OCAML

One such functional language is OCaml, an industrial-strength programming language developed at the Inria Paris Research Centre. OCaml is used to implement software such as proof assistants, automated solvers, and compilers in industry throughout the world by organisations including Facebook, Bloomberg, and Issuu. OCaml is a multi-paradigm language supporting functional, imperative, and object-oriented programming. It has a robust pattern-matching mechanism, a flexible module system, and automatic memory management, so it can be used to write efficient, modular programs that maintain type-safety (preventing type errors caused by discrepancies between differing data types) and deals with side effects such as memory mutation. Mathematical reasoning and modelling techniques to describe and certify parts of computer infrastructures and provide formal software verification. In line with this practice, Dr Mário Pereira and Dr António Ravara from the Nova School of Science and Technology, Lisbon, and their collaborators have developed the Cameleer tool, a formal verification software tool for OCaml-written code.

DEDUCTIVE SOFTWARE VERIFICATION

Deductive software verification involves expressing the correctness of code into a set of mathematical statements, known as verification conditions, and then proving them. Pereira explains how this includes a logical specification that is attached to the code. A logical specification is a description, written in a mathematical language, of the properties that a program must respect at certain points of its execution. Finally, a verification conditions generator algorithm is implemented. This takes the code together with its specification and produces the verification conditions. Proving these mathematical statements formally establishes that the program execution obeys the logical properties described in its specification. Pereira explains that, despite the advances in deductive verification and proof automation in past decades, the family of functional languages used to construct programs, by applying and composing functions, has received little attention.

The best solution involves including robust mathematical guarantees in the software development process. Even though these factors make OCaml code a good fit for formal verification, deductive verification has rarely been used with OCaml programs, explains Pereira. Moreover, prior to the Cameleer project, an automated verification tool that can deal directly with code written in OCaml did not exist. Until now, OCaml programmers choosing to employ proof automation have had to learn a verification-aware language, write programs in it, and then perform code extraction. The alternative was to use tools that necessitated manual proof assistance.

GOSPEL

A specification language is a formal language used during software development that enables precise specification at different levels. It is required to carry out formal software verification. A specification language specifies the semantics and structure of a software system and is used during formal verification to provide mathematical proof that a system works according to prior specifications. Even though these factors make OCaml code a good fit for formal verification, deductive verification has rarely been used with OCaml programs, explains Pereira. Moreover, prior to the Cameleer project, an automated verification tool that can deal directly with code written in OCaml did not exist.

Together with Dr Arthur Charguéraud, Dr Jean-Christophe Filliâtre and Dr Cláudio Belo Loureiro (Charguéraud et al, 2019), Pereira has developed a specification language for OCaml, called GOSPEL – Generic Ocaml SPEcification Language. Since then, GOSPEL has grown into a mature project, currently being developed by a large consortium of researchers which include Clément Pascutto, Nicolas Osborne, Dr François Pottier, and Dr Arnald Guéneau. GOSPEL is designed to facilitate the verification of data structures and algorithms. It has formal semantics that are defined by translating into Separation Logic, a form of reasoning about programs. GOSPEL offers a high-level, concise syntax that is accessible to programmers who are not familiar with Separation Logic. While GOSPEL was developed to specify OCaml code, it is also intended as a generic tool for verification and testing with features that could be applied to other programming languages.

CAMELEER

The researchers have developed Cameleer, the first automated deductive software verification tool for OCaml-written code. From its use in everyday tasks, such as social networking, to highly critical missions, such as space exploration, software is the mainstay that keeps our world operating. Our everyday life depends on programs written in OCaml.

From social networking to space exploration, software is the mainstay that keeps our world operating. But how do we know that we can trust software? Dr Mário Pereira from the Nova School of Science and Technology, Lisbon, and his collaborators have developed Cameleer, a formal verification software tool for OCaml-written code that establishes mathematical proof that a system works according to prior specifications. Cameleer is the first automated deductive software verification tool for programs written in OCaml.

The world has become dependent on software. From its use in everyday tasks, such as social networks, to highly critical missions, such as space exploration, software is the mainstay that keeps our world operating. Software development is an error-prone activity, to say nothing of its susceptibility to malicious attacks. Defective software can have embarrassing and expensive consequences. For instance, in 2015, a failure occurred during a daily system refresh shut down in the point-of-sales systems of 7,000 Starbucks stores in the US and 1,000 in Canada. Employees were unable to accept card payments or register change, so they were forced to give customers free coffee and tea. Only a few hours later, Starbucks is reported to have lost between three and four million dollars that day. Another example occurred in 2008 when the new Heathrow Terminal 5 opened and with it the promise of a modern, efficient, and functioning baggage-handling system. The system was tested prior to opening with over 12,000 pieces of luggage and worked perfectly. When the terminal opened to the public, the system was unable to cope with the vast amount of luggage checked in every day, and only ten days after its launch, approximately 42,000 bags were lost and more than 500 flights cancelled.

So how do we know that we can trust software? One answer is software reliability testing. Such tests, however, only cover a subset of the possible executions that the software will run in practice. The best solution involves including robust mathematical guarantees in the software development process. ‘Formal methods’ is a field of computer science that advocates expressing the correctness of code into a set of mathematical statements. Deductive software verification involves formally establishing that the program execution obeys the logical properties described in its specification. Pereira explains that, despite the advances in deductive verification and proof automation in past decades, the family of functional languages used to construct programs, by applying and composing functions, has received little attention.

Our everyday life depends on software. It’s therefore important that we test its reliability.
Behind the Research: Dr Mario Pereira

Dr Pereira and his collaborators developed the Cameleer tool, a formal verification software for OCaml-written code.

Research Objectives

Dr Pereira and his collaborators developed the Cameleer tool, a formal verification software for OCaml-written code.

Detail

Address

NOVA School of Science and Technology
Computer Science Department, Office 239
28293-166 Caparica, Setubal, Portugal

Bio

Dr Mário Pereira has been Assistant Researcher at Nova School of Science and Technology, Lisbon, since 2020. In March 2022, he was appointed Tenure Track Assistant Professor. He is affiliated with NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) at NOVA University of Lisbon. Dr Pereira completed his PhD in 2018, under the supervision of Jean-Christophe Filliâtre at Université Paris-Saclay. He is an expert in deductive software verification and functional programming and the architect and lead developer of the Cameleer tool. He is also an active member of the Why3 and GOSPEL development teams.

Funding

- H2020-EU.1.3.2. Nurturing excellence by means of cross-border and cross-sector mobility
- MICA-IF-2019 Individual Fellowships (grant agreement ID 897873)
- VOCAL-ANR-15-CE25-008

Collaborators

On the development of Cameleer:
- António Ravara (https://ctf.di.fct.unl.pt/~aravara)
- Simão Melo de Sousa (www.dii.ubr.pt/~desousa)
- Tiago Soares (student)
- Daniel Castanho (student)
- Carlos Pinto (student)

On the development of GOSPEL:
- Jean-Christophe Filliâtre (www.lri.fr/~filliatr/index.en.html)
- François Pottier (pauillac.inria.fr/~fpottier)
- Arthur Charguéraud (www.chargueraud.org)
- Nicolas Osiborne (www.linkedin.com/in/wn-osiborne)
- Naela Guéneau (github.com/gueeneau)
- Cláudio Belfro Lourenço (https://belfrou.github.io)

References


Personal Response

Why do you think it has taken until now for someone to develop an automated verification tool that can deal directly with code written in OCaml?

First, let me state the existence of CFML, a wonderful verification tool that can deal with OCaml code. However, CFML requires a high degree of human proof interaction, which can only be expected from formal methods experts and not regular programmers.

Getting back to the question, the literature includes a lot of automated verification tools, but targeting imperative languages like C/C++, Java, and so on. I honestly believe this is due to the fact that industry (and in particular, critical industry) was much more into the use of imperative (sometimes low-level) languages as they depicted functional languages as ‘too academic’ or ‘not up to the job’ (in terms of efficiency, flexibility, maintainability, etc). Now, the truth is that functional languages have grown from small research projects into fully fledged mainstream languages, hence are gaining momentum within industry. This is therefore the right time to invest in formal verification of functional programs, as I believe more and more programmers will be willing to adopt functional languages and formal methods in their daily routines.