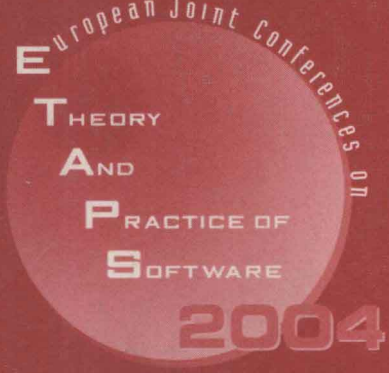
David Schmidt (Ed.)

LNCS 2986

Programming Languages and Systems

13th European Symposium on Programming, ESOP 2004
Held as Part of the Joint European Conferences
on Theory and Practice of Software, ETAPS 2004
Barcelona, Spain, March/April 2004, Proceedings



Springer

David Schmidt (Ed.)

Programming Languages and Systems

13th European Symposium on Programming, ESOP 2004
Held as Part of the Joint European Conferences
on Theory and Practice of Software, ETAPS 2004
Barcelona, Spain, March 29 – April 2, 2004
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editor

David Schmidt
Kansas State University, Department of Computing and Information Sciences
234 Nichols Hall, Manhattan, KS 66506, USA
E-mail: schmidt@cis.ksu.edu

Library of Congress Control Number: 2004102322

CR Subject Classification (1998): D.3, D.1, D.2, F.3, F.4, E.1

ISSN 0302-9743
ISBN 3-540-21313-9 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10994481 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 2986
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Foreword

ETAPS 2004 was the seventh instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference that was established in 1998 by combining a number of existing and new conferences. This year it comprised five conferences (FOSSACS, FASE, ESOP, CC, TACAS), 23 satellite workshops, 1 tutorial, and 7 invited lectures (not including those that are specific to the satellite events).

The events that comprise ETAPS address various aspects of the system development process, including specification, design, implementation analysis and improvement. The languages, methodologies and tools that support these activities are all well within its scope. Different blends of theory and practice are represented, with an inclination towards theory with a practical motivation on the one hand and soundly based practice on the other. Many of the issues involved in software design apply to systems in general, including hardware systems, and the emphasis on software is not intended to be exclusive.

ETAPS is a loose confederation in which each event retains its own identity, with a separate program committee and independent proceedings. Its format is open-ended, allowing it to grow and evolve as time goes by. Contributed talks and system demonstrations are in synchronized parallel sessions, with invited lectures in plenary sessions. Two of the invited lectures are reserved for "unifying" talks on topics of interest to the whole range of ETAPS attendees. The aim of cramming all this activity into a single one-week meeting is to create a strong magnet for academic and industrial researchers working on topics within its scope, giving them the opportunity to learn about research in related areas, and thereby to foster new and existing links between work in areas that were formerly addressed in separate meetings.

ETAPS 2004 was organized by the LSI Department of the Catalonia Technical University (UPC), in cooperation with:

European Association for Theoretical Computer Science (EATCS)
European Association for Programming Languages and Systems (EAPLS)
European Association of Software Science and Technology (EASST)
ACM SIGACT, SIGSOFT and SIGPLAN

The organizing team comprised

Jordi Cortadella (Satellite Events), Nikos Mylonakis, Robert Nieuwenhuis, Fernando Orejas (Chair), Edelmira Pasarella, Sonia Perez, Elvira Pino, Albert Rubio

and had the assistance of TILESA OPC.
ETAPS 2004 received generous sponsorship from:

UPC, Spanish Ministry of Science and Technology (MCYT), Catalan Department for Universities, Research and Information Society (DURSI), IBM, Intel.

Overall planning for ETAPS conferences is the responsibility of its Steering Committee, whose current membership is:

Ratislav Bodik (Berkeley), Maura Cerioli (Genoa), Evelyn Duesterwald (IBM, Yorktown Heights), Hartmut Ehrig (Berlin), José Fiadeiro (Leicester), Marie-Claude Gaudel (Paris), Andy Gordon (Microsoft Research, Cambridge), Roberto Gorrieri (Bologna), Nicolas Halbwachs (Grenoble), Gûrel Hedin (Lund), Kurt Jensen (Aarhus), Paul Klint (Amsterdam), Tiziana Margaria (Dortmund), Ugo Montanari (Pisa), Hanne Riis Nielson (Copenhagen), Fernando Orejas (Barcelona), Mauro Pezzè (Milan), Andreas Podelski (Saarbrücken), Mooly Sagiv (Tel Aviv), Don Sannella (Edinburgh), Vladimiro Sassone (Sussex), David Schmidt (Kansas), Bernhard Steffen (Dortmund), Perdita Stevens (Edinburgh), Andrzej Tarlecki (Warsaw), Igor Walukiewicz (Bordeaux), Michel Wermelinger (Lisbon)

I would like to express my sincere gratitude to all of these people and organizations, the program committee chairs and PC members of the ETAPS conferences, the organizers of the satellite events, the speakers themselves, and finally Springer-Verlag for agreeing to publish the ETAPS proceedings. This year, the number of submissions approached 600, making acceptance rates fall to 25%. I congratulate the authors who made it into the final program! I hope that all the other authors still found a way of participating in this exciting event and I hope you will continue submitting.

In 2005, ETAPS will be organized by Don Sannella in Edinburgh. You will be welcomed by another "local": my successor as ETAPS Steering Committee Chair – Perdita Stevens. My wish is that she will enjoy coordinating the next three editions of ETAPS as much as I have. It is not an easy job, in spite of what Don assured me when I succeeded him! But it is definitely a very rewarding one. One cannot help but feel proud of seeing submission and participation records being broken one year after the other, and that the technical program reached the levels of quality that we have been witnessing. At the same time, interacting with the organizers has been a particularly rich experience. Having organized the very first edition of ETAPS in Lisbon in 1998, I knew what they were going through, and I can tell you that each of them put his/her heart, soul, and an incredible amount of effort into the organization. The result, as we all know, was brilliant on all counts! Therefore, my last words are to thank Susanne Graf (2002), Andrzej Tarlecki and Paweł Urzyczyn (2003), and Fernando Orejas (2004) for the privilege of having worked with them.

Leicester, January 2004

José Luiz Fiadeiro
ETAPS Steering Committee Chairman

Preface

This volume contains the 28 papers presented at ESOP 2004, the 13th European Symposium on Programming, which took place in Barcelona, Spain, March 29–31, 2004. The ESOP series began in 1986 with the goal of bridging the gap between theory and practice, and the conferences continue to be devoted to explaining fundamental issues in the specification, analysis, and implementation of programming languages and systems.

The volume begins with a summary of an invited contribution by Peter O'Hearn, titled *Resources, Concurrency and Local Reasoning*, and continues with the 27 papers selected by the Program Committee from 118 submissions.

Each submission was reviewed by at least three referees, and papers were selected during a ten-day electronic discussion phase. I would like to sincerely thank the members of the Program Committee, as well as their subreferees, for their diligent work; Torben Amtoft, for helping me collect the papers for the proceedings; and Tiziana Margaria, Bernhard Steffen, and their colleagues at METAFrame, for the use of their conference management software.

David Schmidt
Program Chair
ESOP 2004

Organization

Program Committee

Torben Amtoft	Kansas State University (USA)
Henri Bal	Vrije Universiteit (Netherlands)
Radhia Cousot	École Polytechnique (France)
Pierpaolo Degano	Università di Pisa (Italy)
Mariangiola Dezani-Ciancaglini	Università di Torino (Italy)
Pascal Fradet	INRIA Rhône-Alpes (France)
Michael Gordon	University of Cambridge (UK)
Eric Goubault	CEA/Saclay (France)
Joshua Guttman	MITRE (USA)
Fritz Henglein	University of Copenhagen (Denmark)
Matthew Hennessy	University of Sussex (UK)
Markus Müller-Olm	Universität Hagen (Germany)
Martin Odersky	École Polytechnique Fédérale de Lausanne (Switzerland)
German Puebla	Technical University of Madrid (Spain)
David Schmidt	Kansas State University (USA)
Michael Schwartzbach	University of Aarhus (Denmark)
Harald Søndergaard	University of Melbourne (Australia)
Peter Thiemann	Universität Freiburg (Germany)
Mitchell Wand	Northeastern University (USA)
Kwangkeun Yi	Seoul National University (Korea)

Referees

Mads Sig Ager, Elvira Albert, Thomas Ambus, Ki-yung Ahn, Ricardo Baeza-Yates, Roberto Bagnara, Harald James Bailey, Steffen Bakel, Massimo Bartoletti, Josh Berdine, Martin Berger, Frederic Besson, Massimo Bartoletti, Lorenzo Bettini, Bruno Blanchet, Chiara Bodei, Viviana Bono, Mikael Buchholtz, Michele Bugliesi, Emir Burak, Lilian Burdy, Marzia Buscemi, Daniel Cabeza, Luis Caires, Manuel Carro, Rohit Chadha, Manuel Chakravarty, Byeong-Mo Chang, Aske Simon Chistensen, Woongsik Choi, Damien Ciabrini, Ana Cristina Vieira de Melo, Mario Coppo, Andrea Corradini, Patrick Cousot, Silvia Crafa, Vincent Cremet, Michele Curti, Vincent Danos, Kyung-Goo Doh, Guillaume Dufay, Hyunjun Eo, Francois Fages, Manuel Fähndrich, Matthias Felleisen, Jérôme Feret, Mary Fernandez, GianLuigi Ferrari, Kathleen Fisher, Matthew Flatt, Riccardo Focardi, Cedric Fournet, Alain Frisch, Carsten Führmann, Fabio Gadducci, Vladimir Gapeyev, Martin Gasbichler, Thomas Genet, Roberto Giacobazzi, Paola Giannini, Mariangiola Elio Giovannetti, Alain Girault, Roberto Gorrieri, Jean Goubault-Larrecq, Dick Grune, Stefano Guerrini, John Hatcliff, Simon Helsen, Fergus Henderson, Angel Herranz, Christoph Herrmann, Jonat-

han Herzog, Michael Thomas Hildebrandt, Ralf Hinze, Daniel Hirschkoff, Tom Hirschowitz, Frank Huch, Paul Hudak, Sebastian Hunt, Charles Hymans, Ceriel Jacobs, Alan Jeffrey, Johan Jeuring, Neil Jones, Hyun-Goo Kang, Jeroen Ketema, Iksoon Kim, Andy King, Christian Kirkegaard, Oleg Kiselyov, Dexter Kozen, Karl Krukow, Ralf Laemmel, Cosimo Laneve, Oukseh Lee, Ugo de' Liguoro, Jim Lipton, Francesco Logozzo, Michele Loreti, Henning Makholm, Julio Mariño, Matthieu Martel, Damien Massé, Laurent Mauborgne, Richard Mayr, Edison Mera, Nikolay Mihaylov, Dale Miller, Antoine Miné, Alberto Momigliano, Benjamin Monate, David Monniaux, Anders Møller, Jose F. Morales, Peter Müller, Harald Lee Naish, George Necula, Matthias Neubauer, Oliver Niese, Henrik Nilsson, Henning Niss, Thomas Noll, Peter O'Hearn, Peter Padawitz, Benjamin Pierce, Adolfo Piperno, Marco Pistore, Enrico Pontelli, Bernard Pope, Rosario Pugliese, John Ramsdell, Femke Raamsdonk, Olivier Ridoux, Xavier Rival, Simona Ronchi Della Rocca, Andreas Rossberg, Oliver Rüthing, Eric Rutten, Luca Roversi, Jean-Claude Royer, Sukyoung Ryu, Jaime Sanchez-Hernandez, Dave Sands, Vladimiro Sassone, Helmut Seidl, Harald R. Sekar, Sunae Seo, Alexander Serebrenik, Manuel Serrano, Wendelin Serwe, Peter Sestoft, Peter Sewell, Nikolay Shilov, Seung-Cheol Shin, Detlef Sieling, Axel Simon, Élodie-Jane Sims, Jan-Georg Smaus, Christian Stefansen, Erik Stenman, Peter Stuckey, Josef Svenningsson, Vipin Swarup, Nik Swoboda, Lucy Mari Tabuti, Javier Thayer, Simone Tini, Franklyn Turbak, Pawel Urzyczyn, Frank D. Valencia, Vasco Vasconcelos, Roel de Vrijer, Joe Wells, Hongseok Yang, Nobuko Yoshida, Matthias Zenger, Roberto Zunino

Table of Contents

Resources, Concurrency, and Local Reasoning (Abstract)

Peter W. O'Hearn

Queen Mary, University of London

In the 1960s Dijkstra suggested that, in order to limit the complexity of potential process interactions, concurrent programs should be designed so that different processes behave independently, except at rare moments of synchronization [3]. Then, in the 1970s Hoare and Brinch Hansen argued that debugging and reasoning about concurrent programs could be considerably simplified using compiler-enforceable syntactic constraints that preclude interference [4,1]; scope restrictions were described which had the effect that all process interaction was mediated by a critical region or monitor. Based on such restrictions Hoare described proof rules for shared-variable concurrency that were beautifully modular [4]: one could reason locally about a process, and simple syntactic checks ensured that no other process could tamper with its state in a way that invalidated the local reasoning.

The major problem with Hoare and Brinch Hansen's proposal is that its scope-based approach to resource separation is too restrictive for many realistic programs. It does not apply to flexible but unstructured constructs such as semaphores, and the syntactic checks it relies on are insufficient to rule out interference in the presence of pointers and aliasing. Proof methods were subsequently developed which allow reasoning in the presence of interference [9,10,5], and the reasoning that they support is much more powerful that that of [4], but also much less modular.

There is thus a mismatch, between the intuitive basis of concurrent programming with resources, where separation remains a vital design idea, and formal techniques for reasoning about such programs, where methods based on separation are severely limited. The purpose of this work is to revisit these issues, using the recent formalism of separation logic [11]. The general point is that by using a logic of resources [7] rather than syntactic constraints we can overcome the limitations of scope-based approaches, while retaining their modular character. We describe a variation on the proof rules of Hoare for contitional critical regions, using the "separating conjunction" connective to preclude pointer-based interference. With the modified rules we can at once handle many examples where a linked data structure rather than, say, an array is used within a process, or within a data abstraction that mediates interprocess interaction.

The rules have a further interesting effect when a data abstraction keeps track of pointers as part of its data, rather than just as part of its implementation. The separating conjunction allows us to partition memory in a dynamically reconfigurable way, extending the static partioning done by critical regions or monitors when there is no heap. This enables us to handle a number of subtler

D.A. Schmidt (Ed.): ESOP 2004, LNCS 2986, pp. 1–2, 2004.

programs, where a pointer is transferred from one process to another, or between a process and a monitor, and the ownership of the storage pointed to transfers with it. Ownership transfer is common in systems programs. For example, interaction with a memory manager results in pieces of storage transferring between the manager and its clients as allocation and deallocation operations are performed [8]. Another typical example is efficient message passing, where a pointer is transferred from one process to another in order to avoid copying large pieces of data.

Dynamic partitioning turns out also to be the key to treating lower level, unstructured constructs which do not use explicit critical regions. In particular, our formalism supports a view of semaphores as ownership transformers, that (logically) release and seize portions of storage in addition to their (operational) behaviour as counting-based synchronizers. Local reasoning [6] is possible in such a situation because dynamic, non scope-based, uses of semaphores to protect resources are matched by the dynamic, non scope-based, approach to resource separation provided by separation logic.

A special role in this work is played by "precise" assertions, which are ones that unambiguously specify a portion of storage (an assertion is precise if, for any given heap, there is at most one subheap that satisfies it). Precision is essential for the soundness of the proof rules, which work by decomposing the state in a system into that owned by various processes and resources (or monitors). Precise assertions fulfill a similar role in recent work on information hiding [8], and are used by Brookes in his semantic analysis of our concurrency proof rules [2].

References

[1] P. Brinch Hansen. *Operating System Principles.* Prentice Hall, 1973.
[2] S. D. Brookes. A semantics for concurrent separation logic. Draft of 7/25/03, 2003.
[3] E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, 1968.
[4] C. A. R. Hoare. Towards a theory of parallel programming. In Hoare and Perrot, editors, *Operating Systems Techniques.* Academic Press, 1972.
[5] C. B. Jones. Specification and design of (parallel) programs. *IFIP Conference*, 1983.
[6] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *Proceedings of 15th Annual Conference of the European Association for Computer Science Logic*, LNCS, pages 1–19. Springer-Verlag, 2001.
[7] P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 99.
[8] P. W. O'Hearn, H. Yang, and J. C. Reynolds. Separation and information hiding. In *31st POPL*, pages 268–280, Venice, January 2004.
[9] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, (19):319–340, 1976.
[10] A. Pnueli. The temporal semantics of concurrent programs. Theoretical Computer Science, 13(1), 45–60, 1981.
[11] J. C. Reynolds. Separation logic: a logic for shared mutable data structures. Invited Paper, LICS'02, 2002.

Relational Abstract Domains for the Detection of Floating-Point Run-Time Errors*

Antoine Miné

DI-École Normale Supérieure de Paris, France,
mine@di.ens.fr

Abstract. We present a new idea to adapt relational abstract domains to the analysis of IEEE 754-compliant floating-point numbers in order to statically detect, through Abstract Interpretation-based static analyses, potential floating-point run-time exceptions such as overflows or invalid operations. In order to take the non-linearity of rounding into account, expressions are modeled as linear forms with interval coefficients. We show how to extend already existing numerical abstract domains, such as the octagon abstract domain, to efficiently abstract transfer functions based on interval linear forms. We discuss specific fixpoint stabilization techniques and give some experimental results.

1 Introduction

It is a well-established fact, since the failure of the Ariane 5 launcher in 1996, that run-time errors in critical embedded software can cause great financial—and human—losses. Nowadays, embedded software is becoming more and more complex. One particular trend is to abandon fixed-point arithmetics in favor of floating-point computations. Unfortunately, floating-point models are quite complex and features such as rounding and special numbers (infinities, NaN, etc.) are not always understood by programmers. This has already led to catastrophic behaviors, such as the Patriot missile story told in [16].

Much work is concerned about the *precision* of the computations, that is to say, characterizing the amount and cause of drift between a computation on perfect reals and the corresponding floating-point implementation. Ours is *not*. We seek to prove that an exceptional behavior (such as division by zero or overflow) will not occur in any execution of the analyzed program. While this is a simpler problem, our goal is to scale up to programs of hundreds of thousands of lines with full data coverage and very few (even none) false alarms.

Our framework is that of Abstract Interpretation, a generic framework for designing sound static analyses that already features many instances [6,7]. We adapt existing relational numerical abstract domains (generally designed for the analysis of integer arithmetics) to cope with floating-point arithmetics. The need for such domains appeared during the successful design of a commissioned special-purpose prototype analyzer for a critical embedded avionics

* This work was partially supported by the ASTRÉE RNTL project.

D.A. Schmidt (Ed.): ESOP 2004, LNCS 2986, pp. 3–17, 2004.

system. Interval analysis, used in a first prototype [3], proved too coarse because error-freeness of the analyzed code depends on tests that are inherently poorly abstracted in non-relational abstract domains. We also had to design special-purpose widenings and narrowings to compensate for the pervasive rounding errors, not only in the analyzed program, but also introduced by our efficient abstractions. These techniques were implemented in our second prototype whose overall design was presented in [4]. The present paper focuses on improvements and novel unpublished ideas; it is also more generic.

2 Related Work

Abstract Domains. A key component in Abstract-Interpretation-based analyses is the abstract domain which is a computer-representable class of program invariants together with some operators to manipulate them: transfer functions for guards and assignments, a control-flow join operator, and fixpoint acceleration operators (such as widenings ∇ and narrowings Δ) aiming at the correct and efficient analysis of loops. One of the simplest yet useful abstract domain is the widespread interval domain [6]. Relational domains, which are more precise, include Cousot and Halbwachs's polyhedron domain [8] (corresponding to invariants of the form $\sum c_i v_i \leq c$), Miné's octagon domain [14] ($\pm v_i \pm v_j \leq c$), and Simon's two variables per inequality domain [15] ($\alpha v_i + \beta v_j \leq c$). Even though the underlying algorithms for these relational domains allow them to abstract sets of reals as well as sets of integers, their efficient implementation—in a maybe approximate but sound way—using floating-point numbers remains a challenge. Moreover, these relational domains do not support abstracting floating-point expressions, but only expressions on perfect integers, rationals, or reals.

Floating-Point Analyses. Much work on floating-point is dedicated to the analysis of the precision of the computations and the origins of the rounding errors. The CESTAC method [17] is widely used, but also much debated as it is based on a probabilistic model of error distribution and thus cannot give sound answers. An interval-based Abstract Interpretation for error terms is proposed in [1]. Some authors [11,13] go one step further by allowing error terms to be related in relational, even non-linear, domains. Unfortunately, this extra precision does not help when analyzing programs whose correctness also depends upon relations between variables and not only error terms (such as programs with inequality tests, as in Fig. 3).

Our Work. We first present our IEEE 754-based computation model (Sect. 3) and recall the classical interval analysis adapted to floating-point numbers (Sect. 4). We present, in Sect. 5, an abstraction of floating-point expressions in terms of interval linear forms over the real field and use it to refine the interval domain. Sect. 6 shows how some relational abstract domains can be efficiently adapted to work on these linear forms. Sect. 7 presents adapted widening and narrowing techniques. Finally, some experimental results are shown in Sect. 8.

3 IEEE 754-Based Floating-Point Model

We present in this section the concrete floating-point arithmetics model that we wish to analyze and which is based on the widespread IEEE 754-1985 [5] norm.

3.1 IEEE 754 Floating-Point Numbers

The binary representation of a IEEE 754 number is composed of three fields:

- a 1-bit sign s;
- an exponent $e - \mathbf{bias}$, represented by a biased $\mathbf{e}$-bit unsigned integer e;
- a fraction $f = .b_1 \ldots b_\mathbf{p}$, represented by a $\mathbf{p}$-bit unsigned integer.

The values $\mathbf{e}$, $\mathbf{bias}$, and $\mathbf{p}$ are format-specific. We will denote by $\mathbf{F}$ the set of all available formats and by $\mathbf{f} = \mathbf{32}$ the 32-bit *single* format ($\mathbf{e} = 8$, $\mathbf{bias} = 127$, and $\mathbf{p} = 23$). Floating-point numbers belong to one of the following categories:

- *normalized* numbers $(-1)^s \times 2^{2-\mathbf{bias}} \times 1.f$, when $1 \leq e \leq 2^\mathbf{e} - 2$;
- *denormalized* numbers $(-1)^s \times 2^{1-\mathbf{bias}} \times 0.f$, when $e = 0$ and $f \neq 0$;
- $+0$ or -0 (depending on s), when $e = 0$ and $f = 0$;
- $+\infty$ or $-\infty$ (depending on s), when $e = 2^\mathbf{e} - 1$ and $f = 0$;
- error codes (so-called *NaN*), when $e = 2^\mathbf{e} - 1$ and $f \neq 0$.

For each format $\mathbf{f} \in \mathbf{F}$ we define in particular:

- $mf_\mathbf{f} = 2^{1-\mathbf{bias}-\mathbf{p}}$ the smallest non-zero positive number;
- $Mf_\mathbf{f} = (2 - 2^{-\mathbf{p}})2^{2^\mathbf{e}-\mathbf{bias}-2}$, the largest non-infinity number.

The special values $+\infty$ and $-\infty$ may be generated as a result of operations undefined on $\mathbb{R}$ (such as $1/+0$), or when a result's absolute value overflows $Mf_\mathbf{f}$. Other undefined operations (such as $+0/+0$) result in a *NaN* (that stands for *Not A Number*). The sign of 0 serves only to distinguish between $1/+0 = +\infty$ and $1/-0 = -\infty$; $+0$ and -0 are indistinguishable in all other contexts (even comparison).

Due to the limited number of digits, the result of a floating-point operation needs to be rounded. IEEE 754 provides four rounding modes: towards 0, towards $+\infty$, towards $-\infty$, and to nearest. Depending on this mode, either the floating-point number directly smaller or directly larger than the exact real result is chosen (possibly $+\infty$ or $-\infty$). Rounding can build infinities from non-infinities operands (this is called *overflow*), and it may return zero when the absolute value of the result is too small (this is called *underflow*). Because of this rounding phase, most algebraic properties of $\mathbb{R}$, such as associativity and distributivity, are lost. However, the opposite of a number is always exactly represented (unlike what happens in two-complement integer arithmetics), and comparison operators are also exact. See [10] for a description of the classical properties and pitfalls of the floating-point arithmetics.

3.2 Custom Floating-Point Computation Model

We focus our analysis on the large class of programs that treat floating-point arithmetics as a practical approximation to the mathematical reals $\mathbb{R}$: roundings and underflows are tolerated, but not overflows, divisions by zero or invalid operations, which are considered run-time errors and halt the program. Our goal is to detect such behaviors. In this context, $+\infty$, $-\infty$, and *NaN*s can never