Spontane Sicherheitsprüfung mittels individualisierter Programmzertifizierung oder Programmrestrukturierung[1]

Marie-Christine Jakobs[2]

Abstract: Korrekt funktionierende Software gewinnt immer mehr an Bedeutung. Im Vergleich zu früher ist es heutzutage schwieriger einzuschätzen, wie gut eine Software funktioniert. Dies liegt unter anderem daran, dass Endnutzer häufiger Software unbekannter Hersteller installieren. Endnutzer sollten sich also aktiv von der Softwarekorrektheit überzeugen, zum Beispiel in Form einer spontanen Sicherheitsprüfung. Übliche Verifikationstechniken zur Korrektheitsprüfung kommen für Endnutzer, in der Regel Laien, nicht in Frage. Die zentrale Frage ist daher, wie man einem Laien eine solche spontane Sicherheitsprüfung ermöglicht. Die Antwort der Dissertation sind einfache, automatische und generelle Verfahren zur Sicherheitsprüfung. In der Dissertation werden verschiedene Verfahren vorgeschlagen und sowohl theoretisch als auch praktisch untersucht. Die vorgeschlagenen Verfahren lassen sich in zwei Forschungsrichtungen einsortieren, nämlich in die Gruppe der Proof-Carrying Code Verfahren bzw. in die Gruppe des alternativen Programs from Proofs Verfahren. Einige Verfahren kombinieren beide Forschungsrichtungen.

1 Motivation

Im Zeitalter von mobilen Endgeräten und „smarten“ Geräten ist es Gang und Gäbe die Funktionen der Geräte mittels Softwareapplikationen aus dem Internet zu erweitern. (Unbeabsichtigte) Fehler, die in fast jeder Softwareapplikation existieren, können unsere Sicherheit bedrohen, zumindest aber den Betrieb eines unserer Geräte stören. Da wir uns im Alltag immer mehr auf unsere Geräte verlassen, kann ein solcher Fehler einen dramatischen Einfluss auf unser Leben haben.

Früher haben wir überwiegend Software von wenigen, namhaften Herstellern installiert, bei denen wir aufgrund von Erfahrung einschätzen konnten, wie gut die Qualität ihrer Produkte ist, das heißt, wie fehleranfällig ihre Produkte sind. Mit dem Aufkommen der mobilen Endgeräte hat sich unser Verhalten geändert. Wir installieren immer häufiger Applikationen von vielen verschiedenen, uns unbekannten Hersteller. Eine Abschätzung der Fehleranfälligkeit von Applikationen wird deutlich schwieriger. Mehr denn je sollten wir Nutzer uns vor der Installation selbst aktiv davon überzeugen, dass eine Softwareapplikation unseren Qualitätsansprüchen genügt.

Verifikation ist ein geeignetes Mittel der Qualitätsprüfung, sofern man Experte ist. Tatsächlich sind die meisten Nutzer aber Verifikationslaien und scheitern bereits daran, das richtige Verfahren für ihr Verifikationsproblem aus der Vielzahl der Verfahren zu wählen.

[1] Englischer Titel der Dissertation: “On-the-fly Safety Checking – Customizing Program Certification and Program Restructuring”

[2] Software and Computational Systems Lab, LMU Munich, jakobs@sosy.ifi.lmu.de

Es wird also ein möglichst einfaches, generelles und automatisches Verfahren benötigt, dass die spontane Qualitätsprüfung zuverlässig für den Nutzer übernimmt. Da wir uns im Folgenden insbesondere für Sicherheitseigenschaften (im Sinne von „safety“) interessieren, nennen wir die Qualitätsprüfung von nun an Sicherheitsprüfung.

Bereits um die Jahrtausendwende herum hat sich die Verifikationsgemeinschaft mit einer ähnlichen Fragestellung beschäftigt. Die Lösung für die spontane Sicherheitsprüfung war das von Necula vorgeschlagene Proof-Carrying Code Protokoll [Ne97]. Die Idee des Protokolls ist, dass der Softwarehersteller die Verifikation der Software übernimmt und dem Nutzer seine aus der Verifikation gewonnen Beweisinformationen zur Verfügung stellt. Die spontane Sicherheitsprüfung des Nutzers umfasst dann nur noch die einfachere Beweisprüfung. Die Gemeinschaft hat eine Vielzahl von speziellen Proof-Carrying Code Verfahren für diverse Analysen und Eigenschaftsklassen entwickelt. Kürzlich wurde sogar ein alternatives Protokoll, genannt Programs from Proofs [WSW13], vorgeschlagen. Allerdings habe alle Ansätze in der Literatur einen entscheidenden Nachteil. Sie sind nicht allgemein anwendbar oder nicht automatisch.

Ziel der Dissertation [Ja17] ist es daher generelle, automatische Verfahren zur spontanen Sicherheitsprüfung zu entwickeln. Die Verfahren bauen auf dem Proof-Carrying Code Protokoll oder dem Programs from Proofs Protokoll auf. Es werden auch Verfahren entwickelt, die eine Kombination beider Protokolle nutzen. Außerdem werden die Verfahren nicht nur entwickelt, sondern auch hinsichtlich ihrer Nützlichkeit für eine spontane Sicherheitsprüfung untersucht. Zur Nützlichkeit gehören sowohl theoretische Betrachtungen, zum Beispiel ist das Verfahren zuverlässig, als auch praktische Fragestellungen, zum Beispiel wie effizient ist die spontane Sicherheitsprüfung.

Der nächste Abschnitt gibt einen genaueren Überblick über die Anforderungen an die entwickelten Verfahren und eine schematische Darstellung ihrer allgemeinen Funktionsweise. Die Details der verschiedenen Verfahren sowie ihre Eigenschaften werden im danach folgenden Abschnitt beleuchtet. Daran anschließend werden die entwickelten Verfahren verglichen. Zum Schluss wird ein Fazit gezogen.

2 Überblick

In der Dissertation werden verschiedene Verfahren zur spontanen Sicherheitsprüfung entwickelt und evaluiert. Die Verfahren gehören zu einem von zwei Forschungsansätzen (Proof-Carrying Code [Ne97], Programs from Proofs [WSW13]) oder deren Kombination. Nichtsdestotrotz verfolgen alle entwickelten Verfahren die gleichen folgenden Ziele.

Stichhaltigkeit Bestätigt die spontane Sicherheitsprüfung eine Programmeigenschaft, so gilt diese.

Vollständigkeit Eine korrekte Vorbereitung der spontanen Sicherheitsprüfung garantiert ihren Erfolg.

Automatismus Mit Ausnahme einer anfänglichen Konfiguration der Vorbereitungsphase verläuft die Sicherheitsprüfung vollautomatisch ohne menschliche Intervention.

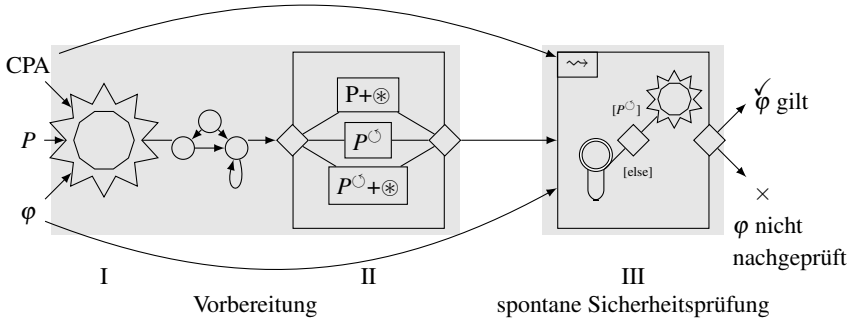

Abb. 1: Schematische Darstellung der Funktionsweise der spontanen Sicherheitsprüfungen

Generalität Die spontane Sicherheitsprüfung unterstützt verschiedene Eigenschaften.

Effizienz Die spontane Sicherheitsprüfung ist schneller und benötigt weniger Speicher als eine normale Verifikation.

Stichhaltigkeit und Vollständigkeit der Verfahren werden formal bewiesen. In wie weit die Verfahren Automatismus und Generalität unterstützen wird diskutiert und die Effizienz wird experimentell evaluiert.

Eine weitere Gemeinsamkeit der Verfahren ist ihre abstrakte Funktionsweise. Trotz all ihrer Detailunterschiede folgen alle entwickelten Verfahren dem gleichen Schema, das in Abb. 1 skizziert ist. Zuerst muss die spontane Sicherheitsprüfung vorbereitet werden (linker, grauer Kasten in Abb. 1). Die Vorbereitung wird in aller Regel nicht vom spontan Prüfenden sondern vom Programmhersteller, -anbieter, etc. durchgeführt und geschieht geplant vorab.

Der erste Vorbereitungsschritt verifiziert, dass das Programm P die Sicherheitseigenschaft φ erfüllt. Die Verifikation ist für alle Verfahren gleich. Um die Generalität zu gewährleisten, nutzen wir eine parametrisierte Verifikation. Diese bekommt neben dem Programm auch die zu prüfende Eigenschaft und eine zur Eigenschaft passenden Analysekonfiguration. Die zu prüfenden „safety“ Eigenschaften werden in sogenannten Eigenschaftsautomaten kodiert. Unter anderem können Eigenschaftsautomaten die Nichterreichbarkeit von Programmzuständen, Invarianten und Protokolle spezifizieren. Allgemein gesehen kodieren sie die Menge der verbotenen Ausführungspfade. Für die Analysekonfiguration verwenden wir das Prinzip der konfigurierbaren Programmanalyse (CPA) [BHT07]. Eine konfigurierbare Programmanalyse beschreibt eine abstrakte Programmsemantik und Operatoren zur Steuerung einer abstrakten Erreichbarkeitsanalyse. Hauptsächlich gesteuert durch die konfigurierbare Programmanalyse führt die parametrisierte Verifikation genauso eine abstrakte Erreichbarkeitsanalyse durch. Das Ergebnis einer erfolgreichen Verifikation ist ein abstrakter Erreichbarkeitsgraph, eine abstrakte Beschreibung des Zustandsraums des analysierten Programms.

Im zweiten Vorbereitungsschritt werden Information aus dem Erreichbarkeitsgraph extrahiert und für die spontane Sicherheitsprüfung zur Verfügung gestellt. Die Vorgehensweise ist dabei abhängig vom Forschungsansatz. Die Verfahren im Bereich Proof-Carrying Code geben dem Programm ein separates *Zertifikat* ($\circledast$) mit. Das Zertifikat speichert Teile des Erreichbarkeitsgraphen. Verfahren im Bereich Programs from Proofs kodieren die extrahierten Informationen direkt in einem *transformierten Programm* $P^{\circlearrowleft}$. Das transformierte Programm ist eine Restrukturierung des Originalprogramms P, in der unmögliche Pfade gelöscht und Ausführungspfade syntaktisch voneinander getrennt wurden. In der Kombination wird sowohl die Programmtransformation aus dem Bereich Programs from Proofs angewendet, als auch ein separates Zertifikat für das transformierte Programm erstellt.

Der dritte und letzte Schritt ist die spontane Sicherheitsprüfung. Neben der zu prüfenden Eigenschaft und dem im Schritt II generierten Artefakt bekommt die spontane Sicherheitsprüfung auch die ursprüngliche Analysekonfiguration. Diese wird benötigt, um die spontane Sicherheitsprüfung automatisch an die zu prüfende Eigenschaft anzupassen. Dafür wird aus der ursprünglichen Konfiguration automatisch eine Konfiguration für die spontane Sicherheitsprüfung abgeleitet ($\rightsquigarrow$), die auf die Prüfung der Eigenschaft mittels des Artefaktes abgestimmt ist. Die Verfahren im Bereich Proof-Carrying Code und der Kombination inspizieren das Zertifikat und prüfen, ob es valide bezeugt, dass das Programm im Artefakt die zu prüfenden Eigenschaft erfüllt. Dagegen verifizieren Verfahren im Bereich Programs from Proofs das transformierte Programm bezüglich der zu prüfenden Eigenschaft unter Verwendung des gleichen Verfahrens wie im Schritt I der Vorbereitung. Details folgen im nächsten Abschnitt, in dem wir einen genaueren Blick auf die konkret entwickelten Verfahren werfen.

3 Verfahren zur spontanen Sicherheitsprüfung

Wie bereits im Überblick skizziert, verfolgt die Dissertation drei grundsätzliche Techniken zur Realisierung der spontanen Sicherheitsprüfung. Die folgenden Abschnitte geben einen Einblick in die entwickelten Verfahren der jeweiligen Technik.

3.1 Konfigurierbare Softwarezertifizierung

Die Verfahren zur konfigurierbaren Softwarezertifizierung folgen der Proof-Carrying Code Idee [Ne97]. Das bedeutet, dass mit dem Programm Teile des Korrektheitsbeweises in Form eines Zertifikats mitgegeben werden. Die Teile des Korrektheitsbeweises ermöglichen die spontanen Sicherheitsprüfung, welche den Beweis effizient nachprüft, das heißt, ihn nachvollzieht und gegebenenfalls in Teilen nachberechnet. In unserem Fall liegt der Beweis in Form des abstrakten Erreichbarkeitsgraphens vor. Seine zentrale Eigenschaft ist eine sichere Überapproximation der erreichbaren Programmzustände, das heißt, die Überapproximation verletzt die zu prüfende Eigenschaft nicht. Im Prinzip ist die Überapproximation vollständig durch die Knoten des Graphens beschrieben. Denn die Knoten repräsentieren die abstrakten Zustände, das heißt, Mengen von realen Programmzuständen.

Daher speichern die Zertifikate der konfigurierbare Softwarezertifizierung lediglich Knoten. Für die konfigurierbare Softwarezertifizierung wurden Verfahren mit verschiedenen Zertifikatstypen entworfen. Aufbauend auf einem simplen Basisverfahren wurden zwei orthogonale Optimierungen, die unterschiedliche Optimierungsziele verfolgen, entwickelt und letztendlich in einem Verfahren integriert. Die folgende Aufstellung gibt einen Überblick über die betrachteten Zertifikatstypen.

1. Basis: Speicherung aller Knoten
2. Optimierungen
   a) Reduktion: Speicherung einer Teilmenge der Knoten
   b) Aufteilung: Speicherung einer Partition der Menge der Knoten
3. Integration: Speicherung einer Partition einer Teilmenge der Knoten

Ziel beider Optimierungen ist es, die Zeit der Zertifikatsprüfung zu verringern. Die Reduktionsoptimierung reduziert die Größe des Zertifikats und somit die Einlesezeit des Zertifikates. Gleichzeitig verhindert sie unnötige Nachprüfungen. Welche Knoten mindestens gespeichert werden müssen hängt von der Struktur des Graphens und der Analyseart ab. Die Details befinden sich in Kapitel 4.1 der Dissertation. Die Aufteilungsoptimierung reduziert die Prüfungszeit, indem sie Einlesen und Nachprüfung parallelisiert. Die Integration kombiniert die vorherigen Optimierungen. Zusätzlich kann man mit ihr alle vorherigen Zertifikate beschreiben, wenn auch nicht in syntaktisch identischer Form, da die Partitionsgröße und die zu speichernde Teilmenge der Knoten prinzipiell eingestellt werden können.

Die Nachprüfung des Beweises mit Hilfe des Zertifikates ist im Wesentlichen eine Überprüfung, ob das Zertifikat eine sichere Überapproximation darstellt oder sich zu einer sicheren Überapproximation erweitern lässt. Dafür werden die abstrakte Semantik der initialen konfigurierbaren Programmanalyse verwendet und Teile ihrer Operatoren im Prinzip wiederverwendet. Die konkrete Überprüfung unterscheidet sich für die verschiedenen Zertifikatstypen. Allerdings, bleibt die Grundidee für alle Verfahren die gleiche:

- Nachberechnen der abstrakten Nachfolgezustände aller gespeicherten und explorierten Zustände.
- Prüfen, ob die Nachfolgezustände von gespeicherten Zuständen überdeckt werden. Nicht überdeckte Nachfolgezustände werden in die Menge der explorierten Zustände aufgenommen.
- Prüfen, ob die finale Abstraktion sicher ist.
- Scheitern, sobald zu viele Zustände exploriert wurden.[3]

Betrachten wir die fünf Ziele, so ergibt sich folgendes Bild. Die Stichhaltigkeit und Vollständigkeit werden in der Dissertation für alle Zertifikatstypen und -verfahren formal bewiesen. Die Automatisierung funktioniert für viele Analysen und alle Eigenschaften. Für

[3] Dieses Abbruchkriterium ist notwendig, um die Terminierung der Überprüfung und somit die Vollständigkeit des Verfahrens zu garantieren.

einige Analysen muss ein verwendeter Analyseoperator manuell präzisiert werden. Somit sind die Verfahren generell.

Die Effizienz wurde mit 20 Analysekonfigurationen und einer Teilmenge einer großen Softwareverifikationsbenchmark evaluiert. Die Ergebnisse der Evaluation sind wie folgt. Die Effizienz des Ansatzes hängt stark von der Analyse ab, die durch Konfiguration, Eigenschaft und Programm bestimmt wird. In der Regel sind die Optimierungen effizienter als der Basisansatz. Die besten Optimierungen sind die Reduktionsoptimierung sowie die Integration. Zwischen den beiden gibt es keinen klaren Sieger. Im Vergleich mit ähnlichen Verfahren zeigt sich, dass diese höchstens einen kleinen Vorteil auf Analysetypen haben, für die sie ursprünglich entwickelt wurden. Im Allgemeinen kann die jeweils beste Optimierung mit den anderen Verfahren mithalten, obwohl ihre Zertifikate deutlich größer als das Programm selbst und auch teils größer als manche Zertifikate konkurrierender Ansätze sind.

3.2 Generisches Programs from Proofs

Ziel des Programs from Proofs Verfahrens ist es, simple und effiziente Datenflussanalysen für die spontanen Sicherheitsprüfungen zu verwenden. Allerdings scheitern Datenflussanalysen alleine häufig daran, die gewünschte Programmeigenschaft zu zeigen, da sie lediglich fluss- aber nicht pfadsensitiv sind. Schauen wir uns zunächst das linke Programm in Abb. 2 an. Um zu zeigen, dass vor dem Fahrversuch, das Fahrzeug gestartet wird, muss die Analyse wissen, dass die beiden if-Blöcke über die Bedingung p gekoppelt sind. Betrachten wir nun das rechte Programm in Abb. 2. Da das rechte Programm die beiden if-Blöcke des linken Programms zu einem Block zusammenfasst, ist ein Wissen über p unnötig. Die Programmstruktur alleine genügt, um die beschriebene Eigenschaft zu zeigen. Die Restrukturierung des linken Programms ermöglicht also eine simple und effiziente Eigenschaftsanalyse. Analysen, die die gewünschte Programmeigenschaft nachweisen können, machen häufig implizit eine solche Programmrestrukturierung und ihre Restrukturierung findet sich im abstrakten Erreichbarkeitsgraph wieder. Dies macht sich das Programs from Proofs Verfahren, welches zum ersten Mal von Wonisch et al. [WSW13] vorgeschlagen wurde, zu Nutze. Beginnend mit einer kombinierten Analyse, die die effiziente Zielanalyse inkorporiert, aber dennoch mächtiger ist, wird das Programm verifiziert. Um die Eigenschaft mit der Zielanalyse durchführen zu können, bedient sich der Ansatz eines Tricks. Die im Erreichbarkeitsgraphen kodierte Restrukturierung wird ins Programm übertragen. Konkret wird der Erreichbarkeitsgraph, insbesondere seine Struktur, in ein Programm übersetzt. Wonisch et al. [WSW13] haben dieses Prinzip lediglich auf eine spezielle Analyse und Eigenschaftsklasse angewendet. In der Dissertation wird das Prinzip generalisiert.

```
if(p)          if(p)
  start();       start();
if(p)            fahr();
  fahr();
```

Abb. 2: Programm mit Restrukturierung

Im Gegensatz zur vorherigen konfigurierbaren Softwarezertifizierung verhindert die Programmrestrukturierung im Verfahren, dass dieses Verfahren mit allen beliebigen Eigenschaften und Analysen funktioniert. Eigenschaften dürfen sich nicht direkt auf den Pro-

grammzähler („program counter“) beziehen. Ebenso benötigen wir für die anfängliche konfigurierbare Programmanalyse eine bestimmte strukturelle Beschaffenheit. So besteht diese Analyse aus zwei Komponenten: dem Eigenschaftsprüfer und der Hilfsanalyse. Der Eigenschaftsprüfer alleine prüft die Eigenschaft. Er ist flusssensitiv und resistent gegenüber Programmrestrukturierung. Seine Datenflussvariante wird später für die spontane Sicherheitsprüfung verwendet. Die Hilfsanalyse darf unmögliche Pfade ausschließen und Programmausführungen voneinander separieren. Sie beeinflusst den Eigenschaftsprüfer aber nie direkt. Die Details können in der Dissertation [Ja17, S. 150ff] nachgelesen werden.

Die Generalität des Verfahrens wurde im vorherigen Absatz bereits beleuchtet. Kommen wir nun zu den vier restlichen Zielen. Da das Verfahren für die spontane Sicherheitsprüfung das Analyseverfahren aus der Vorbereitung nutzt, ergibt sich die Stichhaltigkeit aus diesem Analyseverfahren, welches selbst stichhaltig ist. Der Nachweis der Vollständigkeit gliedert sich in zwei Teile: a) Vollständigkeit unter der Annahme der Terminierung der Analyse und b) Terminierung. Teilaspekt a) wurde formal nachgewiesen. Der Nachweis für Teilaspekt b) gelang für verschiedene Typen von Analysen, zum Beispiel Modelchecking, und insbesondere für alle praktisch relevanten Analysen, allerdings nicht allgemein. Der Automatismus ergibt sich aus der Konstruktion des Verfahrens.

Für die Evaluation der Effizienz wurden 3 Hilfsanalysen mit 12 Eigenschaftsprüfern kombiniert, um mit ihnen das Program from Proofs Verfahren mit verschiedenen Eigenschaften auf Programmen unterschiedlicher Softwareverifikationsbenchmarks auszuführen. Neben dem Vergleich mit der normalen Verifikation, der Verifikation in der Vorbereitung, wurde die spontane Sicherheitsprüfung auch mit der Prüfung in der konfigurierbaren Softwarezertifizierung verglichen. Dabei stellte sich heraus, dass die spontane Sicherheitsprüfung des Programs from Proofs Verfahrens meist effizienter ist als die normale Verifikation und mit der konfigurierbaren Softwarezertifizierung mindestens mithalten kann. Zusätzlich ergab die Evaluation, dass die transformierten Programme häufig nicht allzu viel größer sind als das Originalprogramm, meist höchstens 5-mal so groß, und nicht selten auch kleiner.

Neben den eben betrachteten fünf Zielen ergeben sich aufgrund der Programmtransformation weitere Fragen. Zentral für die Anwendung des Ansatzes ist die Frage, ob das Originalprogramm und das transformierte Programm äquivalent sind. In der Dissertation wurde gezeigt, dass die beiden Programm sich äquivalent modulo Programmzähler verhalten, also sich die möglichen Ausführungspfade der Programm sich nur in den Werten des Programmzählers unterscheiden. Praktisch verhalten sie sich also äquivalent. Zusätzlich wurde untersucht welche Eigenschaften des Originalprogramms auf dem transformierten Programm erhalten bleiben beziehungsweise beweisbar bleiben (siehe [Ja17, S. 171f, 186ff]).

3.3 Konfigurierbare Softwarezertifizierung für Programs from Proofs

Ziel der Kombination ist es, von den Stärken der beiden vorherigen Ansätze zu profitieren, insbesondere soll das Terminierungsproblem des Programs from Proofs Ansatzes überwunden werden. Gleichzeitig ermöglicht eine Kombination eine noch einfachere und gegebenenfalls effizientere, spontane Prüfung.

Die naive Kombination führt die beiden Ansätze sequentiell hintereinander aus. Die Vorbereitung beginnt wie die Programs from Proofs Vorbereitung. Sie hört dort aber nicht auf, sondern setzt mit der spontanen Sicherheitsprüfung der Program from Proofs Technik fort, die gleichzeitig Schritt I der konfigurierbaren Softwarezertifizierung darstellt. Danach endet die Vorbereitung mit Schritt II der konfigurierbaren Softwarezertifizierung. Diese naive Lösung verschiebt daher lediglich das Terminierungsproblem anstatt es zu lösen. Zusätzlich wird in der Vorbereitung das Programm im Prinzip doppelt verifiziert, also unnötig viel Aufwand betrieben. Eine sinnvolle Kombination muss daher ein Zertifikat für das transformierte Programm aus dem Analyseergebnis vom Originalprogramm berechnen.

In der Dissertation werden zwei Varianten aufgezeigt, ein Zertifikat für das transformierte Programm zu berechnen:

- Transformation des abstrakten Erreichbarkeitsgraphens und Zertifikatsgenerierung aus dem transformierten Graphen
- Erstellung eines Zertifikats für das Originalprogramm und Transformation des Zertifikats auf das transformierte Programm

In beiden Fällen wird die Struktur des Graphens beziehungsweise des Zertifikates in der Transformation erhalten. Lediglich die abstrakten Zustände werden angepasst, indem die Informationen der Hilfsanalyse entfernt werden und die Programmzähler („program counter“) entsprechend dem Wissen über die Programmtransformation angepasst werden. Ebenso funktionieren beide Varianten mit allen Zertifikatsarten der konfigurierbaren Softwarezertifizierung. Die beiden Verfahren unterscheiden sich nur in zwei Aspekten. Die Transformation des abstrakten Erreichbarkeitsgraphens braucht mehr Speicher. Im Gegensatz erhält man bei der Transformation des Zertifikats für manche Zertifikatstypen teilweise größere Zertifikate.

Aufgrund der Eigenschaften der konfigurierbaren Softwarezertifizierung und des Programs from Proofs Verfahrens sind die Kombinationsverfahren selber stichhaltig, vollständig und automatisch, sofern die erzeugten Zertifikate von der konfigurierbaren Softwarezertifizierung unter Betrachtung des transformierten Programms erstellt werden könnten. Diese Eigenschaft der erstellten Zertifikate wird in der Dissertation formal für beide Varianten bewiesen. Weil die Kombination die Generalität von dem Program from Proofs Ansatz erbt, bleibt lediglich die Frage der Effizienz.

Anstatt die Kombination mit der normalen Verifikation zu vergleichen, haben wir die spontane Sicherheitsprüfung der Kombination mit der Program from Proofs Technik verglichen. Wir wollten also primär wissen, ob wir die Programs from Proofs Technik verbessern konnten. Das Ergebnis ist, dass dies eher selten möglich ist, da die spontane Sicherheitsprüfung der Programs from Proofs Technik bereits sehr effizient ist.

4 Vergleich der Verfahren

Wir vergleichen die entwickelten Verfahren zunächst anhand der fünf Ziele. Alle Verfahren sind stichhaltig. Die Kombination ist sowohl automatisch als auch vollständig. Im Gegensatz dazu sind die Verfahren der konfigurierbaren Softwarezertifizierung nicht immer automatisch und für die Program from Proofs Technik konnte die Vollständigkeit nicht für alle Analysen gezeigt werden. Die konfigurierbare Softwarezertifizierung ist am allgemeinsten. Sie funktioniert für alle konfigurierbaren Analysen und Eigenschaften. Der Programs from Proofs Ansatz und die Kombination funktionieren lediglich auf einer Teilmenge der Analysen und Eigenschaftsklassen. Diese Teilmenge ist für beide gleich. Bezüglich der Effizienz gilt Folgendes. Auch wenn die Verfahren mit existierenden Verfahren mithalten können, so ist das Program from Proofs Verfahren im direkten Vergleich mit der konfigurierbaren Softwarezertifizierung beziehungsweise der Kombination etwas effizienter.

Kommen wir nun zu zwei weiteren Kriterien. Als erstes betrachten wir die sogenannte „Trusted Computing Base“, das heißt, die Komponenten, deren Implementierung man trauen muss, um dem Ergebnis der spontanen Sicherheitsprüfung zu glauben. Die kleinste „Trusted Computing Base“ hat die Kombination. Der Vergleich der anderen beiden Verfahren gestaltet sich schwieriger. Zum einen ist die Frage, ob die Inspizierung des Zertifikats schwieriger zu implementieren ist als eine Erreichbarkeitsanalyse. Aufgrund der Ähnlichkeit, der Algorithmen ist ihre Implementierung ähnlich schwer. Zum anderen stellt sich die Frage, was ist fehleranfälliger die Hilfsanalyse oder die Kombination von abstrakten Zuständen. Weil die Hilfsanlysen häufig externe Tools nutzen, sind sie, so denken wir, fehleranfälliger. Es scheint so, dass die Program from Proofs Technik einen kleinen Vorteil gegenüber der konfigurierbaren Softwarezertifizierung hat. Als zweites gucken wir uns den Festplattenspeicherbedarf an. Dort ist die Kombination am ineffizientesten. Der Vergleich der anderen beiden Ansätze ergibt, dass die Zertifikate der konfigurierbaren Zertifikate häufig größer sind als die Differenz zwischen der Größe des Originalprogramms und der Größe des transformierten Programms, was den zusätzlichen Speicherbedarf des Programs from Proofs Verfahrens charakterisiert.

5 Fazit

Obwohl alle Verfahren zumindest stichhaltig sind, so haben sie doch unterschiedliche Stärken und Schwächen. Die Kombination empfiehlt sich, wenn Terminierung ein Problem bei der Anwendung des Program from Proofs Verfahrens ist. Ansonsten bietet sich das Programs from Proofs Verfahren für alle Eigenschaften an, die nicht direkt vom Programmzähler abhängen. Falls die Eigenschaft direkt vom Programmzähler abhängt oder keine passende Kombination aus Hilfs- und Eigenschaftsanalyse gefunden werden kann, kann man auf die konfigurierbare Softwarezertifizierung zurückgreifen.

Literaturverzeichnis

[BHT07] Beyer, Dirk; Henzinger, Thomas A.; Théoduloz, Grégory: Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis. In

(Damm, Werner; Hermanns, Holger, Hrsg.): Computer Aided Verification. Jgg. 4590 in LNCS, Springer, Berlin, Heidelberg, S. 504–518, 2007.

[Ja17] Jakobs, Marie-Christine: On-The-Fly Safety Checking – Customizing Program Certification and Program Restructuring. Dissertation, Universität Paderborn, 2017.

[Ne97] Necula, George C.: Proof-carrying Code. In: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM, New York, NY, USA, S. 106–119, 1997.

[WSW13] Wonisch, Daniel; Schremmer, Alexander; Wehrheim, Heike: Programs from Proofs – A PCC Alternative. In (Sharygina, Natasha; Veith, Helmut, Hrsg.): Computer Aided Verification. Jgg. 8044 in LNCS, Springer, Berlin, Heidelberg, S. 912–927, 2013.

Marie-Christine Jakobs wurde am 3. März 1987 in Bühl geboren. Sie studierte von 2006 bis 2012 Informatik an der Universität Paderborn. In dieser Zeit absolvierte sie sowohl ihren Bachelor- als auch Masterabschluss mit Auszeichnung. Im Anschluss forschte sie im Sonderforschungsbereich 901 „On-the-fly Computing“. Gleichzeitig promovierte sie an der Universität Paderborn in der Gruppe von Prof. Dr. Heike Wehrheim. Ihre Promotion schloss sie im Mai 2017 mit Auszeichnung ab. Seit Oktober 2017 ist sie Postdoktorandin an der LMU München am Lehrstuhl von Prof. Dr. Dirk Beyer.