Intra-procedural Optimization of the Numerical Accuracy of Programs

Nasrine Damouche¹,², Matthieu Martel¹,², and Alexandre Chapoutot³

¹ University of Perpignan Via Domitia, DALI Team-Project, Perpignan, France
² University of Montpellier II and CNRS, LIRMM, UMR, 5506, Montpellier, France
³ ENSTA ParisTech, Palaiseau, France
nasrine.damouche@univ-perp.fr

Abstract. Numerical programs performing floating-point computations are very sensitive to the way formulas are written. These last years, several techniques have been proposed concerning the transformation of arithmetic expressions in order to improve their accuracy and, in this article, we go one step further by automatically transforming larger pieces of code containing assignments and control structures. We define a set of transformation rules allowing the generation, under certain conditions and in polynomial time, of larger expressions by performing limited formal computations, possibly among several iterations of a loop. These larger expressions are better suited to improve the numerical accuracy of the target variable. We use abstract interpretation-based static analysis techniques to over-approximate the roundoff errors in programs and during the transformation of expressions. A prototype has been implemented and experimental results are presented concerning classical numerical algorithm analysis and algorithm for embedded systems.

Keywords: Program transformation · Floating-point numbers · Static analysis · IEEE754 standard

1 Introduction

These last years, as the complexity of the floating-point computations [1,23] carried out in embedded systems and elsewhere increased, numerical accuracy has become a more and more sensitive subject in computer science. Due to the important impact of accuracy on the reliability of embedded systems, many industries and companies encourage research to validate [5,10,14,13] and improve [16,21] their software in order to avoid failures and eventually disasters in aeronautics, automotives, robotics, etc.

In this article, we focus on the transformation [6,8] of intra-procedural pieces of code in order to automatically improve their accuracy. For automatic transformation of single arithmetic expressions, several techniques have already been proposed. We can mention [16] which introduces a new intermediary representation (IR) that manipulates in a single data structure a large set of equivalent arithmetic expressions. This IR, called APEG [16,17] for Abstract Program Expression Graphs, succeeds to reduce the complexity of the transformation in
polynomial size and time. Starting from this state of the art, we aim at going a step further by automatically transforming larger pieces of code. Our interest is to transform automatically sequences of commands that contain assignments and control structures in order to improve their numerical accuracy. This transformation consists in optimizing a target variable with respect to some given ranges for the input variables of the program. Accuracy bounds are computed by abstract interpretation [7] techniques for the floating-point arithmetic [13].

We start by motivating our work with a case study concerning an algorithm frequently used in robotics for odometry. We show how to rewrite it into another program which is more accurate numerically but equivalent semantically (in the sense that both programs compute the same function in exact arithmetic). This transformation operates by simplifying and developing the expressions and inlining them into other expressions. This allows one to generate new formulas and to reduce the number of operations in programs. We also rewrite the codes by unfolding the body of loops, manner to have more computations on a single iteration. The transformation of the odometry program and the rewriting rules used to automatically rewrite codes are the main contribution of this article. These rules are presented as sequents containing conditions under which the transformation may be applied without breaking the semantical equivalence between the source and target programs. In addition, these rules are applied deterministically, yielding a polynomial time transformation. This work is completed by experimental results involving the transformation of codes coming from multiple domains of science.

This article is organized as follows. Section 2 is consecrated to our case study about odometry and Section 3 introduces related work concerning the analysis and transformation of arithmetic expressions. In Section 4, we give the set of transformation rules for commands together with the conditions required to conserve the semantical equivalence of programs. Section 5 presents experimental results and shows various experimentations obtained using our prototype. Finally, Section 6 concludes.

2 Case Study: Odometry

In this section, we are interested in an example widely used in embedded systems, taken from robotics and whose code is given in Figure 2. It concerns the computation of the position of a two wheeled robot by odometry. Given the instantaneous rotation speeds $s_l$ and $s_r$ of the left and right wheels, we aim at computing the position of the robot in a cartesian space $(x, y)$. Let $C$ be the circumference of the wheels of the robot and $L$ the length of its axle (see Figure 1). We assume that $s_l$ and $s_r$ are updated by the system, by side-effect. The computation of the position is given by

$$x(t+1) = x(t) + \Delta d(t+1) \times \cos \left( \theta(t) + \frac{\Delta \theta(t+1)}{2} \right),$$

$$y(t+1) = y(t) + \Delta d(t+1) \times \sin \left( \theta(t) + \frac{\Delta \theta(t+1)}{2} \right),$$

(1) (2)