Kedar S. Namjoshi
Tomohiro Yoneda
Teruo Higashino
Yoshio Okamura (Eds.)

LNCS 4762

Automated Technology for Verification and Analysis

5th International Symposium, ATVA 2007
Tokyo, Japan, October 2007
Proceedings

Springer

Kedar S. Namjoshi Tomohiro Yoneda
Teruo Higashino Yoshio Okamura (Eds.)

Automated Technology for Verification and Analysis

5th International Symposium, ATVA 2007
Tokyo, Japan, October 22-25, 2007
Proceedings

Springer

Volume Editors

Kedar S. Namjoshi
Alcatel-Lucent
Bell Labs
600 Mountain Avenue, Murray Hill, NJ 07974, USA
E-mail: kedar@research.bell-labs.com

Tomohiro Yoneda
National Institute of Informatics
Information Systems Architecture Research Division
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: yoneda@nii.ac.jp

Teruo Higashino
Osaka University
Department of Information Networking
Graduate School of Information Science and Technology
Suita, Osaka 565-0871, Japan
E-mail: higashino@ist.osaka-u.ac.jp

Yoshio Okamura
Semiconductor Technology Academic Research Center (STARC)
17-2, Shin Yokohama 3-chome, Kohoku-ku, Yokohama 222-0033, Japan
E-mail: okamura.yoshio@starc.or.jp

Library of Congress Control Number: 2007937234

CR Subject Classification (1998): B.1.2, B.5.2, B.6, B.7.2, C.2, C.3, D.2, D.3, F.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN	0302-9743
ISBN-10	3-540-75595-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-75595-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12173525 06/3180 5 4 3 2 1 0

Lecture Notes in Computer Science 4762

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Lecture Notes in Computer Science

Sublibrary 2: Programming and Software Engineering

For information about Vols. 1– 4143
please contact your bookseller or Springer

Vol. 4767: F. Arbab, M. Sirjani (Eds.), International Symposium on Fundamentals of Software Engineering. XIII, 450 pages. 2007.

Vol. 4764: P. Abrahamsson, N. Baddoo, T. Margaria, R. Messnarz (Eds.), Software Process Improvement. XI, 225 pages. 2007.

Vol. 4762: K.S. Namjoshi, T. Yoneda, T. Higashino, Y. Okamura (Eds.), Automated Technology for Verification and Analysis. XIV, 566 pages. 2007.

Vol. 4758: F. Oquendo (Ed.), Software Architecture. XVI, 340 pages. 2007.

Vol. 4757: F. Cappello, T. Herault, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. XVI, 396 pages. 2007.

Vol. 4753: E. Duval, R. Klamma, M. Wolpers (Eds.), Creating New Learning Experiences on a Global Scale. XII, 518 pages. 2007.

Vol. 4749: B.J. Krämer, K.-J. Lin, P. Narasimhan (Eds.), Service-Oriented Computing – ICSOC 2007. XIX, 629 pages. 2007.

Vol. 4748: K. Wolter (Ed.), Formal Methods and Stochastic Models for Performance Evaluation. X, 301 pages. 2007.

Vol. 4741: C. Bessière (Ed.), Principles and Practice of Constraint Programming – CP 2007. XV, 890 pages. 2007.

Vol. 4735: G. Engels, B. Opdyke, D.C. Schmidt, F. Weil (Eds.), Model Driven Engineering Languages and Systems. XV, 698 pages. 2007.

Vol. 4716: B. Meyer, M. Joseph (Eds.), Software Engineering Approaches for Offshore and Outsourced Development. X, 201 pages. 2007.

Vol. 4680: F. Saglietti, N. Oster (Eds.), Computer Safety, Reliability, and Security. XV, 548 pages. 2007.

Vol. 4670: V. Dahl, I. Niemelä (Eds.), Logic Programming. XII, 470 pages. 2007.

Vol. 4652: D. Georgakopoulos, N. Ritter, B. Benatallah, C. Zirpins, G. Feuerlicht, M. Schoenherr, H.R. Motahari-Nezhad (Eds.), Service-Oriented Computing ICSOC 2006. XVI, 201 pages. 2007.

Vol. 4634: H. Riis Nielson, G. Filé (Eds.), Static Analysis. XI, 469 pages. 2007.

Vol. 4615: R. de Lemos, C. Gacek, A. Romanovsky (Eds.), Architecting Dependable Systems IV. XIV, 435 pages. 2007.

Vol. 4610: B. Xiao, L.T. Yang, J. Ma, C. Muller-Schloer, Y. Hua (Eds.), Autonomic and Trusted Computing. XVIII, 571 pages. 2007.

Vol. 4609: E. Ernst (Ed.), ECOOP 2007 – Object-Oriented Programming. XIII, 625 pages. 2007.

Vol. 4608: H.W. Schmidt, I. Crnković, G.T. Heineman, J.A. Stafford (Eds.), Component-Based Software Engineering. XII, 283 pages. 2007.

Vol. 4591: J. Davies, J. Gibbons (Eds.), Integrated Formal Methods. IX, 660 pages. 2007.

Vol. 4589: J. Münch, P. Abrahamsson (Eds.), Product-Focused Software Process Improvement. XII, 414 pages. 2007.

Vol. 4574: J. Derrick, J. Vain (Eds.), Formal Techniques for Networked and Distributed Systems – FORTE 2007. XI, 375 pages. 2007.

Vol. 4556: C. Stephanidis (Ed.), Universal Access in Human-Computer Interaction, Part III. XXII, 1020 pages. 2007.

Vol. 4555: C. Stephanidis (Ed.), Universal Access in Human-Computer Interaction, Part II. XXII, 1066 pages. 2007.

Vol. 4554: C. Stephanidis (Ed.), Universal Acess in Human Computer Interaction, Part I. XXII, 1054 pages. 2007.

Vol. 4553: J.A. Jacko (Ed.), Human-Computer Interaction, Part IV. XXIV, 1225 pages. 2007.

Vol. 4552: J.A. Jacko (Ed.), Human-Computer Interaction, Part III. XXI, 1038 pages. 2007.

Vol. 4551: J.A. Jacko (Ed.), Human-Computer Interaction, Part II. XXIII, 1253 pages. 2007.

Vol. 4550: J.A. Jacko (Ed.), Human-Computer Interaction, Part I. XXIII, 1240 pages. 2007.

Vol. 4542: P. Sawyer, B. Paech, P. Heymans (Eds.), Requirements Engineering: Foundation for Software Quality. IX, 384 pages. 2007.

Vol. 4536: G. Concas, E. Damiani, M. Scotto, G. Succi (Eds.), Agile Processes in Software Engineering and Extreme Programming. XV, 276 pages. 2007.

Vol. 4530: D.H. Akehurst, R. Vogel, R.F. Paige (Eds.), Model Driven Architecture - Foundations and Applications. X, 219 pages. 2007.

Vol. 4523: Y.-H. Lee, H.-N. Kim, J. Kim, Y.W. Park, L.T. Yang, S.W. Kim (Eds.), Embedded Software and Systems. XIX, 829 pages. 2007.

Vol. 4498: N. Abdennahder, F. Kordon (Eds.), Reliable Software Technologies - Ada-Europe 2007. XII, 247 pages. 2007.

Vol. 4486: M. Bernardo, J. Hillston (Eds.), Formal Methods for Performance Evaluation. VII, 469 pages. 2007.

Vol. 4470: Q. Wang, D. Pfahl, D.M. Raffo (Eds.), Software Process Dynamics and Agility. XI, 346 pages. 2007.

Vol. 4468: M.M. Bonsangue, E.B. Johnsen (Eds.), Formal Methods for Open Object-Based Distributed Systems. X, 317 pages. 2007.

Vol. 4467: A.L. Murphy, J. Vitek (Eds.), Coordination Models and Languages. X, 325 pages. 2007.

Vol. 4454: Y. Gurevich, B. Meyer (Eds.), Tests and Proofs. IX, 217 pages. 2007.

Vol. 4444: T. Reps, M. Sagiv, J. Bauer (Eds.), Program Analysis and Compilation, Theory and Practice. X, 361 pages. 2007.

Vol. 4440: B. Liblit, Cooperative Bug Isolation. XV, 101 pages. 2007.

Vol. 4408: R. Choren, A. Garcia, H. Giese, H.-f. Leung, C. Lucena, A. Romanovsky (Eds.), Software Engineering for Multi-Agent Systems V. XII, 233 pages. 2007.

Vol. 4406: W. De Meuter (Ed.), Advances in Smalltalk. VII, 157 pages. 2007.

Vol. 4405: L. Padgham, F. Zambonelli (Eds.), Agent-Oriented Software Engineering VII. XII, 225 pages. 2007.

Vol. 4401: N. Guelfi, D. Buchs (Eds.), Rapid Integration of Software Engineering Techniques. IX, 177 pages. 2007.

Vol. 4385: K. Coninx, K. Luyten, K.A. Schneider (Eds.), Task Models and Diagrams for Users Interface Design. XI, 355 pages. 2007.

Vol. 4383: E. Bin, A. Ziv, S. Ur (Eds.), Hardware and Software, Verification and Testing. XII, 235 pages. 2007.

Vol. 4379: M. Südholt, C. Consel (Eds.), Object-Oriented Technology. VIII, 157 pages. 2007.

Vol. 4364: T. Kühne (Ed.), Models in Software Engineering. XI, 332 pages. 2007.

Vol. 4355: J. Julliand, O. Kouchnarenko (Eds.), B 2007: Formal Specification and Development in B. XIII, 293 pages. 2006.

Vol. 4354: M. Hanus (Ed.), Practical Aspects of Declarative Languages. X, 335 pages. 2006.

Vol. 4350: M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, All About Maude - A High-Performance Logical Framework. XXII, 797 pages. 2007.

Vol. 4348: S. Tucker Taft, R.A. Duff, R.L. Brukardt, E. Plödereder, P. Leroy, Ada 2005 Reference Manual. XXII, 765 pages. 2006.

Vol. 4346: L. Brim, B. Haverkort, M. Leucker, J. van de Pol (Eds.), Formal Methods: Applications and Technology. X, 363 pages. 2007.

Vol. 4344: V. Gruhn, F. Oquendo (Eds.), Software Architecture. X, 245 pages. 2006.

Vol. 4340: R. Prodan, T. Fahringer, Grid Computing. XXIII, 317 pages. 2007.

Vol. 4336: V.R. Basili, H.D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, R.W. Selby (Eds.), Empirical Software Engineering Issues. XVII, 193 pages. 2007.

Vol. 4326: S. Göbel, R. Malkewitz, I. Iurgel (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. X, 384 pages. 2006.

Vol. 4323: G. Doherty, A. Blandford (Eds.), Interactive Systems. XI, 269 pages. 2007.

Vol. 4322: F. Kordon, J. Sztipanovits (Eds.), Reliable Systems on Unreliable Networked Platforms. XIV, 317 pages. 2007.

Vol. 4309: P. Inverardi, M. Jazayeri (Eds.), Software Engineering Education in the Modern Age. VIII, 207 pages. 2006.

Vol. 4294: A. Dan, W. Lamersdorf (Eds.), Service-Oriented Computing – ICSOC 2006. XIX, 653 pages. 2006.

Vol. 4290: M. van Steen, M. Henning (Eds.), Middleware 2006. XIII, 425 pages. 2006.

Vol. 4279: N. Kobayashi (Ed.), Programming Languages and Systems. XI, 423 pages. 2006.

Vol. 4262: K. Havelund, M. Núñez, G. Roşu, B. Wolff (Eds.), Formal Approaches to Software Testing and Runtime Verification. VIII, 255 pages. 2006.

Vol. 4260: Z. Liu, J. He (Eds.), Formal Methods and Software Engineering. XII, 778 pages. 2006.

Vol. 4257: I. Richardson, P. Runeson, R. Messnarz (Eds.), Software Process Improvement. XI, 219 pages. 2006.

Vol. 4242: A. Rashid, M. Aksit (Eds.), Transactions on Aspect-Oriented Software Development II. IX, 289 pages. 2006.

Vol. 4229: E. Najm, J.-F. Pradat-Peyre, V.V. Donzeau-Gouge (Eds.), Formal Techniques for Networked and Distributed Systems - FORTE 2006. X, 486 pages. 2006.

Vol. 4227: W. Nejdl, K. Tochtermann (Eds.), Innovative Approaches for Learning and Knowledge Sharing. XVII, 721 pages. 2006.

Vol. 4218: S. Graf, W. Zhang (Eds.), Automated Technology for Verification and Analysis. XIV, 540 pages. 2006.

Vol. 4214: C. Hofmeister, I. Crnković, R. Reussner (Eds.), Quality of Software Architectures. X, 215 pages. 2006.

Vol. 4204: F. Benhamou (Ed.), Principles and Practice of Constraint Programming - CP 2006. XVIII, 774 pages. 2006.

Vol. 4199: O. Nierstrasz, J. Whittle, D. Harel, G. Reggio (Eds.), Model Driven Engineering Languages and Systems. XVI, 798 pages. 2006.

Vol. 4192: B. Mohr, J.L. Träff, J. Worringen, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. XVI, 414 pages. 2006.

Vol. 4184: M. Bravetti, M. Núñez, G. Zavattaro (Eds.), Web Services and Formal Methods. X, 289 pages. 2006.

Vol. 4166: J. Górski (Ed.), Computer Safety, Reliability, and Security. XIV, 440 pages. 2006.

Vol. 4158: L.T. Yang, H. Jin, J. Ma, T. Ungerer (Eds.), Autonomic and Trusted Computing. XIV, 613 pages. 2006.

Vol. 4157: M. Butler, C.B. Jones, A. Romanovsky, E. Troubitsyna (Eds.), Rigorous Development of Complex Fault-Tolerant Systems. X, 403 pages. 2006.

Preface

This volume contains the papers presented at ATVA 2007, the 5th International Symposium on Automated Technology for Verification and Analysis, which was held on October 22–25, 2007 at the National Center of Sciences in Tokyo, Japan.

The purpose of ATVA is to promote research on theoretical and practical aspects of automated analysis, verification and synthesis in East Asia by providing a forum for interaction between the regional and the international research communities and industry in the field. The first three ATVA symposia were held in 2003, 2004 and 2005 in Taipei, and ATVA 2006 was held in Beijing.

The program was selected from 88 submitted papers, with 25 countries represented among the authors. Of these submissions, 29 regular papers and 7 short papers were selected for inclusion in the program. In addition, the program included keynote talks and tutorials by Martin Abadi (University of California, Santa Cruz and Microsoft Research), Ken McMillan (Cadence Berkeley Labs), and Moshe Vardi (Rice University), and an invited talk by Atsushi Hasegawa (Renesas Technology). A workshop on Omega-Automata (OMEGA 2007) was organized in connection with the conference.

ATVA 2007 was sponsored by the National Institute of Informatics, the Kayamori Foundation of Information Science Advancement, the Inoue Foundation for Science, and the Telecommunications Advancement Foundation. We are grateful for their support.

We would like to thank the program committee and the reviewers for their hard work and dedication in putting together this program. We would like to thank the Steering Committee for their considerable help with the organization of the conference. We also thank Michihiro Koibuchi for his help with the local arrangements.

October 2007

Kedar Namjoshi
Tomohiro Yoneda
Teruo Higashino
Yoshio Okamura

Conference Organization

General Chairs

Teruo Higashino	Osaka University, Japan
Yoshio Okamura	STARC, Japan

Program Chairs

Kedar S. Namjoshi	Bell Labs, USA
Tomohiro Yoneda	National Institute of Informatics, Japan

Program Committee

Rajeev Alur	University of Pennsylvania
Christel Baier	University of Dresden
Jonathan Billington	University of South Australia
Sung-Deok Cha	Korea Advanced Inst. of Sci. and Techn.
Ching-Tsun Chou	Intel
Jin Song Dong	National University of Singapore
E. Allen Emerson	University of Texas at Austin
Masahiro Fujita	University of Tokyo
Susanne Graf	VERIMAG
Wolfgang Grieskamp	Microsoft Research
Aarti Gupta	NEC Labs America
Teruo Higashino	Osaka University
Kiyoharu Hamaguchi	Osaka University
Moonzoo Kim	KAIST
Orna Kupferman	Hebrew University
Robert P. Kurshan	Cadence
Insup Lee	University of Pennsylvania
Xuandong Li	Nanjing University
Shaoying Liu	Hosei University
Zhiming Liu	IIST/United Nations University
Mila E. Majster-Cederbaum	University of Mannheim
Shin Nakajima	National Institute of Informatics
Akio Nakata	Hiroshima City University
Kedar S. Namjoshi	Bell Labs
Mizuhito Ogawa	JAIST
Olaf Owe	University of Oslo
Doron A. Peled	University of Warwick and Bar Ilan University
Mike Reed	UNU-IIST, Macao

Hiroyuki Seki	NAIST
Xiaoyu Song	Portland State University
Yih-Kuen Tsay	National Taiwan University
Irek Ulidowski	University of Leicester
Bow-Yaw Wang	Academia Sinica
Farn Wang	National Taiwan University
Yi Wang	Uppsala University
Baowen Xu	Southeast University of China
Hsu-Chun Yen	National Taiwan University
Tomohiro Yoneda	National Institiute of Informatics
Shoji Yuen	Nagoya University
Wenhui Zhang	Chinese Academy of Sciences
Lenore Zuck	University of Illinois at Chicago

Steering Committee

E. Allen Emerson	University of Texas at Austin, USA
Oscar H. Ibarra	University of California, Santa Barbara, USA
Insup Lee	University of Pennsylvania, USA
Doron A. Peled	University of Warwick, UK and Bar Ilan University, Israel
Farn Wang	National Taiwan University, Taiwan
Hsu-Chun Yen	National Taiwan University, Taiwan

Referees

Benjamin Aminof	Johan Dovland	Yunho Kim
Madhukar Anand	Arvind Easwaran	Dmitry Korchemny
David Arney	Sebastian Fischmeister	Piotr Kosiuczenko
Colin Atkinson	Felix Freiling	Pavel Krcal
Louise Avila	Carsten Fritz	Keiichirou Kusakari
Syed Mahfuzul Aziz	Guy Gallasch	Marcel Kyas
Noomene Ben Henda	Malay Ganai	Yuan Fang Li
Domagoj Babic	Jim Grundy	Guoqiang Li
Hanene Ben-Abdallah	Yi Hong	Nimrod Lilith
Armin Biere	Reiko Heckel	Xinxin Liu
Lei Bu	Monika Heiner	Lin Liu
Lin-Zan Cai	Nao Hirokawa	Chi-Jian Luo
Wen-Chin Chan	Geng-Dian Huang	Yoad Lustig
Yu-Fang Chen	John Håkansson	Michael J. May
Chunqing Chen	Keigo Imai	Christoph Minnameier
Zhenbang Chen	Franjo Ivancic	Van Tang Nguyen
Chang-beom Choi	Einar Broch Johnsen	Peter Csaba Olveczky
Jyotirmoy Deshmukh	Vineet Kahlon	Geguang Pu
Nikhil Dinesh	Yuichi Kaji	Zvonimir Rakamaric

Roopsha Samanta
Gerardo Schneider
Nishant Sinha
Martin Steffen
Volker Stolz
Ryo Suetsugu
Jun Sun
Yoshiaki Takata
Murali Talupur
Kai-Fu Tang
Ming-Hsien Tsai
Emilio Tuosto
Kazunori Ueda
Thomas Wahl
Chao Wang
Verena Wolf
Rong-Shiun Wu
Cong Yuan
Naijun Zhan
Miaomiao Zhang
Jianhua Zhao

Table of Contents

Invited Talks

Regular Papers

Short Papers

Policies and Proofs for Code Auditing

Nathan Whitehead[1], Jordan Johnson[1], and Martín Abadi[1,2]

[1] University of California, Santa Cruz
[2] Microsoft Research

Abstract. Both proofs and trust relations play a role in security decisions, in particular in determining whether to execute a piece of code. We have developed a language, called BCIC, for policies that combine proofs and trusted assertions about code. In this paper, using BCIC, we suggest an approach to code auditing that bases auditing decisions on logical policies and tools.

1 Introduction

Deciding to execute a piece of software can have substantial security implications. Accordingly, a variety of criteria and techniques have been proposed and deployed for making such decisions. These include the use of digital signatures (as in ActiveX [12]) and of code analysis (as in typed low-level languages [5, 9, 10]). The digital signatures can be the basis of practical policies that reflect trust relations—for instance, the trust in certain software authors or distributors. The code analysis can lead to proofs, and thereby to proof-carrying code [11]. Unfortunately, neither trust relations nor proofs are typically sufficient on their own. Trust can be wrong, and code analysis is seldom comprehensive.

We are developing a system for defining and evaluating policies that combine proofs and trusted assertions about code [18, 19, 20]. The core of the system is a logical query language, called BCIC. BCIC is a combination of Binder [4], a logic-programming language for security policies in distributed systems, with Coq's Calculus of Inductive Constructions (CIC) [3], a general-purpose proof framework.

Whereas the focus of most previous work (including our own) is on the decision to execute pieces of code, similar considerations arise in other situations. For instance, from a security perspective, installing a piece of code can be much like executing it. Further upstream, auditing code is also critical to security. Auditing can complement other techniques for assurance, in the course of software production or at various times before execution. Although humans perform the auditing, they are often guided by policies (e.g., what aspects of the code should be audited) and sometime supported by tools (e.g., for focusing attention on questionable parts of the code).

In this paper, using BCIC, we suggest an approach to code auditing that bases auditing decisions on logical policies and tools. Specifically, we suggest that policies for auditing may be expressed in BCIC and evaluated by logical means. Thus, this approach leverages trust relations and proofs, but it also allows

K.S. Namjoshi et al. (Eds.): ATVA 2007, LNCS 4762, pp. 1–14, 2007.

auditing to complement them. We recognize that this approach is still theoretical and probably incomplete. Nevertheless, it emphasizes the possibility of looking at techniques for verification and analysis in the context of policy-driven systems, and the attractiveness of doing so in a logical setting.

We present two small examples. The first example concerns operating system calls from an application extension language. With a BCIC policy, every operating system call must be authorized by an audit. A policy rule can allow entire classes of calls without separate digital signatures from an authority. In the second example, we consider an information-flow type system [14], specifically a type system that tracks trust (much like Perl's taint mode [6], but statically) due to Ørbæk and Palsberg [13]. The type system includes a form of declassification, in which expressions can be coerced to be trusted (that is, untainted). If any program could use declassification indiscriminately, then the type system would provide no benefit. With a BCIC policy, a trusted authority must authorize each declassification. In both examples, security decisions can rely on nuanced, fine-grained combinations of reason and authority.

We treat these examples in Sections 2 and 3, respectively. We consider implementation details in Section 4. We conclude in Section 5.

2 Example: Auditing Function Calls

In this example we consider the calling behavior of programs in a managed environment of libraries. A base application may allow extensions that provide additional functionality not only to the user but also to other extensions. The extensions may come from many different sources, and accordingly they may be trusted to varying extents. By constraining calls, the security policy can selectively allow different functionality to different extensions.

2.1 Language

For simplicity, we study an interpreted extension language. Specifically, we use an untyped λ-calculus with a special `call` construct that represents operating system calls and calls to other libraries. All calls take exactly one argument, which they may ignore. In order to allow primitive data types, we also include a representation for data constructors (`constr0`, `constr1`, and `constr2`, for constructors that take zero, one, and two arguments respectively). These constructors are enough to handle all the data types that appear in our implementation, including natural numbers, pairs, and lists. Destructors have no special syntax, but are included among the calls.

In Coq notation [2, 3, 16], the syntax of the language is:

```
Inductive exp : Set :=
| var : nat -> exp
| abs : exp -> exp
| app : exp -> exp -> exp
```

```
| call : funcname -> exp -> exp
| constr0 : constrname -> exp
| constr1 : constrname -> exp -> exp
| constr2 : constrname -> exp -> exp -> exp.
```

A detailed knowledge of Coq is not required for understanding this and other definitions in this paper. This definition introduces a class of expressions, inductively by cases with a type for each case; expressions rely on De Bruijn notation, so variables are numbered and binding occurrences of variables are unnecessary. Similarly, other definitions introduce other classes of expressions and propositions, and some parameters for them.

A policy can decide which calls any piece of code may execute. The policy can be expressed in terms of a parameter `audit_maycall`.

```
Parameter audit_maycall : exp -> funcname -> Prop.
```

According to this type, every audit requirement mentions the entire program `exp` that is the context of the audit. Mentioning a subexpression in isolation would not always be satisfactory, and it may be dangerous, as the effects of a subexpression depend on context. The audit requirement also mentions the name of the function being called. We omit any restrictions on the arguments to the function, in order to make static reasoning easier; we assume that the callee does its own checking of arguments. (Section 3 says more on going further with static analysis.)

The predicate `audited_calls` indicates that a piece of code has permission to make all the calls that it could make. This predicate is defined inductively by:

```
Inductive audited_calls : exp -> exp -> Prop :=
| audited_calls_var :
  forall e n,
    audited_calls e (var n)
| audited_calls_app :
  forall e e1 e2,
    audited_calls e e1 ->
    audited_calls e e2 ->
    audited_calls e (app e1 e2)
| audited_calls_abs :
  forall e e1,
    audited_calls e e1 ->
    audited_calls e (abs e1)
| audited_calls_call :
  forall f e e1,
    audited_calls e e1 ->
    audit_maycall e f ->
    audited_calls e (call f e1)
```