA ist effizienter oder mindestens genauso effizient wie andere Algorithmen, die dem Stand der Technik entsprechen. Das Ausmaß der Verringerung der Sampling-Zeit im Vergleich zu anderen Algorithmen hängt von den Parametereinstellungen von YASA ab. Der Parameter für die Anzahl von Resamplings korreliert invers linear zur Sampling-Zeit. Ein niedriger Wert beschleunigt die Erstellung eines Samples, kann aber auch die resultierende Sample-Größe erhöhen. Wird die Anzahl der Features, die von YASA betrachtet werden sollen verringert, so verringert sich auch die Anzahl der in einem Sample berücksichtigten Interaktionen, so dass sich die Sampling-Zeit erheblich verkürzt. Allerdings verringert sich dadurch auch die erreichte Coverage, was nur in einigen Anwendungsfällen akzeptabel sein kann. (2) Die von YASA erzeugten Samples sind im Vergleich zu anderen Algorithmen für T-Wise Interaction Sampling meist kleiner. Die Sampling-Größe ist ebenfalls von den Parametereinstellungen von YASA abhängig. Die Wahl eines hohen Wertes für die Anzahl an Resamplings kann die Sampling-Größe verringern, während gleichzeitig die Sampling-Zeit erhöht wird. (3) Die Nutzer von YASA haben mehr Kontrolle über Sampling-Zeit, Sampling-Größe und Coverage als bei anderen Algorithmen. Durch die Wahl spezifischer Parametereinstellungen können Nutzer das Verhalten von YASA an das konkrete Anwendungsszenario anpassen. (4) Es gibt derzeit eine Grenze für die Skalierbarkeit von YASA hinsichtlich der Sampling-Zeit. Für Feature-Modelle mit mehr als 10.000 Features und einem t-Wert von ≥ 2 kann die Sampling-Zeit von YASA für viele Anwendungen nicht praktikabel sein. Dies kann jedoch durch eine geeignete Parameterwahl gemildert werden. Wird zum Beispiel die Menge der betrachteten Features auf der Grundlage von Domänenwissen eines konfigurierbaren Systems beschränkt, könnte dies eine praktikable Lösung für die Erstellung von Samples auch für große Feature-Modelle sein.

2.3 T-Wise Presence Condition Coverage

Um die Testeffektivität und -effizienz für konfigurierbare Systeme zu verbessern haben wir das Coverage-Kriterium *T-Wise Presence Condition Coverage* eingeführt. Im Gegensatz zum interaction coverage betrachtet presence condition coverage den Lösungsraum eines konfigurierbaren Systems, indem es die Bedingungen unter denen Implementierungsartefakte in einer Variante enthalten sind (presence conditions) berücksichtigt. Dieses Coverage-Kriterium kann verwendet werden, um festzustellen in welchem Maße ein gegebenes Sample die Kombinationen konkreter Implementierungsartefakte abdeckt und damit das Fehlererkennungspotenzial des Samples besser abschätzen. Darüber hinaus haben wir unseren Algorithmus YASA so erweitert, dass er Samples mit einem T-Wise Presence Condition Coverage von 100% erzeugen kann.

Nach der Evaluierung des neuen Kriteriums und unserer Erweiterungen zu YASA erlangen wir zu einigen wichtigen Erkenntnisse. (1) Samples mit einer 100%igen T-Wise Presence Condition Coverage sind in der Lage, mehr Fehler zu erkennen als Samples mit einer 100%igen T-Wise Interaction Coverage für den gleichen Wert für t. Somit scheint T-Wise Presence Condition Coverage ein besserer Indikator für die Effektivität eines Samples bei der Erkennung von Fehlern während des Testens zu sein. (2) YASA ist in der Lage, ein Sample mit einem 100%igen T-Wise Presence Condition Coverage zu erzeugen. Andere Algorithmen erreichen im Durchschnitt nur 97%. Daher decken die Samples dieser Algorithmen einige Interaktionen zwischen Implementierungsartefakten nicht ab, die dann nicht getestet werden können. (3) Die von YASA erzeugten Samples für die T-Wise Presence Condition Coverage sind im Durchschnitt kleiner als die Samples von Algorithmen für T-Wise Interaction Coverage. Somit führen die durch T-Wise Presence Condition Sampling erzeugten Samples zu einer erhöhten Testeffizienz. (4) Die Sampling-Zeit von YASA ist für T-Wise Presence Condition Coverage vergleichbar schnell oder schneller als T-Wise Interaction Coverage mit YASA. Somit ist die Sampling-Effizienz für das T-Wise Presence Condition Sampling etwa gleich hoch wie das für reguläres T-Wise Interaction Sampling.

3 Fazit

Mit den oben beschriebenen Erkenntnissen sind wir in der Lage, unsere Eingangs gestellten Forschungsfragen zu beantworten. RQ_1 MIGs sind gut geeignet, um Decision-Propagation im halbautomatischen Konfigurationsprozess zu beschleunigen. Indem sie die Effizienz von Decision-Propagation erhöhen, erleichtern sie den gesamten interaktiven Konfigurationsprozess eines Nutzers. Darüber hinaus lassen sich MIGs auch in unserem Sampling-Algorithmus YASA einsetzen. So können wir die MIGs auch zur Steigerung der Effizienz des vollautomatischen Konfigurationsprozesses nutzen.

RQ_2 Während die Erstellung eines funktionierenden, aber unvollständigen und nicht minimalen MIGs nur einen vernachlässigbaren Rechenaufwand erfordert, ist die Erstellung eines vollständigen und minimalen MIGs wesentlich rechenintensiver. Andererseits verbessert die Verwendung eines vollständigen MIGs dessen Wirksamkeit im Vergleich zu einer unvollständigen Version nur geringfügig. Wir können also die Effektivität und die Erstellungszeit eines MIGs durch die Wahl eines geeigneten Build-Prozesses steuern. Der Kompromiss zwischen zusätzlicher Erstellungszeit und Effektivität ist jedoch stark zugunsten der Erstellung unvollständiger MIGs verzerrt, die etwas weniger effektiv sind. In den meisten Szenarien kann ein MIG den Konfigurationsprozess unterstützen und gleichzeitig schnell erstellt und aktualisiert werden. Nur in bestimmten Szenarien ist die Erstellung eines vollständigen MIGs den erforderlichen Rechenaufwand wert.

RQ_3 Mit YASA können wir Konfigurationen effizienter erzeugen als mit anderen Sampling-Algorithmen, die dem Stand der Technik entsprechen. Dies gilt für die Berechnung von Samples mit gleicher Coverage wie bei anderen Sampling-Algorithmen. Wenn wir jedoch die Coverage durch bestimmte Parametereinstellungen verringern, können wir Samples

noch schneller berechnen. So gelingt es uns, den vollautomatischen Konfigurationsprozess mit YASA zu erleichtern.

RQ4 YASA ist in der Lage, kleinere Samples zu produzieren und auch schneller zu sein als existierende Sampling-Algorithmen, allerdings nicht immer gleichzeitig. Das Verhalten von YASA kann durch seine Parametereinstellungen gesteuert werden, die einen Kompromiss zwischen Sampling-Größe und Sampling-Zeit ermöglichen. Insbesondere der Parameter für die Anzahl von Resamplings hat einen großen Einfluss auf beide Eigenschaften. Ein hoher Wert für diesen Parameter erzeugt kleinere Samples, erhöht aber auch die Sampling-Zeit und umgekehrt. Damit ermöglichen wir Nutzern, die Sampling-Zeit, Sampling-Größe und Coverage eines Samples zu durch entsprechende Parametereinstellungen zu kontrollieren.

RQ5 Mit T-Wise Presence Condition Coverage sind wir in der Lage, kleine Samples zu erstellen, die in Bezug auf ihr Fehlererkennungspotenzial effektiver sind als Samples für T-Wise Interaction Coverage. Die Sampling-Zeit ist dabei gleich oder schneller als bei der Erstellung von Samples für T-Wise Interaction Coverage. Somit sind wir in der Lage, sowohl die Testeffizienz als auch die Testeffektivität für konfigurierbare Systeme zu erhöhen, indem wir T-Wise Presence Condition Sampling verwenden.

Literatur

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

[Ba05] Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: SPLC. Springer, S. 7–20, 2005.

[BBR06] Batory, D.; Benavides, D.; Ruiz-Cortés, A.: Automated Analyses of Feature Models: Challenges Ahead. Comm. ACM 49/12, S. 45–47, 2006.

[Be07] Benavides, D.: On the Automated Analysis of Software Product Lines Using Feature Models - A Framework for Developing Automated Tool Support, Diss., University of Seville, 2007.

[Be10] Berger, T.; She, S.; Lotufo, R.; Wąsowski, A.; Czarnecki, K.: Variability Modeling in the Real: A Perspective from the Operating Systems Domain. In: ASE. ACM, S. 73–82, 2010.

[Be13] Berger, T.; Rublack, R.; Nair, D.; Atlee, J. M.; Becker, M.; Czarnecki, K.; Wąsowski, A.: A Survey of Variability Modeling in Industrial Practice. In: VaMoS. ACM, 7:1–7:8, 2013.

[CE00] Czarnecki, K.; Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.

[Co71] Cook, S. A.: The Complexity of Theorem-Proving Procedures. In: STOC. ACM, 1971.

[HXC12] Hubaux, A.; Xiong, Y.; Czarnecki, K.: A User Survey of Configuration Challenges in Linux and eCos. In: VaMoS. ACM, S. 149–155, 2012.

[Kn17] Knüppel, A.; Thüm, T.; Mennicke, S.; Meinicke, J.; Schaefer, I.: Is There a Mismatch Between Real-World Feature Models and Product-Line Research? In: ESEC/FSE. ACM, S. 291–302, 2017.

[Kr22] Krieter, S.: Efficient interactive and automated product-line configuration, Diss., Otto-von-Guericke University Magdeburg, Germany, 2022.

[Me09] Mendonça, M.: Efficient Reasoning Techniques for Large Scale Feature Models, Diss., University of Waterloo, 2009.

[MWC09] Mendonça, M.; Wąsowski, A.; Czarnecki, K.: SAT-Based Analysis of Feature Models is Easy. In: SPLC. Software Engineering Institute, S. 231–240, 2009.

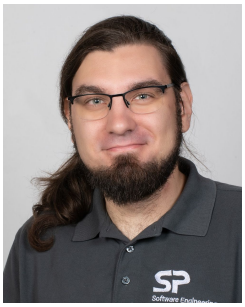
[Na15] Nadi, S.; Berger, T.; Kästner, C.; Czarnecki, K.: Where Do Configuration Constraints Stem From? An Extraction Approach and an Empirical Study. TSE 41/8, S. 820–841, 2015.

[Oh17] Oh, J.; Batory, D.; Myers, M.; Siegmund, N.: Finding Near-Optimal Configurations in Product Lines by Random Sampling. In: FSE. S. 61–71, 2017.

[STS20] Sundermann, C.; Thüm, T.; Schaefer, I.: Evaluating #SAT Solvers on Industrial Feature Models. In: VaMoS. ACM, 3:1–3:9, 2020.

[Va18] Varshosaz, M.; Al-Hajjaji, M.; Thüm, T.; Runge, T.; Mousavi, M. R.; Schaefer, I.: A Classification of Product Sampling for Software Product Lines. In: SPLC. ACM, S. 1–13, 2018.

[YL05] Ye, H.; Liu, H.: Approach to Modelling Feature Variability and Dependencies in Software Product Lines. IEE Proceedings - Software 152/3, S. 101–109, 2005.



Sebastian Krieter wurde am 18. Oktober 1990 in Magdeburg geboren. Er absolvierte dort im Jahr 2009 sein Abitur am Werner-von-Siemens-Gymnasium. Von 2010 bis 2015 studierte er Informatik an der Otto-von-Guericke-Universität Magdeburg. Ebenda promovierte er zwischen 2016 und 2022 unter Prof. Dr. Gunter Saake in der Arbeitsgruppe Datenbanken und Software Engineering. Zur Zeit seiner Promotion war er ebenfalls an der Hochschule Harz in Wernigerode im Fachbereich Automatisierung und Informatik bei Prof. Dr. Thomas Leich als wissenschaftlicher Mitarbeiter und Lehrender tätig. Nach seiner Promotion arbeitet er als Postdoktorand an der Universität Ulm im Institut für Softwaretechnik und Programmiersprachen mit dem Fokus auf Forschung im Bereich Testen und Analyse von Produktlinien.