Methoden

Marco Grochowski, Hendrik Simon, Dimitri Bohlender, Stefan Kowalewski, Andreas Löcklin*, Timo Müller, Nasser Jazdi, Andreas Zeller und Michael Weyrich

Formale Methoden für rekonfigurierbare cyber-physische Systeme in der Produktion

Formal methods for reconfigurable cyber-physical systems in production

https://doi.org/10.1515/auto-2019-0115
Empfangen 20. September 2019; angenommen 22. November 2019

Zusammenfassung: Durch zunehmende Agilität im Entwicklungsprozess, kürzere Lebenszyklen und sich ändernde Kunden- und Gesetzgeberanforderungen müssen Produktionssysteme wandlungsfähig sein. Aber jede Veränderung des Systemverhaltens muss anschließend auch erneut abgesichert werden. Zur Absicherung sicherheitskritischer Funktionen eignen sich formale Verifikationsmethoden. Allerdings ist hierzu ein hoher Modellierungsaufwand notwendig, was den Einsatz formaler Verifikationsmethoden in der Praxis hemmt. Zudem empfiehlt sich eine Überprüfung der zur formalen Verifikation eingesetzten Modelle. Zur Bewältigung der Herausforderungen für die Absicherung, die eine Rekonfiguration während des Betriebs nach sich zieht, werden in diesem Beitrag zwei vielversprechende und sich gegenseitig ergänzende Ansätze für die lebenszyklusübergreifende Absicherung von Produktionssystemen vorgestellt.

Schlagwörter: Formale Verifikation, CPS, CPPS, Rekonfiguration

***Korrespondenzautor: Andreas Löcklin,** Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Deutschland, E-Mail: andreas.loecklin@ias.uni-stuttgart.de
Marco Grochowski, Hendrik Simon, Dimitri Bohlender, Stefan Kowalewski, Lehrstuhl Informatik 11 – Embedded Software, RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Deutschland, E-Mails: grochowski@embedded.rwth-aachen.de, simon@embedded.rwth-aachen.de, bohlender@embedded.rwth-aachen.de, kowalewski@embedded.rwth-aachen.de
Timo Müller, Nasser Jazdi, Andreas Zeller, Michael Weyrich, Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Deutschland, E-Mails: timo.mueller@ias.uni-stuttgart.de, nasser.jazdi@ias.uni-stuttgart.de, andreas.zeller@ias.uni-stuttgart.de, michael.weyrich@ias.uni-stuttgart.de

Abstract: Due to the increasing agility in the development process, shorter life cycles, and changing customer and legislator requirements, production systems must be adaptable. In addition to the ability to react flexibly to changing production requirements, the behavior of the production system must be reassured after a change. Formal verification methods are suitable for safeguarding safety-critical functions. However, this requires a high modeling effort, which inhibits the use of formal verification methods in practice. For reliable verification results, the quality and correctness of the models is decisive. Therefore, it is reasonable to analyze the underlying models. This paper presents two promising and mutually complementing approaches for the lifecycle-spanning safeguarding of production systems to overcome the challenges for the verification that a reconfiguration of the production systems brings with it during operation.

Keywords: formal verification, CPS, CPPS, reconfiguration

1 Einleitung

Aus der erhöhten Volatilität der Märkte, kürzeren Innovations- und Produktlebenszyklen und dem enormen Anstieg der geforderten Variantenvielfalt steigt der Bedarf an Produktionssystemen, welche diesen Trends gerecht werden können [16]. Die Identifikation aller relevanten Anforderungen an ein Produktionssystem zur Entwicklungszeit wird durch die Steigerung der Anzahl der über den Lebenszyklus hinweg hergestellten Produktvarianten stark erschwert. Daher werden zukünftig Änderungen an Produktionssystemen zur Betriebszeit nicht die Ausnahme, sondern die Regel sein [24].

In dieser Veröffentlichung wird die Absicherung solcher Änderungen behandelt. Die vorgestellten Ansätze behandeln hierbei den Einsatz modellbasierter, formaler Verifikationsmethoden über den gesamten Lebenszyklus eines Produktionssystems. Des Weiteren wird die Überprü-

fung der zur formalen Verifikation eingesetzten Modelle behandelt. Hierbei ergänzen sich die vorgestellten Ansätze, da die modellbasiert erhaltenen Ergebnisse der formalen Verifikationsverfahren von der Qualität der verwendeten Modelle abhängen.

Der weitere Inhalt dieser Veröffentlichung gliedert sich folgendermaßen. Zunächst erfolgt in Abschnitt 2 eine kurze Einführung in das betrachtete Szenario der Rekonfiguration von wandlungsfähigen Produktionssystemen sowie in die Methoden der formalen Verifikation. Der anschließend in Abschnitt 3 vorgestellte Ansatz TestIAS verbessert die Effizienz der formalen Verifikation von rekonfigurierten CPPS. Demgegenüber wird in Abschnitt 4 der Ansatz Arcade.PLC vorgestellt, der die automatische Verknüpfung zwischen formalen Verifikationsverfahren und ausführbaren Testfällen ermöglicht. Dadurch kann eine zusätzliche vollautomatische Überprüfung der Ergebnisse der formalen Verifikation erreicht werden. Abschließend werden in Abschnitt 5 die vorgestellten Inhalte zusammengefasst und ein Ausblick für die lebenszyklusübergreifende Absicherung von CPPS gegeben.

2 Rekonfiguration von CPPS und formale Verifikationsmethoden

Ein cyber-physisches Produktionssystem (CPPS) ist gemäß [1] ein „CPS, das in der Produktion eingesetzt wird“. Basierend auf [13] sind die Kernaspekte von cyber-physischen Systemen (CPS) neben ihren physischen Komponenten ihre Vernetzung sowie ihre Fähigkeit zur Informationsverarbeitung. Insbesondere können CPS dadurch einen gewissen Grad an Intelligenz innehaben. Im Vergleich zu klassischen automatisierten Produktionssystemen verfügen die in der Produktion eingesetzten CPS über erweiterte kognitive Fähigkeiten. Hierzu zählen gemäß [30] die Fähigkeit zur automatischen Integration neuer intelligenter Komponenten, die Fähigkeit zur automatischen Bewertung und Verbesserung der Fertigungsplanung sowie die Fähigkeit der Wissensbildung und Argumentation. Diese CPPS stellen gemäß [7, 17, 20] eine Kerntechnologie zur zukünftigen Realisierung von hoch wandlungsfähigen und über einen großen Flexibilitätskorridor verfügenden Produktionssystemen dar.

Wandlungsfähige Produktionssysteme zeichnen sich durch eine einfache Anpassbarkeit an neue Produkte oder Prozesse aus [15]. Anpassungen können entweder innerhalb des Flexibilitätskorridors eines Produktionssystems liegen oder darüber hinaus gehen. Zumeist müssen Anpassungen, die unter Nutzung von vorhandenem und damit meist auch abgesicherten Flexibilitätspotenzial durchgeführt werden, nicht gesondert abgesichert werden. Demgegenüber müssen Anpassungen, die eine Erweiterung des Flexibilitätskorridors beinhalten, unbedingt abgesichert werden. Solche Anpassungen werden im Rahmen dieser Veröffentlichung als Rekonfiguration bezeichnet.

Die Rekonfiguration eines Produktionssystems kann wie in [18] dargestellt auf physischer und logischer Ebene stattfinden und hierbei unterschiedlichste Domänen und Disziplinen mit verschiedensten Zeithorizonten betreffen. Wie in [31] dargestellt mangelt es an einem eindeutigen Rekonfigurationsbegriff. Im Rahmen dieser Veröffentlichung wird die Begriffsdefinition von Matevska verwendet: „Eine Rekonfiguration stellt die technische Sicht des Prozesses der Veränderung eines bereits entwickelten und operativ eingesetzten Systems dar, um es an neue Anforderungen anzupassen, Funktionalität zu erweitern, Fehler zu beseitigen oder die Qualitätseigenschaften zu verbessern “ [23]. Hierbei wird nicht eingeschränkt, ob eine Rekonfiguration eine vom System autonom durchgeführte Änderung in der Softwaredomäne zur Laufzeit darstellt oder hingegen einen kompletten Engineeringprozess beschreibt, der bei grundlegenden Änderungen an der Mechanik oder Elektrik notwendig wird. Daher können beispielsweise sowohl Änderungen des Ausführungsortes von Softwarebausteinen bzw. gemäß IEC 61499 sogenannten Funktionsblöcken [2, 22] als auch die Ertüchtigung von Anlagen zur Herstellung neuer Produktvarianten [14, 18, 19] als Rekonfigurationen im Bereich der Produktionssysteme angesehen werden. Die in Kapitel 3 und 4 vorgestellten Ansätze setzen eine Rekonfiguration im Sinne einer Änderung des Flexibilitätskorridors voraus. Der auf einer Modellkomposition basierende Ansatz TestIAS kann auch in autonom zur Laufzeit durchgeführten Rekonfigurationsszenarien eingesetzt werden, da hierbei rein formal abgesichert wird.

Die Eigenschaften von CPPS ermöglichen die Realisierung von hoch wandlungsfähigen, funktional einfach rekonfigurierbaren Produktionssystemen. Daher werden in Bereichen, in denen häufig Anforderungsänderungen auftreten und Produktionssysteme häufig verändert werden müssen, CPPS eingesetzt. Allerdings ist die Absicherung von Rekonfigurationen von CPPS mit großem Aufwand verbunden, da hierbei der hohe Funktionsumfang und damit die hohe Flexibilität von CPPS erneut verifiziert und validiert werden muss. Aus Gründen der Wirtschaftlichkeit müssen zudem Anlagenstillstandszeiten minimal gehalten werden, was den Einsatz von formalen Verifikationsmethoden, statischen Analysewerkzeugen und simulationsgestützten Verfahren zur Absicherung notwendig macht.

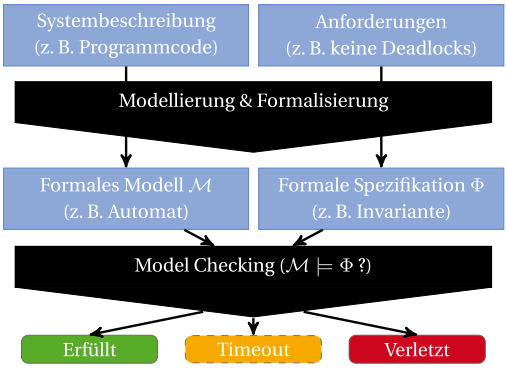

Abb. 1: Schematischer Model-Checking-Prozess [4].

Allerdings stellt laut [35] die Anwendung dynamischer Testverfahren den bisher praktizierten Stand der Technik zur Absicherung von CPS dar. Die auf einer Expertenbefragung beruhenden Ergebnisse zeigen auf, dass CPS in der Praxis überwiegend durch schlichtes Ausprobieren verifiziert und validiert werden. Dieses intuitive und mit wenig Mehraufwand verbundene Vorgehen birgt aber schwer zu tolerierende Risiken bei sicherheitskritischen Anwendungen. Zudem können dynamische Testverfahren, welche eine Ausführung des zu testenden CPS erfordern, nur dann eingesetzt werden, wenn eine Beschädigung des CPS und dessen Umgebung ausgeschlossen werden kann.

Ein anderer Ansatz ist die Nutzung von *formalen Methoden*, welche Formalismen und Verfahren zur rigorosen Modellierung und Überprüfung von Systemverhalten umfassen. Wie in Abbildung 1 skizziert, werden hierbei zunächst informelle Systembeschreibungen und Anforderungen formalisiert, d. h. in einem Formalismus ausgedrückt, dessen Syntax und Semantik eindeutig definiert ist. Für die resultierenden formalen Modelle und Spezifikationen können anschließend Analyseverfahren mit beweisbaren Eigenschaften entworfen oder angewendet werden. Die auf Exploration basierende Überprüfung, ob eine formale Spezifikation von allen erreichbaren Zuständen erfüllt wird, wird als *Model Checking* bezeichnet.

Um Steuerungssoftware formal zu analysieren, wird der informelle Programmcode in geeignete Formalismen überführt. Während zur Formalisierung von in Hochsprachen geschriebener Software Kontrollflussautomaten [8, 10, 21] verwendet werden, die es Analysen ermöglichen die Struktur und Existenz des Kontrollflusses auszunutzen, sind bei abstrahierten, booleschen und nebenläufigen Systembeschreibungen eher Varianten von Automaten, symbolischen Transitionssystemen und Petri-Netzen üblich [27, 4, 3]. Welcher Formalismus am geeignetsten ist, hängt davon ab was damit beschrieben, wie gearbeitet und welche Analysen dadurch ermöglicht werden sollen.

Methoden der formalen Verifikation zählen zu den statischen Testverfahren und können die Testabdeckung, gegenüber dem gängigen Testen durch Ausprobieren, deutlich erhöhen [35]. Zudem muss hierzu das zu testende CPS nicht ausgeführt werden und kritische Fehler können ohne Beschädigung des CPS gefunden werden. Laut [35] sind formale Verifikationsmethoden aber zu benutzerunfreundlich und ineffizient, um bei der Absicherung realer CPS Anwendungen eingesetzt zu werden.

Im folgenden Abschnitt wird mit TestIAS ein Ansatz vorgestellt, der durch Automatisierung sowohl die Anwendbarkeit als auch die Effizienz von modellbasierten formalen Verifikationsmethoden erhöht und die in [35] identifizierten Probleme aufgreift.

3 Formale Verifikation mit Modellkomposition

Wie in Abschnitt 2 dargestellt, sind CPPS hoch vernetzte Produktionssysteme mit erweiterten kognitiven Fähigkeiten. Die Realisierung dieser zusätzlichen Fähigkeiten erfordert im Vergleich zu klassischen Produktionssystemen einen höheren Softwareanteil. In [29] zeigen Vogel-Heuser, Fay et al., dass die Mechanik und Elektrik automatisierter Produktionssysteme oft jahrelang unverändert betrieben wird, wohingegen die Software teilweise sogar wöchentlich geändert wird. Diese Entwicklung wird sich durch den höheren Softwareanteil von CPPS noch verstärken. Daher adressiert der am Institut für Automatisierungstechnik und Softwaresysteme (IAS) entwickelte Ansatz TestIAS die Absicherung von Änderungen der Steuerungssoftware von CPPS. Die hierbei betrachteten Softwaremodifikationen verändern durch eine Neucodierung das Steuerungsverhalten und stellen gemäß [23] eine Rekonfiguration dar. Durch den Einsatz der in Abschnitt 2 vorgestellten formalen Verifikationsmethoden sichert TestIAS das durch die Rekonfiguration geänderte Steuerungsverhalten ab und ermöglicht so eine sichere Wiederinbetriebnahme des CPPS.

Ziel von TestIAS ist die anwenderfreundliche und effiziente Nutzung von Model Checking zur formalen Verifikation von Softwareänderungen. Um Model Checking einzusetzten muss das Verhalten der gesamten Steuerungssoftware geeignet spezifiziert und modelliert werden. Wie in Abschnitt 2 dargelegt stellen sowohl der hohe Aufwand zur Modellierung und Modellpflege als auch die lange Dauer des Verifikationsvorgangs hohe Hindernisse für die Nutzung formaler Verifikationsmethoden in der Praxis

dar. Beide Problemfelder können durch eine automatisierte Modellkomposition adressiert werden. Zur Absicherung von Softwäreänderungen setzt der als Prototyp realisierte Ansatz TestIAS das formale Verifikationsverfahren des Model Checkings in Kombination mit einer automatisierten Modellkomposition ein.

Die automatisierte Komposition von Modellen ermöglicht das Zusammensetzen von Teilmodellen mit definierten Schnittstellen zu einem Gesamtverhaltensmodell. Infolgedessen kann bei TestIAS eine komponentenbasierte Modellierung erfolgen und somit der komplexe Erstellungs- und Pflegeprozess eines Verhaltensmodells vereinfacht werden. Statt mühsam ein Gesamtmodell des CPPS-Verhaltens zu erstellen und zu pflegen, können deutlich einfachere Modelle verwendet werden, die nur einen Teil der Verhaltens modellieren. Eine komponentenbasierte Modellierung erleichtert darüber hinaus die Wiederverwendung und Weitergabe von Verhaltensmodellen. Damit vereinfacht die durch die Modellkomposition ermöglichte komponentenbasierte Modellierung die Anwendung von Model Checking zur formalen Verifikation von Verhaltensmodellen.

Dieser an „Teile-und-herrsche“-Verfahren angelehnte Mechanismus reduziert aber nicht nur die Komplexität der Modellierung für den Anwender. Vielmehr können auch bei der Verifikation maßgeschneiderte Teilmodelle zum Einsatz kommen. Nach einer automatisierten Analyse der zu verifizierenden Anforderungen können Modelle komponiert werden, welche den zur Verifikation einer Anforderung jeweilig notwendigen Teilausschnitt des CPPS-Verhaltens abbilden. Dadurch können Anforderungen an Teilverhaltensmodellen verifiziert werden. Verglichen mit einer Verifikation an einem Gesamtmodell ergeben sich hierbei Geschwindigkeitsvorteile. Im Folgenden wird die Funktionsweise von TestIAS genauer erläutert. Eine ausführliche Beschreibung findet sich auch in [33, 34].

In Abbildung 2 ist der Ablauf der anwenderfreundlichen und effizienten formalen Verifikation mit TestIAS dargestellt. Ausgangspunkt für jegliche Rekonfiguration eines CPPS stellt eine mündliche Anforderungsänderung dar. Dabei kann es sich um einen Änderungs- oder um einen Erweiterungswunsch handeln. Daraus abgeleitet müssen die zu verifizierenden, formalisierten Anforderungen der formalen Spezifikation Φ sowie die formalen Verhaltensmodelle $\mathcal{M}$ der Steuerungssoftware angepasst und aktualisiert werden. Eine erneute Verifikation der formalisierten Anforderungen mit TestIAS muss immer dann erfolgen, wenn sich formalisierte Anforderungen und beziehungsweise oder Verhaltensmodelle geändert haben.

Mithilfe der Modellkomposition von TestIAS ist es möglich, mehrere Teilmodelle zu einem Gesamtmodell zu-

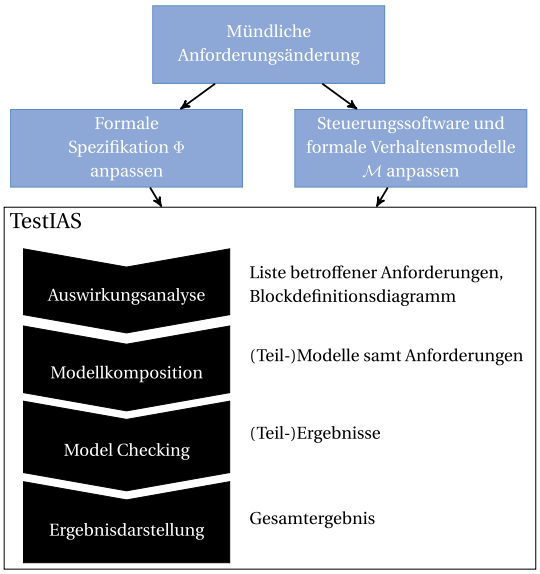

Abb. 2: Formale Verifikation von Änderungen mit TestIAS .

sammenzusetzen. Dies erfordert eine Modellierungsart, bei der Modelle über Schnittstellen zu anderen Modellen verfügen können. Mit TestIAS können Verhaltensmodelle verifiziert werden, welche mit offenen, steuerungstechnisch interpretierten Petri-Netzen (SIPN) (siehe [28, 12]) modelliert sind. Des Weiteren muss der Modellierungsgegenstand eine sinnvolle Modellierung mittels Teilmodellen erlauben. Zur Evaluierung von TestIAS wurde ein service-orientiert gesteuertes Produktionssystem angenommen. Die Steuerung erfolgt dezentral durch Dienstaufrufe. Hierzu stellt jedes Subsystem des CPPS Dienste zur Verfügung, welche zum Beispiel einzelne Produktionsschritte darstellen können. In diesem Szenario ist es vorteilhaft, wenn die Steuerung des CPPS über die Modellierung und formalisierte Spezifikation der angebotenen Dienste beschrieben wird. Um später Model Checking einzusetzten, muss daher kein Gesamtverhaltensmodell des CPPS bereitgestellt werden. Stattdessen sind deutlich kleinere Verhaltensmodelle von einzelnen Subsystemen bei der Ausführung eines bestimmten Dienstes ausreichend.

Kommt es nun zu einer Rekonfiguration eines CPPS und wurden infolgedessen die formalisierte Spezifikation Φ und die Verhaltensmodelle $\mathcal{M}$ aktualisiert, wird wie in Abb. 2 dargestellt zunächst eine Auswirkungsanalyse durchgeführt. Hierbei werden automatisiert alle Abhängigkeiten zwischen den Verhaltensmodellen untersucht. Da für die Verhaltensmodellierung offene Petri-Netze mit Interface-Stellen zum Einsatz kommen, gleicht dieser Schritt bei TestIAS der Analyse aller Modellschnittstellen aller Modelle aller vom CPPS angebotener Services. Wie in Abb. 2 dargestellt sind die Ergebnisse der Aus-

wirkungsanalyse die Auflistung aller veränderter formalisierter Anforderungen sowie ein sogenanntes Blockdefinitionsdiagramm. Das Blockdefinitionsdiagramm zeigt die Abhängigkeiten zwischen den Verhaltensmodellen und ist das Ergebnis der Analyse der Modellschnittstellen.

Die nachfolgende Modellkomposition stellt dann das Kernstück von TestIAS dar. Ausgehend vom Blockdefinitionsdiagramm können dann die zur Verifikation von durch Änderungen betroffenen formalisierten Anforderungen notwendigen Modelle komponiert werden. Je nach Umfang der Rekonfiguration des CPPS ist es hierbei möglich, auf die Komposition eines Gesamtverhaltensmodells zu verzichten. Häufig ist es ausreichend, Anforderungen an Teilmodellen zu verifizieren. Bei der Modellkomposition werden alle benötigten Verhaltensmodelle zusammengesetzt und die daran zu verifizierenden Anforderungen angehängt. Dies stellt dann die Eingänge für das verwendete Model Checking Werkzeug dar.

Die eigentliche Verifikation der von Änderungen betroffenen formalisierten Anforderungen erfolgt dann mit einem zur gewählten Modellierung passenden Model Checker. Bei TestIAS werden die in temporaler Computational Tree Logic (CTL) formal beschriebenen Anforderungen mit dem Model Checker „ITS-tools“ [26, 27] verifiziert. Die Wahl der Modellierungsart und Modellierungspräzision entscheidet über die durch Model Checking auffindbaren Fehlerarten. Da bei TestIAS die erneute Verifikation von parallel ablaufenden, diskreten Produktionsprozessen nach Änderungen im Fokus steht, wurden mit offenen, steuerungstechnisch interpretierten Petri-Netzen und CTL hierfür geeignete Modellierungsarten ausgewählt. Das Grundkonzept einer Modellkomposition lässt sich aber auch auf andere Modellierungsarten übertragen [32]. Somit ließen sich beispielsweise auch die zur Modellierung verteilter Steuerungen verbreiteten Net Condition/Event Systems (NCES) Modelle verwenden.

Durch die Modellkomposition wird der gewählte Model Checker mit den kleinst möglichen Teilmodellen versorgt. Anstatt Anforderungen am Gesamtverhaltensmodell zu verifzizieren, ist es meist ausreichend, diese an kleineren Teilmodellen zu verizifzieren. Dadurch kann eine Beschleunigung der Verifikationsdauer erreicht werden, da diese von der Modellgröße und Modellkomplexität abhängt. Um Nebenläufigkeiten korrekt zu behandeln, müssen diese in den eingesetzten Modellen durch Schnittstellen modelliert werden. Dadurch werden bei der Modellkomposition alle benötigten Modelle mitberücksichtigt.

Wenn geänderte oder von Änderungen betroffene Anforderungen anhand mehrerer Teilmodelle verifiziert werden, ist, wie in Abb. 2 dargestellt, eine Sammlung aller Teilergebnisse notwendig. Erst nach Abschluss aller Verifikationsmaßnahmen kann eine gesamtheitliche Aussage darüber getroffen werden, ob das Gesamtsystem den spezifizierten Anforderungen entspricht.

Der Ansatz TestIAS ist anwenderfreundlich, da durch die Modellkomposition eine kleinteilige Verhaltensmodellierung unterstützt wird. Zudem können durch die Auswirkungsanalyse und Modellkomposition maßgeschneiderte Modelle beim Model Checking zum Einsatz kommen, wodurch das Model Checking beschleunigt wird.

Wie bei jedem statischen Testverfahren, bei dem Systeme allein auf Basis von Modellen und formalisierten Anforderungen verifiziert werden, ist die Belastbarkeit der Ergebnisse in hohem Maße von der Qualität der Eingangsgrößen abhängig. Infolgedessen sind die Ergebnisse von TestIAS immer dann belastbar und korrekt, wenn auch die verwendeten Modelle das tatsächliche Verhalten des CPPS korrekt abbilden beziehungsweise die formalisierten Anforderungen das geforderte Verhalten korrekt spezifizieren. Im Abschnitt 4 wird daher ein Ansatz vorgestellt, welcher eine Brücke zwischen statischen und dynamischen Testmethoden bildet. Dadurch kann eine zusätzliche Verifikation der TestIAS Ergebnisse erreicht werden.

4 Software-geleitete Testfallgenerierung

Die zusätzlichen Anforderungen und Fähigkeiten rekonfigurierbarer Systeme gehen mit einem höheren Softwareanteil einher (vgl. Abschnitt 3). Mit der Notwendigkeit dieser Art von Komplexität umgehen zu können, haben sich die ursprünglich einfachen und hardwarenahen Programmiersprachen des IEC 61131-3 zunehmends in Richtung von universellen Programmiersprachen entwickelt – der Standard zieht oftmals den Vergleich mit Java. Diese Entwicklung führt auch dazu, dass die Steuerungssoftware komplexere Konstrukte der Programmiersprachen verwendet und eine Überführung in die gängigen rein booleschen formalen Modelle oftmals nicht möglich oder praktikabel ist. Entsprechend greifen aktuelle Ansätze vermehrt auf Methoden der Software-Verifikation zurück [10, 6, 21].

Der formale Beweis, dass eine Spezifikation Φ von einem formalen Modell $\mathcal{M}$ des zu untersuchenden Systems erfüllt wird, ist allerdings kein Beleg für die Korrektheit des eigentlichen Systems. So könnte z. B. das tatsächliche Verhalten der untersuchten Steuerungssoftware mit der verwendeten Modellierung in Abschnitt 3 über Petri-Netze abstrakter ausfallen, oder Eigenheiten wie Überläufe nicht

im formalen Modell eingefangen werden. Aussagen über das Verhalten des formalen Modells wären dann nicht unbedingt auf das eigentliche System übertragbar. Um dem entgegenzuwirken, wird in ARCADE.PLC [10] – einem Werkzeug zur formalen Analyse von Steuerungssoftware – das formale Modell automatisch aus dem Programmcode abgeleitet, um Diskrepanzen zwischen modelliertem und tatsächlichem Programmverhalten zu minimieren.

Nach einer Rekonfiguration der Steuerungssoftware ist die Anwendung von Regressionsverfahren eine geeignete Technik zur Absicherung. Während Verfahren zur Regressions-Verifikation nach einer Änderung der Steuerungssoftware, welches absichtlich verändertes Verhalten aufweist, zusätzliche Informationen über das geänderte Verhalten in Form von Spezifikationen benötigen um einen formalen Beweis für die Äquivalenz zu liefern, ist es mit Hilfe der Regressionstests möglich neues Verhalten, welches nicht nur auf die Software limitiert, sondern auch die physikalischen Eigenheiten der Hardware berücksichtigt, zu testen [6]. Da gerade im Kontext von rekonfigurierbarer Steuerungssoftware von CPPS Regressionstests mit hoher Testabdeckung eine wichtige Rolle spielen und von Normen wie der IEC 61508 vorgeschrieben werden, beleuchten wir im Folgenden zwei existierende und in Software-Verifikationstechniken begründete Ansätze, die ohne anwendungsspezifisches Wissen anwendbar sind.

4.1 Regressionstest-Harnisch aus Zeugen

Wird das in Abbildung 1 skizzierte Vorgehen zur Überprüfung der Erreichbarkeit bestimmter Systemzustände verwendet, so erhält man im Falle der Erreichbarkeit eine Sequenz von Zuständen, die das System zu einem der Zustände von Interesse durchläuft, oder eben über-approximierte Variablenwerte die belegen, dass es keinen Schnitt zwischen dem erreichbaren Zustandsraum und den besagten Zuständen gibt.

Solche Pfade durch den Zustandsraum sind *Zeugen* der Erreichbarkeit, da sie bezeugen, dass es eine Sequenz von Eingaben gibt, die die untersuchte Steuerungssoftware in einen Zustand von Interesse führen, und können als Tests für eine spätere Variante der Software dienen. So können Eingabesequenzen für frühere Softwareversionen verwendet werden, um zu testen ob die Software nach Rekonfiguration immer noch das geforderte Verhalten realisiert bzw. ob das problematische Verhalten durch die letzte Rekonfiguration behoben wurde, da die Zeugen alle relevanten Informationen, die zur Erfüllung des Testzwecks erforderlich sind, beinhalten. Ein wesentlicher Vorteil in der Verwendung von aus Zeugen generierten Testfällen ist,

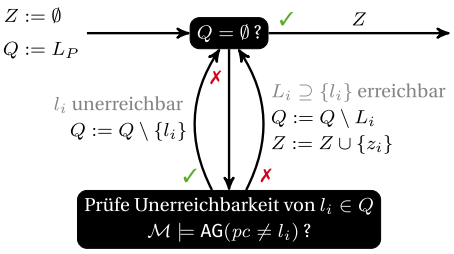

Abb. 3: Methodik zum Finden von Zeugen der Erreichbarkeit aller Anweisungen durch Model Checking.

dass potentielle Modellierungsdiskrepanzen und Fehler in der Verifikationssoftware umgangen werden, da die Eingabevektoren und erwarteten Variablenbelegungen nicht nur im formalen Modell, sondern auch auf der tatsächlichen Hardware verwendet werden können.

Dieses Konzept wurde in [25] aufgegriffen, um Testsammlungen, sogenannte *Test-Harnische*, mit maximaler Testabdeckung zu generieren. Es wird also nicht nur die Generierung von Tests aus Zeugen der Erreichbarkeit bestimmter Zustände ermöglicht, sondern auch die geleitete Generierung von Testsammlungen anhand gängiger Metriken, wie z. B. Anweisungs- oder Bedingungsüberdeckung.

Abbildung 3 veranschaulicht den Ablauf einer metrikgeleiteten Suche nach Zeugen am Beispiel der Anweisungsüberdeckung, welche das Verhältnis der von einer Testsammlung durchlaufenen Anweisungen zu allen Anweisungen des Programms beziffert. Um diese zu maximieren, müssen also Zeugen der Erreichbarkeit jeder Anweisung gefunden werden und schließlich in Tests überführt werden. Entsprechend beginnt das skizzierte Verfahren mit einer noch leeren Menge von Zeugen Z und einer Warteschlange Q von Adressen bzw. *Locations* L_P aller noch nicht erreichten Anweisungen des Programms. Um die Erreichbarkeit der i-ten Anweisung bzw. dessen Location l_i zu analysieren, kann ein Model Checker gefragt werden, ob im formalisierten Programm $\mathcal{M}$ die Invariante $pc \neq l_i$ gilt, d. h. der Programmzähler pc nie auf l_i zeigt. Ist das der Fall, so ist l_i bewiesenermaßen unerreichbar, was auf einen Fehler im Programm hindeutet – l_i wird im Folgenden nicht mehr betrachtet. Gilt die Invariante jedoch nicht, dann gibt es einen Pfad durch den Zustandsraum, der zur Anweisung an l_i führt. Dieser Pfad ist der Zeuge z_i, der über eine Menge von Locations L_i zu l_i führt, für die nicht nach weiteren Zeugen gesucht werden muss. So reduziert sich die Warteschlange in jeder Iteration um mindestens eine Location und resultiert schließlich in einer Menge Z von Zeugen zu jeder erreichbaren Anweisung der Steuerungssoftware.

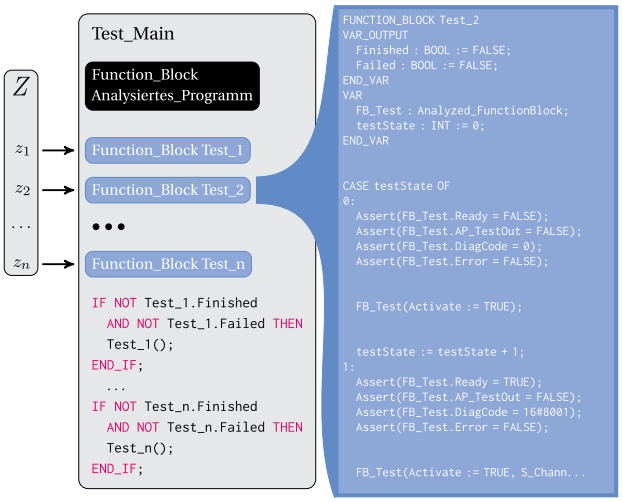

Abb. 4: Struktur generierter Test-Harnische.

Die resultierenden Pfade durch den Zustandsraum erreichen alle Zustände von Interesse und enthalten insbesondere die Variablenbelegungen der Ein- und Ausgaben – es sei denn Zustände sind beweisbar unerreichbar. Diese Werte werden von ARCADE.PLC genutzt, um einen Test-Harnisch zu synthetisieren, der das zu analysierende Programm mit den auftretenden Eingaben stimuliert und auch prüft, ob die vom Programm durchlaufenen Zustände denen des Zeugen gleichen.

Abbildung 4 visualisiert beispielhaft die Struktur solcher Harnische und deren Zusammenhang zu den Zeugen und dem analysierten Programm. Der synthetisierte Harnisch `Test_Main` ist ein Programm das auf der Zielplattform ausgeführt werden kann und das analysierte Programm oder eine neuere Version davon als *Funktionsblock* bzw. Submodul kapselt, aber im Gegensatz dazu selbst keine Eingänge verarbeitet. Stattdessen wird jeder Zeuge z_i in einen Funktionsblock `Test_i` überführt, welcher das gekapselte Programm stimuliert und nach jedem Aufruf abgleicht, ob der vom gekapselten Programm eingenommene Zustand dem des Zeugen gleicht. Hierbei werden Unterschiede als Fehlschläge gewertet und nach außen kommuniziert. Das „Nachspielen“ der nächsten Zeugen beginnt erst nach (un-)erfolgreichem Abarbeiten des vorherigen und erfordert das Zurücksetzen des gekapselten Programms auf dessen Ursprungszustand.

Während das in Abbildung 4 dargestellte Akzeptanzkriterium erfordert, dass alle Werte übereinstimmen, sind je nach Zweck auch schwächere Definitionen von Gleichheit denkbar. So könnte es z. B. auch hinreichend sein, wenn nur die Ausgänge übereinstimmen, oder nur die gleichen Anweisungen durchlaufen wurden.

Der auf Model Checking basierende Ansatz zur Generierung von Testfällen hat sich sowohl im Kontext von universellen Programmiersprachen [9], als auch in einer industrienahen Fallstudie mit Implementierungen von ABB als praktikabel erwiesen [25]. Dies liegt insbesondere daran, dass moderne Algorithmen für Model Checking den Zustandsraum nicht naiv explorieren und die potentielle Explosion des erreichbaren Zustandsraums in vielen Fällen umgehen können. So werden z. B. im Analysekontext gleichartige Zustände abstrahiert, um nur einen kleineren, abstrakten Zustandsraum explorieren zu müssen und erst bei Bedarf bzw. auftretenden Gegenbeispielen die Abstraktion zu verfeinern.

4.2 Symbolic Execution

In der Praxis lassen sich jedoch längst nicht alle Software-Komponenten und Systeme mit Model Checking analysieren, da die zur Verfügung stehenden Ressourcen limitiert sind und die Erreichbarkeit genauso wie das *Halteproblem* unentscheidbar ist. Gibt man jedoch den Anspruch auf Vollständigkeit, den Model Checking mit sich bringt, auf und verwendet stattdessen auf den Zustandsraum bezogene unter-approximierende Verfahren, die die Erreichbarkeit feststellen, aber Unerreichbarkeit nicht beweisen können, so lassen sich auch in solchen Fällen oft dennoch Tests mit hoher Abdeckung generieren [11].

Um der Zustandsraumexplosion zu begegnen, werden die Zustände bei symbolischem Model Checking nicht explizit konstruiert, sondern implizit durch Lösungen eines Bedingungserfüllbarkeitsproblems charakterisiert. So charakterisiert z. B. die Bedingung $x > 0 \wedge y > x$ alle Zustände in denen x größer als 0 und y größer als x ist. Da Maschinenzahlen im Gegensatz zu $\mathbb{Z}$ und $\mathbb{R}$ jedoch endlich sind, werden solche Bedingungen von sogenannten *SMT-Solvern (Satisfiability Modulo Theories)* gelöst, die die Erfüllbarkeit über verschiedenen *Theorien*, wie eben Maschinen- und Gleitkommazahlen, feststellen können.

Symbolic Execution bezeichnet die symbolische Interpretation vom Programmcode, welche dieses Konzept auf die Charakterisierung ganzer Ausführungspfade erweitert. Statt das Verhalten von Software für konkrete Werte zu simulieren, werden die Variablenbelegungen hier als symbolische Ausdrücke beschrieben. Da nichtdeterministische Eingaben dabei durch Konstanten ohne Einschränkungen charakterisiert werden, kann bei Verzweigungen ein SMT-Solver gefragt werden, wie die Eingänge konkret zu belegen sind – sofern überhaupt möglich – um einen bestimmten Zweig zu erreichen. Die Möglichkeit die Ausführung so in bestimmte Zustände von Interesse zu len-

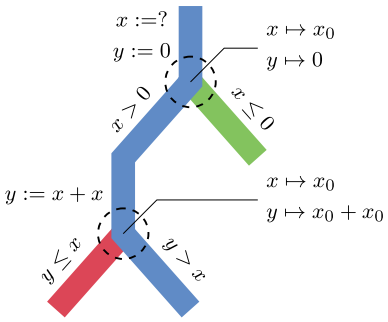

Abb. 5: Konzept symbolischer Ausführung.

ken, ermöglicht eine nach Belieben geleitete Exploration des Zustandsraums. Es können z. B. Pfade bevorzugt werden, die zu noch nicht besuchten Anweisungen führen, wenn Zeugen der Erreichbarkeit für diese von Interesse sind.

Abbildung 5 visualisiert den Graphen einer symbolischen Ausführung, welcher im Wesentlichen einem Abrollen des Kontrollflussgraphen entlang des durchlaufenen Pfades entspricht. Wie schon durch das Fragezeichen angedeutet, wird x im Beispiel als Eingabevariable behandelt, die einen beliebigen Wert annehmen kann, während y initial mit 0 belegt ist. Entsprechend ist der symbolische Wert von x an der ersten Verzweigung x_0, was eine bisher auftretende und uneingeschränkte Konstante ist. Während eine konkrete Ausführung z. B. im Zustand $[x \mapsto 42, y \mapsto 0]$ sein könnte und nun dem Zweig mit $x > 0$ folgen müsste, sind in der symbolischen Ausführung potentiell beide Pfade möglich. Hierzu wird ein SMT-Solver jeweils nach der Erfüllbarkeit von $x > 0$ und dessen Negation im aktuellen Zustand gefragt – d. h. von $x_0 > 0$ und $x_0 \leq 0$. In der Tat gibt es Belegungen von x_0 die in beide Zweige führen und sich somit Zeugen der Erreichbarkeit für beide Zweige konstruieren lassen.

Führt man die Ausführung im linken Pfad fort, so wird der Wert von y auf $x + x$ gesetzt und somit ebenfalls symbolisch. Der symbolische Zustand an der nächsten Verzweigung entspricht also $[x \mapsto x_0, y \mapsto x_0 + x_0]$. Fragt man einen SMT-Solver nun nach der Erfüllbarkeit von $y \leq x$ zusätzlich zur vorherigen Bedingung $x > 0$, d. h. von $x_0 + x_0 \leq x_0 \wedge x_0 > 0$, so stellt sich diese Bedingung als unerfüllbar heraus. An dieser Verzweigung gibt es keine Möglichkeit die Exploration in eine Richtung zu leiten, und die Ausführung geht im rechten Zweig weiter.

Diese Feststellung ist jedoch nur möglich, sofern $x, y \in \mathbb{Z}$ bzw. unbeschränkt große Integer sind. Wären sie hingegen vorzeichenlose Variablen mit 8 Bit, so würde der SMT-Solver Belegungen für beide Zweige finden. Aufgrund der Überlaufsemantik solcher Zahlen würde z. B. die Eingabe 128 in den roten Zweig führen, da insbesondere $128 + 128 = 0$ gilt.

Symbolische Ausführung vermag es die Programmsemantik präzise zu beschreiben und die Exploration in Zustände von Interesse zu lenken. Für jeden so erreichten Zustand lässt sich ein entsprechender Zeuge aus dem durchlaufenen Pfad und der vom SMT-Solver gefundenen Belegung konstruieren. Da der SMT-Solver hier im Gegensatz zum Model Checking nicht das gesamte Programmverhalten berücksichtigen muss, skaliert der Ansatz auch für größere Systeme. Die Technik wird aktiv in diversen Anwendungen und verschiedenen Erweiterungen erfolgreich verwendet [5] und hat auch im Kontext von Steuerungssoftware mehr Zeugen und entsprechende Regressionstests generieren können als der auf Model Checking basierende Ansatz [11].

5 Fazit und Ausblick

Im Zuge der agilen Produktion und der neu aufkommenden Geschäftsmodelle erfordern immer mehr Branchen eine steigende Flexibilität und Wandlungsfähigkeit der in der Produktion eingesetzten CPPS. Im Vergleich zu klassischen Produktionssystemen sind CPPS zunehmend kommunikationsfähiger, heterogener, und weisen emergentes Verhalten auf. Diese Eigenschaften gehen mit einem erhöhten Softwareanteil bis in die kleinste Komponente einher, welche auf Grund der hohen Wandlungsfähigkeit im Gegensatz zu klassischen Produktionssystemen während des Betriebs im Feld rekonfiguriert werden können. Die Verkürzung der Produktions- und Entwicklungszyklen nimmt Einfluss auf die Anforderungen für die Steuerungssoftware der CPPS, welche innerhalb kürzerer Umrüstungszeiträume möglichst effektiv angepasst und überprüft werden müssen. Dies erzeugt bei auftretenden Rekonfigurationen während des Betriebs oder in Umrüstungsphasen aufgrund des großen abzusichernden Flexibilitätskorridor von CPPS einen hohen Absicherungsaufwand.

Wie in [35] dargestellt reichen die für die Absicherung von CPPS bisher eingesetzten Verfahren nicht aus. Demgegenüber eignen sich zwar traditionelle formale Verifikationsmethoden zur gefahrlosen Absicherung von sicherheitskritischer Steuerungssoftware, stoßen aber an ihre Grenzen, welche durch den disruptiven Wandel des Qualitätsbegriffs begründet sind. Aufgrund der hohen Wandlungsfähigkeit von rekonfigurierbaren CPPS verschwimmt die klare Trennung zwischen dem statischen Testen während der Entwicklungs- und Inbetriebnahmephase und

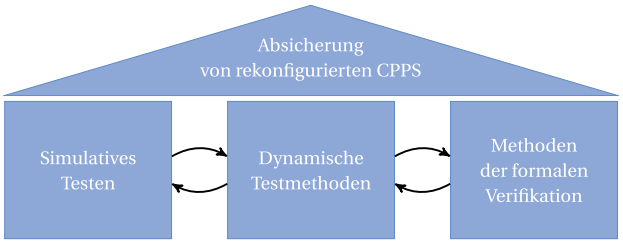

Abb. 6: Absicherung rekonfigurierter CPPS.

dem dynamischen Testen und der Qualitätssicherung während der Betriebsphase.

In diesem Beitrag wurden zwei Ansätze vorgestellt, um die aufkommenden Schwierigkeiten bei der Absicherung von Steuerungssoftware und den damit einhergehenden Problemen für die Qualitätsicherung von rekonfigurierbaren CPPS beherrschbar zu machen. Der in [35] als kritisch angesehene hohe Aufwand für den Einsatz von formalen Verifikationsmethoden kann durch automatisierte Modellkomposition reduziert werden. Hierdurch kann der Modellierungsaufwand verringert und die Verifikation beschleunigt werden. Der in Abschnitt 3 vorgestellte Ansatz TestIAS ist ein Anwendungsbeispiel für den Einsatz der automatisierten Modellkomposition. Hierbei kann insbesondere der Modellierungsaufwand zur Absicherung rekonfigurierter CPPS gesenkt werden, da die automatisierte Modellkomposition zusätzlich eine komponentenbasierte Modellierung erlaubt, welche wiederum die Wiederverwendung von Modellen vereinfacht. Im Kontext von rekonfigurierbarer Steuerungssoftware spielen Regressionstests mit hoher Testabdeckung eine wichtige Rolle. Um eine Brücke zwischen formalen Verifikationsmethoden und dynamisch ausführbaren Testfällen herzustellen, wurden formale Verfahren zur Software-geleiteten Testfallgenerierung in Arcade.PLC implementiert und im industriellen Kontext evaluiert. Durch die dynamisch ausführbaren Testfälle können Ergebnisse der formalen Verifikation bei Rekonfigurationen die während der Betriebsphase des CPPS entstanden sind, während der Umrüstung oder Wartung überprüft werden. Bringt man beide Ansätze zusammen ergibt sich ein geschlossenes Assistenzsystem zur Absicherung von CPS und CPPS.

Als Ausblick für die Absicherung von rekonfigurierten CPPS dient Abbildung 6, in welcher die drei wichtigen Säulen für die Qualitätsicherung und Absicherung der Steuerungssoftware für rekonfigurierbare CPPS symbolisiert sind. Zukünftig werden vermehrt formale Verifikationsmethoden eingesetzt, die gepaart mit dynamischen Testmethoden eine effektive Möglichkeit zur Absicherung sicherheitskritischer CPPS darstellen. Neben den schon angewandten dynamischen Testverfahren, welche die Ausführung des Programmcodes auf der Hardware erfordern, gibt es zukünftig nach Ansicht der Autoren auch noch eine dritte Säule, die des simulativen Testens. Das simulative Testen stellt eine Erweiterung der gängigen SiL-Verfahren dar, da hierbei sowohl die physikalisch-basierten Modelle des technischen Prozesses und das Verhalten der Umgebung, als auch das emergente Verhalten und die Heterogenität mit einfließen und erprobt werden können. Es zeichnet sich ab, dass formale und simulative Methoden, sowie deren Verbund zukünftig Gegenstand der Forschung bleiben und weiter befähigt werden müssen, um den entstehenden Absicherungsaufwand von rekonfigurierbaren CPPS beherrschbar zu gestalten. Von besonderem Interesse ist die Auswirkung der Rekonfiguration auf die Testautomatisierung und Testselektion. Durch die sich ändernden Konfigurationen muss mit einer sich dynamisch ändernden Testabdeckung gerechnet werden. Dies benötigt neue Verfahren für die effektive und zielgerichtete Testauswahl, welche umfangreich anhand von industriellen Fallbeispielen erprobt werden müssen. Im Vordergrund steht die Reduktion des Wartungsaufwands und der Minimierung der Stillstandszeit eines CPPS nach einer Rekonfiguration. Um die Effizienz des Quality-Engineerings auch in späteren Lebenszyklusphasen weiter zu erhöhen, sind darüber hinaus Ansätze von Interesse, welche Synergieeffekte zwischen Verfahren zur Absicherung von Software sowie der Mechanik und Elektrik ermöglichen.

Autorenbeiträge: Die Autoren Marco Grochowski und Andreas Löcklin haben in gleichem Maße zu dieser Veröffentlichung beigetragen.

Literatur

1. Definition von cpps durch vdi/vde gma fachausschuss 7.20 (cyber physical systems) und 7.21 (industrie 4.0). http://i40.iosb.fraunhofer.de/Cyber-Physical Production System (CPPS). Aufgerufen am: 21.08.2019.
2. Y. Al-Safi and V. Vyatkin. An ontology-based reconfiguration agent for intelligent mechatronic systems. In V. Mařík, V. Vyatkin and A. W. Colombo, editors, *Holonic and Multi-Agent Systems for Manufacturing*, pages 114–126, 2007. Springer Berlin Heidelberg, Berlin, Heidelberg.
3. E. G. Amparore, B. Berthomieu, G. Ciardo, S. Dal-Zilio, F. Gallà, L. Hillah, F. Hulin-Hubard, P. G. Jensen, L. Jezequel, F. Kordon, D. L. Botlan, T. Liebke, J. Meijer, A. S. Miner, E. Paviot-Adet, J. Srba, Y. Thierry-Mieg, T. van Dijk and K. Wolf. Presentation of the 9th edition of the model checking contest. In *Tools and Algorithms for the Construction and Analysis of Systems – 25 Years of TACAS: TOOLympics*, Held as Part of ETAPS 2019,

Prague, Czech Republic, April 6–11, 2019, Proceedings, Part III, pages 50–68, 2019.
4. C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.
5. R. Baldoni, E. Coppa, D. C. D'Elia, C. Demetrescu and I. Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3):50:1–50:39, 2018.
6. B. Beckert, M. Ulbrich, B. Vogel-Heuser and A. Weigl. Regression verification for programmable logic controller software. In *Formal Methods and Software Engineering – 17th International Conference on Formal Engineering Methods, ICFEM 2015*, Paris, France, November 3–5, 2015, Proceedings, pages 234–251, 2015.
7. K. Bettenhausen and S. Kowalewski. Cyber-physical systems: Chancen und nutzen aus sicht der automation. Technical report, VDI/VDE GMA, Düsseldorf, 2013.
8. D. Beyer, T. A. Henzinger and G. Théoduloz. Configurable software verification: Concretizing the convergence of model checking and program analysis. In *Computer Aided Verification, 19th International Conference, CAV 2007*, Berlin, Germany, July 3–7, 2007, Proceedings, pages 504–518, 2007.
9. D. Beyer and T. Lemberger. Software verification: Testing vs. model checking – A comparative evaluation of the state of the art. In *Hardware and Software: Verification and Testing – 13th International Haifa Verification Conference, HVC 2017*, Haifa, Israel, November 13–15, 2017, Proceedings, pages 99–114, 2017.
10. S. Biallas, J. Brauer and S. Kowalewski. Arcade.plc: a verification platform for programmable logic controllers. In *IEEE/ACM International Conference on Automated Software Engineering, ASE'12*, Essen, Germany, September 3–7, 2012, pages 338–341, 2012.
11. D. Bohlender, H. Simon, N. Friedrich, S. Kowalewski and S. Hauck-Stattelmann. Concolic test generation for PLC programs using coverage metrics. In C. G. Cassandras, A. Giua and Z. Li, editors, *13th International Workshop on Discrete Event Systems, WODES 2016*, Xi'an, China, May 30–June 1, 2016, pages 432–437. IEEE, 2016.
12. G. Frey. Hierarchical design of logic controllers using signal interpreted petri nets. *IFAC Proceedings Volumes*, 36(6):361–366, 2003.
13. E. Geisberger and M. Broy. *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*, volume 1. Springer-Verlag, 2012.
14. A. Hees and G. Reinhart. Approach for production planning in reconfigurable manufacturing systems. *Procedia CIRP*, 33:70–75, 2015. 9th CIRP Conference on Intelligent Computation in Manufacturing Engineering – CIRP ICME'14.
15. A. F. Hees. *System zur Produktionsplanung für rekonfigurierbare Produktionssysteme*. Dissertation, Technische Universität München, München, 2017.
16. E. Järvenpää, N. Siltala and M. Lanz. Formal resource and capability descriptions supporting rapid reconfiguration of assembly systems. In *2016 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pages 120–125. IEEE, 2016.
17. N. Jazdi. Cyber physical systems in the context of industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–4, May 2014.
18. E. Järvenpää, N. Siltala, O. Hylli and M. Lanz. Capability matchmaking procedure to support rapid configuration and re-configuration of production systems. *Procedia Manufacturing*, 11:1053–1060, 2017. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27–30 June 2017, Modena, Italy.
19. Y. Koren, X. Gu and W. Guo. Reconfigurable manufacturing systems: Principles, design, and future trends. *Frontiers of Mechanical Engineering*, 13(2):121–136, Jun 2018.
20. S. Kowalewski and K. D. Bettenhausen. *Cyber-physical systems: Chancen und nutzen aus sicht der automation*. VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik, 2013.
21. T. Lange, M. R. Neuhäußer and T. Noll. Speeding up the safety verification of programmable logic controller code. In *Hardware and Software: Verification and Testing – 9th International Haifa Verification Conference, HVC 2013*, Haifa, Israel, November 5–7, 2013, Proceedings, pages 44–60, 2013.
22. W. Lepuschitz, A. Zoitl, M. Vallée and M. Merdan. Toward self-reconfiguration of manufacturing systems using automation agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1):52–69, Jan 2011.
23. J. Matevska. *Rekonfiguration komponentenbasierter Softwaresysteme zur Laufzeit*. Springer, 2010.
24. C. Müller-Schloer, H. Schmeck and T. Ungerer. Organic computing. *Informatik-Spektrum*, 35:71–73, 04 2012.
25. H. Simon, N. Friedrich, S. Biallas, S. Hauck-Stattelmann, B. Schlich and S. Kowalewski. Automatic test case generation for PLC programs using coverage metrics. In *20th IEEE Conference on Emerging Technologies & Factory Automation, ETFA 2015*, Luxembourg, September 8–11, 2015, pages 1–4. IEEE, 2015.
26. Y. Thierry-Mieg. Symbolic model-checking using its-tools. In *Tools and Algorithms for the Construction and Analysis of Systems – 21st International Conference, TACAS 2015*, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11–18, 2015, Proceedings, pages 231–237, 2015.
27. Y. Thierry Mieg. *From Symbolic Verification To Domain Specific Languages*. Habilitation à diriger des recherches, Sorbonne Université, UPMC; Laboratoire d'informatique de Paris 6 [LIP6], Dec. 2016.
28. W. M. Van Der Aalst, N. Lohmann, P. Massuthe, C. Stahl and K. Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *The Computer Journal*, 53(1):90–106, 2010.
29. B. Vogel-Heuser, A. Fay, I. Schaefer and M. Tichy. Evolution of software in automated production systems: Challenges and research directions. *Journal of Systems and Software*, 110:54–84, 2015.
30. M. Weyrich, M. Klein, J.-P. Schmidt, N. Jazdi, K. D. Bettenhausen, F. Buschmann, C. Rubner, M. Pirker and K. Wurm. *Evaluation Model for Assessment of Cyber-Physical Production Systems*, pages 169–199. Springer International Publishing, Cham, 2017.
31. J. Yan and V. Vyatkin. Extension of reconfigurability provisions in iec 61499. In *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–7, 09 2013.

32. A. Zeller, N. Jazdi and M. Weyrich. Functional verification of distributed automation systems: Assisting production line operators by an automated model composition. *The International Journal of Advanced Manufacturing Technology*, pages 1–14, 07 2019.
33. A. Zeller and M. Weyrich. Component based verification of distributed automation systems based on model composition. *Procedia CIRP*, 72:352–356, 2018.
34. A. Zeller and M. Weyrich. Composition of modular models for verification of distributed automation systems. *Procedia Manufacturing*, 17:870–877, 2018.
35. X. Zheng and C. Julien. Verification and validation in cyber physical systems: Research challenges and a way forward. In *1st IEEE/ACM International Workshop on Software Engineering for Smart Cyber-Physical Systems, SEsCPS 2015*, Florence, Italy, May 17, 2015, pages 15–18, 2015.

Autoreninformationen

Marco Grochowski
Lehrstuhl Informatik 11 – Embedded Software, RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Deutschland
grochowski@embedded.rwth-aachen.de

Marco Grochowski, M.Sc. ist seit 2017 wissenschaftlicher Mitarbeiter am Lehrstuhl Informatik 11 – Embedded Software. Er arbeitet im Exzellenzcluster „Internet of Production“ (IoP) und untersucht wie traditionelle Verifikations- und Testmethoden für Software auf die im Rahmen des IoP entstehenden Technologien anwendbar gemacht werden können, während gleichzeitig die gängigen industriellen Anforderungen eingehalten werden.

Hendrik Simon
Lehrstuhl Informatik 11 – Embedded Software, RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Deutschland
simon@embedded.rwth-aachen.de

Hendrik Simon, M.Sc. hat Informatik an der RWTH Aachen in Deutschland studiert und die Abschlüsse B.Sc./M.Sc. in 2010/13 erlangt. Danach schloss er sich der Forschungsgruppe von Professor Kowalewski an und arbeitete von 2014 bis 2018 als wissenschaftlicher Mitarbeiter am Lehrstuhl Informatik 11 – Embedded Systems. Der Fokus seiner Arbeit lag auf der automatischen Generierung von Testfällen für SPS-Software. In diesem Rahmen beschäftigte sich Herr Simon in seiner Forschung unter anderem mit den Themen Model Checking, Program Slicing, Symbolic Execution und Concolic Testing.

Dimitri Bohlender
Lehrstuhl Informatik 11 – Embedded Software, RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Deutschland
bohlender@embedded.rwth-aachen.de

Dimitri Bohlender, M.Sc. ist seit 2014 wissenschaftlicher Mitarbeiter am Lehrstuhl Informatik 11 – Embedded Software. In seiner Promotion forscht er an symbolischen Methoden zur formalen Verifikation von Software reaktiver Systeme, denen Techniken wie „Symbolic Model Checking“ und „SAT/SMT/CHC-solving“ zugrunde liegen. Beim „Workshop on Dependable Control of Discrete Systems“ (DCDS’13) wurde ihm für die Ergebnisse seiner Bachelorarbeit der „Best Paper Award“ verliehen.

Stefan Kowalewski
Lehrstuhl Informatik 11 – Embedded Software, RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Deutschland
kowalewski@embedded.rwth-aachen.de

Prof. Dr.-Ing. Stefan Kowalewski ist Leiter des Lehrstuhls Informatik 11 – Embedded Software der RWTH Aachen University mit dem Forschungsschwerpunkt der Entwurfs- und Analysemethoden für softwareintensive, eingebettete und cyber-physische Systeme. Sein besonderes Interesse gilt den formalen und semi-formalen Methoden zur Programmanalyse und des Designs und der Validierung von sicherheitskritischen Systemen.

Andreas Löcklin
Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Deutschland
andreas.loecklin@ias.uni-stuttgart.de

Andreas Löcklin, M.Sc. ist wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik und Softwaresysteme. Hauptforschungsgebiet: Absicherung rekonfigurierter cyber-physischer Produktionssysteme.

Timo Müller
Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Deutschland
timo.mueller@ias.uni-stuttgart.de

Timo Müller, M.Sc. ist wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik und Softwaresysteme. Hauptforschungsgebiet: Rekonfiguration cyber-physischer Produktionssysteme.

Nasser Jazdi
Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Deutschland
nasser.jazdi@ias.uni-stuttgart.de

Dr.-Ing. Nasser Jazdi ist akademischer Oberrat des Instituts für Automatisierungstechnik und Softwaresysteme der Universität Stuttgart. Hauptforschungsgebiete: Internet of Things sowie Lernfähigkeit, Zuverlässigkeit und Sicherheit in der Automatisierungstechnik.

Andreas Zeller
Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Deutschland
andreas.zeller@ias.uni-stuttgart.de

Andreas Zeller, M.Sc. war von 2014–2019 wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik und Softwaresysteme. Hauptforschungsgebiet: Softwaretest verteilter Automatisierungssysteme.

Michael Weyrich
Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Deutschland
michael.weyrich@ias.uni-stuttgart.de

Prof. Dr.-Ing. Dr.h.c. Michael Weyrich ist Leiter des Instituts für Automatisierungstechnik und Softwaresysteme der Universität Stuttgart. Hauptforschungsgebiete: Methoden und Tools zur Komplexitätsreduktion von Software in der Automatisierungstechnik.