Science of Computer Programming 96 (2014) 279–296

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Verification of the functional behavior of a floating-point program: An industrial case study

Claude Marché [a,b,*]

[a] Inria Saclay – Île-de-France, Palaiseau F-91120, France
[b] LRI, Univ. Paris-Sud, CNRS, Orsay F-91405, France

HIGHLIGHTS

- We analyze a critical C code involving floating-point computations.
- The code is dealing with rotations represented by quaternions.
- A functional requirement is given, about the norms of quaternions involved.
- We express the requirement in a formal specification language.
- Code is verified with respect to the specification using theorem proving.

ARTICLE INFO

Article history:
Received 18 March 2013
Received in revised form 12 March 2014
Accepted 7 April 2014
Available online 18 April 2014

Keywords:
Deductive program verification
Automated theorem proving
Floating-point computations
Quaternions

ABSTRACT

We report a case study that was conducted as part of an industrial research project on static analysis of critical C code. The example program considered in this paper is an excerpt of an industrial code, only slightly modified for confidentiality reasons, involving floating-point computations. The objective was to establish a property on the functional behavior of this code, taking into account rounding errors made during computations. The property is formalized using ACSL, the behavioral specification language available inside the Frama-C environment, and it is verified by automated theorem proving.

1. Introduction

The objective of the U3CAT project[1] was to design various kinds of static analyses of C source code, to implement them inside the Frama-C environment [1], and to experiment them on critical industrial C programs. A part of this project was focused on the verification of programs involving floating-point computations. Several case studies of this particular kind were proposed by industrial partners of the project, and were analyzed using techniques based on abstract interpretation and on deductive verification.

This paper reports one such case study. A functional property of its behavior is formalized using ACSL—the behavioral specification language of Frama-C—and proved using a combination of automated theorem provers. These are either fully automatic ones: SMT (*Satisfiability Modulo Theories*) solvers Alt-Ergo [2,3], CVC3 [4] and Z3 [5], the solver Gappa [6] for real arithmetic; or the interactive proof assistant Coq [7].

* Correspondence to: Bât. 650, Université Paris-Sud, 91405 Orsay cedex, France.

[1] This work was partly funded by the U3CAT project (ANR-08-SEGI-021, http://frama-c.com/u3cat/) of the French national research organization (ANR), and the Hisseo project, funded by RTRA Digiteo (DIGITEO-2008-Hisseo, http://hisseo.saclay.inria.fr/).

http://dx.doi.org/10.1016/j.scico.2014.04.003
0167-6423/

We first present in Section 2 the case study itself and the functional property that should be validated. We discuss there why we believe this case study is interesting to publish. In Section 3 we describe the basics of the verification environment in which we verified the program: Frama-C, the ACSL specification language [8] including its specific features about floating-point computations, and the Jessie/Why plug-in [9–11] for deductive verification in Frama-C. We emphasize an important point of the methodology we followed: in the first step, one should specify the program, and prove it, using an idealized model of execution, where no rounding errors occur, that is where computations are assumed to be made in infinite precision. This is the mode we use to perform a preliminary analysis of the case study in Section 4. Only in the second step one should adapt the specifications, and the proof, to take into account rounding errors in floating-point computations: this is done for our case study in Section 5.

2. Presentation of the case study

The case study was provided by the company *Sagem Défense et Sécurité* (http://www.sagem-ds.com/), which is part of the larger group *Safran*. It is specialized in high-technology, and holds leadership positions in optronics, avionics, electronics and critical software for both civil and military markets. *Sagem* is the first company in Europe and third worldwide for inertial navigation systems used in air, land and naval applications.

The case study is an excerpt of a code related to inertial navigation, that deals with rotations in the three-dimensional space. A standard representation of such rotations makes use of the mathematical notion of *quaternions* [12]. To perform the verification of that case study, there is indeed no need to understand why or how this representation works. We summarized below only the basic notions about quaternions that are needed for our purpose.

2.1. Quaternions in a nutshell

Basically, the set of quaternions $\mathbb{H}$ can be identified with the four-dimensional vector space $\mathbb{R}^4$ over the real numbers. As a vector space, $\mathbb{H}$ is naturally equipped with the operations of addition and multiplication by a scalar. A common notation is made by choosing some basis denoted as $(1, i, j, k)$, so that every quaternion q is uniquely written as a linear combination $q_1 + q_2 i + q_3 j + q_4 k$. Using this basis, the *multiplication* of two quaternions can be defined thanks to the identities

$$\begin{aligned} &i^2 = j^2 = k^2 = -1 \\ &ij = k \qquad jk = i \qquad ki = j \\ &ji = -k \qquad kj = -i \qquad ik = -j \end{aligned}$$

leading to the formula

$$\begin{aligned} (q_1 + q_2 i + q_3 j + q_4 k) \times (p_1 + p_2 i + p_3 j + p_4 k) = {} & q_1 p_1 - q_2 p_2 - q_3 p_3 - q_4 p_4 \\ & + (q_1 p_2 + q_2 p_1 + q_3 p_4 - q_4 p_3) i \\ & + (q_1 p_3 - q_2 p_4 + q_3 p_1 + q_4 p_2) j \\ & + (q_1 p_4 + q_2 p_3 - q_3 p_2 + q_4 p_1) k \end{aligned}$$

It is worth to remind that multiplication is *not* commutative.

The *norm* of a quaternion is also defined, as

$$\|q\| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}$$

Among other properties, an important property is that the norm of a product is equal to the product of the norms. Quaternions of norm 1 are of particular interest for representing rotations.

2.2. The source code

The source code that was given to analyze mainly amounts to repeatedly multiplying a quaternion by other quaternions that come from some external sources of measure. The simplified source code is given in Fig. 1, where the external source of quaternion is abstracted by the C function `random_unit_quat` returning an arbitrary quaternion. In C, a quaternion is represented by an array of four double-precision floating-point numbers (type `double`). We remind that the *precision* of type `double` is 53 binary digits, meaning that the relative precision of the representation of real numbers is approximately 10^{-16}.

The arbitrary quaternions returned by function `random_unit_quat` are intended to be of norm 1, so the repeated multiplication should in principle remain of norm 1 over time. However, due to the imprecision of the floating-point representation, this property is not valid. First, the norm of those arbitrary quaternions cannot be exactly 1, only close to 1 up to a small amount. Second, due to additional imprecisions of the computation of multiplication, the norm of the iterated