Science of Computer Programming 133 (2017) 74–87

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Improved precision in polyhedral analysis with wrapping

Stefan Bygde [a], Björn Lisper [a,*], Niklas Holsti [b]

[a] School of Innovation, Design, and Engineering, Mälardalen University, SE-721 23, Västerås, Sweden
[b] Tidorum Ltd, Tiirasaarentie 32, FI-00200, Helsinki, Finland

ARTICLE INFO

Article history:
Received 28 January 2015
Received in revised form 30 June 2016
Accepted 25 July 2016
Available online 2 August 2016

Keywords:
Abstract interpretation
Numerical abstract domains
Convex polyhedra
Widening
Wrapping

ABSTRACT

Abstract interpretation using convex polyhedra is a common and powerful program analysis technique to discover linear relationships among variables in a program. A value analysis based on a polyhedral abstract domain can be very precise, thus reducing the number of false alarms when used to verify the absence of bugs in the code. However, the classical way of performing polyhedral analysis does not model the fact that values typically are stored as fixed-size binary strings and usually have wrap-around semantics in the case of overflows. In resource-constrained embedded systems, where 8 or 16 bit processors are used, wrapping behaviour may even be used intentionally to save instructions and execution time. Thus, to analyse such systems accurately and correctly, the wrapping has to be modelled.

Simon and King [1] devised a technique to handle wrapping in polyhedral analysis in a sound way. However, their approach can sometimes yield large overapproximations. We show how the analysis can be made significantly more precise by simple means, thus making the approach more practically useful for program verification. An experimental evaluation shows that there indeed can be significant improvements in precision.

1. Introduction

Static program analysis [2] is increasingly being used for program verification. Static analysis is usually designed to be *sound*, meaning that it will never underestimate the possible set of program behaviours. This makes it very interesting for analysing safety-critical software, since it then can eliminate the possibility that the software exhibits certain erroneous behaviours. This is in contrast to testing, which cannot prove the absence of such behaviours unless exhaustive.

A particular form of static program analysis is value analysis, which aims to bound the set of possible program states in each different program point. The considered states are most often mappings from program variables to values: the result of a value analysis then restricts the possible values of the program variables. Value analysis can be used to directly prove the absence of certain kinds of run-time errors like, for instance, division by zero, indexing outside the range of an array, or dereferencing a null pointer. It can also be used by client analyses: an example is Worst-Case Execution Time (WCET) analysis [3], where value analysis can aid the construction of a tight upper bound to the execution time by restricting the ranges of memory accesses or by helping to find safe program flow constraints [4,5].

* Corresponding author.
E-mail addresses: stefanbygde@gmail.com (S. Bygde), bjorn.lisper@mdh.se (B. Lisper), niklas.holsti@tidorum.fi (N. Holsti).

http://dx.doi.org/10.1016/j.scico.2016.07.006
0167-6423/

```
unsigned char x;

x = N;
while (x >= 1) do {
    x = x + 1;
}
```

Fig. 1. A simple example of an intentional wrap-around: the loop is executed for `x` ranging from `N` to 255, and it terminates when `x` wraps around from 255 to 0. This saves some instructions as compared with a solution that does not use wraparounds.

Value analyses are most often based on the framework of abstract interpretation, where concrete and abstract domains are linked through a Galois connection and the analysis is done by a fixed-point computation in the abstract domain over equations formed from transfer functions for the program statements. Already the seminal paper [6] on abstract interpretation presented a value analysis – the well-known interval analysis, where sets of program states are approximated in an abstract domain of mappings from program variables to intervals. This abstract domain is an example of a numerical abstract domain, approximating states with numerical variables. It is also a nonrelational domain, i.e., it cannot capture dependencies between variable values. Relational domains can capture such dependencies, and are thus as a rule more precise, but the precision usually comes at a cost of reduced scalability. Many relational and non-relational numerical domains have been developed [6–10], providing different tradeoffs between precision and scalability.

Most "classical" abstract numeric domains assume that numbers are unbounded. However, in real programs, a value is typically stored as a fixed-size binary string. This introduces the risk of *overflows*, meaning that a value is too large to be stored in a binary string of the given size. An overflow could result in a run-time error, saturation of the result at the largest or smallest representable value of the type, or, as is most common for integers: a wrap-around.

This means that value analyses based on these abstract domains are unsound for integers with wrapping behaviour. This poses a problem for the analysis of any software that is vulnerable for overflows but is particularly grave for embedded systems, which typically both have real-time constraints and limited resources. Because they may use processors with a short word-length (due to resource constraints) wrap-arounds may very well be present, by mistake or even intentionally to save some instructions: see Fig. 1 for an example of an intentional wraparound. Thus, to analyse such programs it is crucial to have abstract numerical domains that are sound also in the presence of wrap-arounds. There are now such versions of many non-relational abstract domains [11–14], but for relational domains such results are scarce. This is a problem for the analysis of embedded software for safety-critical applications, especially for small processors, where precision often may be of greater concern than scalability and thus the power of relational domains is desired.

An interesting class of abstract relational numerical domains are *polyhedral* domains [7,10]. They can capture relations between different program variables as linear inequalities. Simon and King [1] proposed a way to make value analysis based on polyhedral domains sound in the presence of wrapping. Their approach assumes that each integer variable has a type of bounded size, with a wraparound semantics that is basically modulo the size, and is based on the observation that linear transformations commute with modulo. However, linear inequalities do not: $x > y$ does not imply that $x \bmod n > y \bmod n$ in general. Thus transfer functions that are based on inequalities, typically for conditionals, must be augmented with an explicit wrapping procedure where the polyhedron is piecewise wrapped back into the "base window" given by the sizes of the variable types. This wrapping turns out to be a potential source of large imprecision, rendering the analysis less useful in practical situations.

Our contributions are the following: we extend the approach by Simon and King in order to reduce the imprecision introduced by the wrapping. Our approach is based on a combination of limited widening [15], an appropriate placement of the widening points, and imposing bounds on variables based on type information. A comparative evaluation shows that it is possible to increase the precision of the analysis significantly by these simple means. The improvement comes at the cost of a minor increase in the number of fixed-point iterations of the analysis. The evaluation is done for a version of Halbwachs' polyhedral domain [7] that is modified to handle wraparounds, but our approach is also applicable to other polyhedral domains.

Our original motivation for developing this analysis is in the use of polyhedral value analyses in the parametric program flow analysis described in [16,17]. This analysis requires a relational abstract domain, and the results are typically used in a subsequent parametric WCET analysis. Thus, to make such analysis sound for software using wrap-arounds, a sound value analysis is required. As mentioned wrap-arounds are not uncommon in code for small embedded processors, which are much used in embedded time-critical applications. Our results are however general and by no means restricted to this particular application.

The rest of this paper is organised as follows. Section 2 introduces terminology and basic concepts of abstract interpretation, and explains Simon & King's method. In Section 3 we describe our improved analysis, and we prove a theorem about the boundedness of the polyhedra appearing in the analysis. In Section 4 we give an example illustrating the differences between our approach and earlier approaches. Section 5 describes the experimental setup, and Section 6 contains a subsequent comparative evaluation. Section 8 gives an account of related work. We conclude with a summary of the results, discussion, and directions for future research in Section 9.