Lecture Notes in Computer Science 10012

Yliès Falcone · César Sánchez (Eds.)

Runtime Verification

16th International Conference, RV 2016
Madrid, Spain, September 23–30, 2016
Proceedings

Springer

Editors

Yliès Falcone
Université Grenoble Alpes, Inria
Grenoble
France

César Sánchez
IMDEA Software Institute
Madrid
Spain

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-46981-2 ISBN 978-3-319-46982-9 (eBook)
DOI 10.1007/978-3-319-46982-9

Library of Congress Control Number: 2016952525

LNCS Sublibrary: SL2 – Programming and Software Engineering

Preface

This volume contains the proceedings of the 16th International Conference on Runtime Verification (RV 2016), which was held September 23–30, 2016, at La Residencia de Estudiantes of the Spanish Council for Scientific Research (CSIC) in Madrid, Spain.

During the first half of the twentieth century, La Residencia was a prestigious cultural institution that helped foster and create the intellectual environment for young thinkers, writers, and artists. It was one of the most vibrant and successful experiences of scientific and artistic creation and exchange of interwar Europe. Some of the brightest minds of the time, like Albert Einsten, Marie Curie, and Salvador Dali, visited La Residencia in this early epoch. In the last few years there has been a very intense attempt to recover the memory of La Residencia and its founding principles, and to promote new cultural and scientific activities based on the spirit of cooperation and sharing of knowledge. We hope that the attendees of RV 2016 enjoyed this unique venue.

The RV conference is concerned with all aspects of monitoring and analysis of hardware, sotfware, and more general system executions. Runtime verification techniques are lightweight techniques to asses correctness, reliability, and robustness; these techniques are significantly more powerful and versatile than conventional testing, and more practical than exhaustive formal verification.

RV started in 2001 as an annual workshop and turned into an annual conference in 2010. The proceedings from 2001 to 2005 were published in the *Electronic Notes in Theoretical Computer Science*. Since 2006, the RV proceedings have been published in Springer's *Lecture Notes in Computer Science*. The previous five editions of the RV conference took place in San Francisco, USA (2011), Istanbul, Turkey (2012), Rennes, France (2013), Toronto, Canada (2014), and Vienna, Austria (2015).

RV 2016 received 72 submissions, 49 of which were regular papers, ten short papers, six regular tool papers, two tool demonstration papers, and five tutorial proposals. Most papers were reviewed by four reviewers. The Program Committee accepted 18 regular papers, four short papers, three regular tool papers, two tool demonstration papers, and the five submitted tutorials.

The evaluation and selection process involved thorough discussions among the members of the Program Committee and external reviewers through the EasyChair conference manager, before reaching a consensus on the final decisions.

This year, the RV conference also included the organization of The First International Summer School on Runtime Verification, co-organized and sponsored by EU COST Action IC1402 "ArVi: Runtime Verification Beyond Monitoring." Additionally, the Third International Competition on Runtime Verification, also sponsored by EU COST Action IC1402, was colocated with RV 2016.

The conference program included the presentation of the peer-reviewed papers and tool demonstrations, tutorials, and invited keynote speeches. The conference program spanned over four rich days (see http://rv2016.imag.fr).

We are pleased to have hosted three top invited speakers:

- Gul Agha, Professor of Computer Science at the University of Illinois at Urbana-Champaign, talked about how to build dependable concurrent systems through probabilistic inference, predictive monitoring, and self-adaptation.
- Oded Maler, Research Director of CNRS at Verimag, talked about how to monitor qualitative and quantitative properties, in real and virtual executions of systems, in the online and offline approaches of runtime verification.
- Fred B. Schneider, Professor of Computer Science and Chair of Cornell's CS Department, talked about tag specification languages for policy enforcement.

The conference included the following five tutorials:

- Doron Peled presented a tutorial on "Using Genetic Programming for Software Reliability"
- Nikolaï Kosmatov and Julien Signoles presented a tutorial on "Frama-C, a Collaborative Framework for C Code Verification"
- Philip Daian, Dwight Guth, Chris Hathhorn, Yilong Li, Edgar Pek, Manasvi Saxena, Traian Florin Serbanuta, and Grigore Rosu presented a tutorial on "Runtime Verification at Work"
- Sylvain Hallé presented a tutorial on "When RV Meets CEP"
- Borzoo Bonakdarpour and Bernd Finkbeiner presented a tutorial on "Runtime Verification for HyperLTL"

We would like to thank the authors of all submitted papers, the members of the Program Committee, and the external reviewers for their exhaustive task of reviewing and evaluating all submitted papers. We would like to thank Christian Colombo for co-organizing the Summer School and Sylvain Hallé and Giles Reger for co-organizing the third edition of the competition on Runtime Verification (CRV 2016).

We would also like to thank Universidad Carlos III and the IMDEA Software Institute for their administrative support and their generous monetary contribution to the conference, the Laboratoire d'Informatique de Grenoble for its IT support, and La Residencia for sharing their facilities to hold the conference at reduced prices. We highly appreciate EasyChair for its system to manage submissions. Finally, we would like to extend our special thanks to the chair of the Steering Committee, Klaus Havelund, for his support during the organization of RV 2016.

August 2016

Yliès Falcone
César Sánchez

Organization

Program Chairs

Yliès Falcone	Université Grenoble Alpes, Inria, Grenoble, France
César Sánchez	IMDEA Software Institute, Madrid, Spain

Tool Track Chair

Klaus Havelund	Nasa Jet Propulsion Laboratory, USA

Tool Committee

Steven Arzt	EC Spride, Germany
Howard Barringer	The University of Manchester, UK
Ezio Bartocci	TU Wien, Austria
Martin Leucker	University of Lübeck, Germany
Gordon Pace	University of Malta, Malta
Giles Reger	The University of Manchester, UK
Julien Signoles	CEA, France
Oleg Sokolsky	University of Pennsylvania, USA
Bernhard Steffen	University of Dortmund, Germany
Nikolai Tillmann	Microsoft Research, USA
Eugen Zalinescu	ETH Zurich, Switzerland

CRV'16 Chairs

Yliès Falcone	Université Grenoble Alpes, Inria, France
Sylvain Hallé	Université du Québec à Chicoutimi, Canada
Giles Reger	University of Manchester, Manchester, UK

Local Organization Chair

Juan Tapiador	Universidad Carlos III de Madrid, Madrid, Spain

Program Committee

Erika Abraham	RWTH Aachen University, Germany
Steven Artz	EC SPRIDE
Howard Barringer	The University of Manchester, UK
Ezio Bartocci	TU Wien, Austria
Andreas Bauer	NICTA and Australian National University, Australia

Saddek Bensalem	VERIMAG, France
Eric Bodden	Fraunhofer SIT and Technische Universität Darmstadt, Germany
Borzoo Bonakdarpour	McMaster University, Canada
Laura Bozzelli	Technical University of Madrid (UPM), Spain
Juan Caballero	IMDEA Software Institute, Spain
Wei-Ngan Chin	National University of Singapore, Singapore
Christian Colombo	University of Malta, Malta
Jyotirmoy Deshmukh	Toyota Technical Center
Alexandre Donzé	UC Berkeley, USA
Ylies Falcone	University Grenoble Alpes, Inria, Laboratoire d'Informatique de Grenoble, France
Bernd Finkbeiner	Saarland University, Germany
Adrian Francalanza	University of Malta, Malta
Vijay Garg	University of Texas at Austin, USA
Patrice Godefroid	Microsoft Research
Susanne Graf	Joseph Fourier University/CNRS/VERIMAG, France
Radu Grosu	Vienna University of Technology, Austria
Sylvain Hallé	Université du Québec à Chicoutimi, Canada
Klaus Havelund	Jet Propulsion Laboratory, California Institute of Technology, USA
Joxan Jaffar	National University of Singapore, Singapore
Thierry Jéron	Inria Rennes - Bretagne Atlantique, France
Johannes Kinder	Royal Holloway, University of London, UK
Felix Klaedtke	NEC Europe Ltd.
Kim Larsen	Aalborg University, Denmark
Axel Legay	IRISA/Inria, Rennes, France
Martin Leucker	University of Lübeck, Germany
Benjamin Livshits	Microsoft Research
Joao Lourenço	Universidade Nova de Lisboa, Portugal
Rupak Majumdar	MPI-SWS
Oded Maler	CNRS-VERIMAG, France
Leonardo Mariani	University of Milano-Bicocca, Italy
David Naumann	Stevens Institute of Technology, USA
Gordon Pace	University of Malta, Malta
Doron Peled	Bar-Ilan University
Lee Pike	Galois, Inc.
Giles Reger	University of Manchester, UK
Grigore Rosu	University of Illinois at Urbana-Champaign, USA
Gwen Salaün	Grenoble Alpes University, Inria, France
Cesar Sanchez	IMDEA Software Institute, Spain
Sriram Sankaranarayanan	University of Colorado, Boulder, USA
Gerardo Schneider	Chalmers University of Gothenburg, Sweden
Julien Signoles	CEA LIST
Scott Smolka	Stony Brook University, USA
Oleg Sokolsky	University of Pennsylvania, USA

Bernhard Steffen	University of Dortmund, Germany
Scott Stoller	Stony Brook University, USA
Volker Stolz	University of Oslo, Norway
Jun Sun	Singapore University of Technology and Design, Singapore
Juan Tapiador	Universidad Carlos III de Madrid, Spain
Serdar Tasiran	Koc University, Turkey
Nikolai Tillman	Microsoft Research
Michael Whalen	University of Minnesota, USA
Eugen Zalinescu	Technical University of Munich, Germany
Lenore Zuck	University of Illinois in Chicago, USA

Additional Reviewers

Assaf, Mounir
Azzopardi, Shaun
Bertrand, Nathalie
Dabaghchian, Maryam
Daian, Philip
Decker, Normann
Della Monica, Dario
Duan, Lian
Duc Hiep, Chu
Evrard, Hugues
Faymonville, Peter
Gossen, Frederik
Hedin, Daniel
Jaksic, Stefan
Khoury, Raphael
Komp, John
Kopetzki, Dawid
Kuester, Jan-Christoph
Le, Ton-Chanh
Lee, Benedict
Li, Yilong
Matar, Hassan Salehe
Maubert, Bastien
Mens, Irini-Eleftheria
Mikučionis, Marius
Mohammad Hasani, Ramin
Mutlu, Erdal
Neubauer, Johannes
Quilbeuf, Jean
Ratasich, Denise
Rodionova, Alena
Ruething, Oliver
Scheffel, Torben
Schmitz, Malte
Selyunin, Konstantin
Serwe, Wendelin
Siddique, Umair
Sirjani, Marjan
Srivastav, Abhinav
Tan, Tian Huat
Tekle, Tuncay
Torfah, Hazem
Traonouez, Louis-Marie
Ulus, Dogan
Vorobyov, Kostyantyn
Walulya, Ivan
Yong, Chang
Zadok, Erez
Zhang, Yi

Invited Papers

Building Dependable Concurrent Systems Through Probabilistic Inference, Predictive Monitoring and Self-adaptation (Abstract)

Gul Agha

University of Illinois at Urbana-Champaign, Champaign, USA
http://osl.cs.illinois.edu

Abstract. The infeasibility of statically verifying complex software is well established; in concurrent systems, the difficulty is compounded by nondeterminism and the possibility of 'Heisenbugs'. Using runtime verification, one can not only monitor a concurrent system to check if it has violated a specification, but potentially predict future violations. However, a key challenge for runtime verification is that specifications are often incomplete. I will argue that the safety of concurrent systems could be improved by observing patterns of interaction and using probabilistic inference to capture intended coordination behavior. Actors reflecting on their choreography this way would enable deployed systems to continually improve their specifications. Mechanisms to dynamically add monitors and enforce coordination constraints during execution would then facilitate self-adaptation in concurrent systems. I will conclude by suggesting a program of research to extend runtime verification so systems an evolve robustness through such self-adaptation.

Acknowledgements. The work on this paper has been supported in part by Air Force Research Laboratory and the Air Force Office of Scientific Research under agreement number FA8750-11-2-0084, and by National Science Foundation under grant number CCF-1438982 and NSF CCF 16-17401.

References

1. Astley, M., Sturman, D.C., Agha,G.: Customizable middleware for modular distributed software. Commun. ACM, **44**(5), 99–107 (2001)
2. Donkervoet, B., Agha, G.: Reflecting on aspect-oriented programming, metaprogramming, and adaptive distributed monitoring. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) FMCO 2006. LNCS, vol. 4709, pp. 246–265. Springer, Heidelberg (2007)
3. Frolund, S., Agha, G.: A language framework for multi-object coordination. In: Nierstrasz, O. (ed.) ECOOP 1993. LNCS, vol. 707, pp. 346–360. Springer, Heidelberg (1993)
4. Sen, K., Rosu, G., Agha, G.: Online efficient predictive safety analysis of multi-threaded programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 123–138. Springer, Heidelberg (2004)

5. Sen, K., Vardhan, A., Agha, G., Rosu, G.: Efficient decentralized monitoring of safety in distributed systems. In: Finkelstein, A., Estublier, J., Rosenblum, D.S. (eds.) ICSE 2004, Edinburgh, United Kingdom, 23–28 May 2004, pp. 418–427. IEEE Computer Society (2004)
6. Sturman, D.C., Agha, G.: A protocol description language for customizing semantics. In: 13th Symposium on Reliable Distributed Systems (SRDS 1994), Dana Point, California, 25–27 October 1994, pp. 148–157. ACM (1994)

Why Tags Could be It? Keynote Lecture Extended Abstract

Fred B. Schneider

Department of Computer Science, Cornell University, Ithaca, New York, 14853, USA
fbs@cs.cornell.edu

Abstract. Reference monitors embody specifications about permitted and prohibited operation invocations. That limits what policies they can enforce. Those limitations have prompted us to explore alternative approaches to policy enforcement—specifically, expressive classes of labels that give permitted and prohibited uses for a piece of information. These *reactive information flow* (RIF) labels will be described, along with means for static and run-time verification of programs that process such labelled data. Use of RIF labels for specifying use-based privacy also will be discussed.

1 Introduction

Security policies can be enforced by defining guards on operations or by associating labels with values, as follows.

– A *guard* on an operation *Op* is checked each time *Op* is invoked; the guard blocks any invocation that would not comply with the policy.
– A *security label* on a value or variable *V* is checked before *V* is read or written; the access is blocked when it is inconsistent with what the security label allows.

Today's systems tend to be built in terms of guards on operations rather than in terms of security labels on values. This is unfortunate, because security labels specify and provide end-to-end guarantees about information use, whereas guards on operations do not.

For example, consider a system that creates and maintains a replica *F*′ of some file *F*. A guard that prevented principal *Alice* from invoking a `read` operation naming *F* is not obliged to prevent *Alice* from invoking a `read` operation naming *F*′. But an end-to-end guarantee that stipulates *Alice* not read the contents in *F* would have to

Joint work with Cornell Ph.D. students Elisavet Kozyri and Eleanor Birrell.

F.B. Schneider—Supported in part by AFOSR grant F9550-16-0250 and grants from Microsoft. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government.

prevent attempts by *Alice* to learn the contents of F' or other values derived directly or indirectly from the contents in F. In addition, security tags can afford providers of information with flexibility to choose security policies after a system has been developed, deployed, or put into operation. Policy now accompanies a system's inputs instead of being fixed in the code.

2 Reactive Information Flow Specifications

The prevalence today of guards over security labels is not surprising, given limitations in the expressive power of currently available classes of security labels. To help overcome those limitations, we have been developing a new class of security labels: *reactive information flow* (RIF) specifications. Informally, a RIF specification for a value V gives

(i) allowed uses for V, and
(ii) the RIF specification for any value that might be directly or indirectly derived from V.

RIF specifications thus give allowed uses for the value produced by evaluating a function, where those restrictions may differ from the allowed uses for inputs to that evaluation. For instance, using RIF specifications as labels, the output of an encryption function can be public even though is inputs (plaintext and a key) are secret. In general, RIF specifications support *reclassifiers* that increase restrictions, decrease restrictions, or associate incomparable restrictions.

Various *carriers* can be instantiated to embody RIF specifications. A carrier must accept a language of reclassifiers, and it must associate a set of restrictions with each word in that language. Carriers for which language-inclusion is decidable are a good choice when we wish to treat RIF specifications as types, since the resulting type system will be statically checkable. To date, we have experience with two classes of (decidable) carriers.

- Finite state automata suffice for many common security needs. Here, each automaton state gives a set of use restrictions; reclassifiers label transitions between automaton states, with the successor automaton state giving the new set of use restrictions for a derived value.
- A simple form of push-down automata suffice for handling confidentiality when encryption and decryption are used to transform values (typically from secret to public and back). Encryption pushes a key onto the stack; decryption causes pop if the key being provided matches the key contained in top entry on the stack (and otherwise the decryption causes a push).

Type systems have been formulated for both kinds of carriers, where type correctness ensures that certain non-interference properties are satisfied. The conservative nature of type checking, however, is now leading us to contemplate run-time monitors for programs having RIF specifications as labels for values and variables. We also have been exploring practical aspects of using RIF specifications. For this, the information-flow type system in the JIF programming language has been replaced by a RIF type system based on finite-state automata. Prototype applications that we programmed in this JRIF language have given us experience with defining RIF specifications.

3 What RIF Tags May Restrict

Security labels traditionally have been interpreted as characterizing sets of principals. For confidentiality, a label specifies principals that are allowed to read a value (or any value derived); for integrity, a label describes principals that must be trusted for the labeled value to be trusted (which implies that the label defines a set of principals that may update the labeled value).

In practice, other forms of use restrictions are important too. In *use-based security*, pieces of information are labeled—actually or notionally—with tags that specify *use restrictions*, and principals who hold or process such pieces of information are obliged to comply with those restrictions. Use restrictions may come from those who submit or control the information, systems that process the information, and/or regulations imposed by the jurisdiction in which a system is located, the data originates, or its owners reside.

Use-based security can be quite general if we are given an expressive enough language for specifying the use restrictions. By choosing a suitable language, for example, we can support the various definitions of privacy that are being discussed, now that the failings of classical "notice and consent" have become apparent. We can also support regimes where data collection and use are limited by legislative authorities that specify when and how data may used, combined, how long it must be saved, etc.

RIF specifications seem well suited for defining restrictions for use-based security. Here, restrictions are not limited to being sets of principals; the restrictions instead can be permissions, prohibitions, and/or obligations for invoking arbitrary classes of operations. Reclassifiers, as before, allow derived values to be subject to different use restrictions. This capability, for example, would enable a RIF specification to assert that an individual's value must be kept confidential, but any derived value produced by statistical aggregation is public.

4 Enforcement

Formal verification, automated analysis, and run-time monitoring all are time-honored methods to ensure that a program will satisfy some property of interest. The trade-offs between expressiveness, conservatism, and automation are likely to be the same for RIF specifications as has been found for other classes of program properties. In connection with privacy, however, audit, with deterrence through accountability is sensible. So instead of preventing violations, a system detects violations and recovers. Prevention is not necessary, here.

Contents