Static Analysis of Bounded Polyhedra

Stefan Bygde
Mälardalen University
Västerås, Sweden
stefan.bygde@mdh.se

Björn Lisper
Mälardalen University
Västerås, Sweden
bjorn.lisper@mdh.se

Niklas Holsti
Tidorum Ltd
Hesinki, Finland
niklas.holsti@tidorum.fi

Abstract

We present a method for polyhedral abstract interpretation which derives fully bounded polyhedra for every step in the analysis. Contrary to classical polyhedral analysis, this method is sound for integer-valued variables stored as fixed-size binary strings; wrap-arounds are correctly modelled. Our work is based on earlier work by Axel Simon and Andy King but aims to significantly reduce the precision loss introduced in their method.

1 Introduction

A commonly used application of program analysis is to derive which numerical values the program variables can take at each point in the program. This is typically done using abstract interpretation [2] and some numerical abstract domain to get an approximation of the possible values. Abstract interpretation derives a superset of the possible values that each variable can take at each point in the program. Since exact information about the semantics of a program is uncomputable, abstract interpretation introduces a sound approximation of the possible values. The nature of the abstraction is implemented via *abstract domains* which determine which properties of the values should be abstracted. An abstract domain consists of an abstract representation of a set of program states (called abstract environments) and defines transfer functions for each construct in the programming language. The transfer functions describe how a construct changes the abstract environments. Abstract interpretation is then done by iterating and propagating the abstract environments through the program via the transfer functions until no changes occur (fixed-point iteration).

Most numerical domains suggested in the literature are based on the unsound assumption that variables can take arbitrary integer values. In real applications, a value is usually stored as a fixed-sized binary string. Analyses based on an abstract domain that assumes unbounded integers fail to model effects such as wrap-arounds. One of the most common abstract domains is the domain of *convex polyhedra* [3], which is a powerful domain that captures linear relationships between the values of the variables. The set of possible environments [1] at a program point is modelled as the set of solutions to a system of linear inequalities in $\mathbb{R}^n$ space. Each dimension corresponds to one integer-valued variable. Each (integer) solution corresponds to a possible environment at one point in a program. The set of solutions forms a convex polyhedron in $\mathbb{R}^n$ space, hence the name. Since a linear inequality may contain several variables, this domain is able to capture linear relations between variables, which is important to achieve precision.

The polyhedral domain is also based on the assumption that variables can take arbitrary integer values, so it is unsound for programs containing wrap-arounds. Simon and King [5] suggest a modification to the domain which makes it sound for wrapping integers, but this modification introduces a lot of imprecision.

2 Contributions

Our work aims to provide a sound and precise polyhedral analysis for integers with possible wrap-arounds. Our method builds on earlier work by Simon and King [5], but we attempt to minimise the

[1] In this context, an environment is an assignment of an integer value to each integer-valued variable.

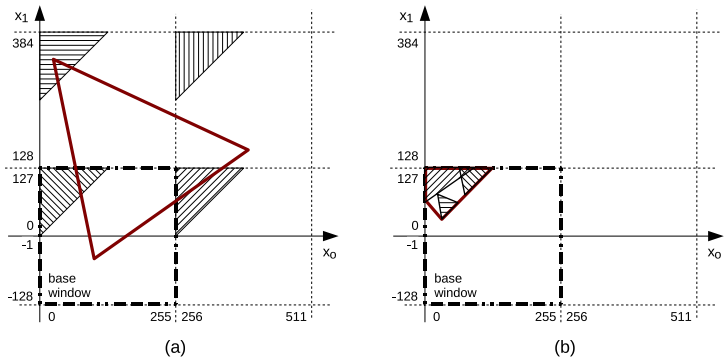

Figure 1: The picture on the left (a), shows a polyhedron before wrapping. The base window is shown outlined by a dot-dashed square. The grid of variously hatched triangles shows the condition $x_0 \leq x_1$ taken as a signed comparison of the 8-bit unsigned residue of x_0 with the 8-bit signed residue of x_1. The polyhedron intersects three components of this condition. To the right (b), the intersections of the condition with the unwrapped polyhedron are shown, shifted to the base window, and their convex hull, which is the resulting wrapped polyhedron.

introduced imprecision of their method. In classical polyhedral analysis, and indeed in Simon and King's approach, it is possible to derive polyhedra which are unbounded in some dimension. Unbounded polyhedra unfortunately introduce a lot of imprecision in Simon and King's approach. Thus, we devise a method which makes sure that every analysis step derives a fully bounded polyhedron. This makes our method potentially more precise than [5] and allows analysis with the polyhedral domain without compromising soundness or causing too much imprecision. The details of our method are available in [1].

3 Wrapping Polyhedra

The method in [5] interprets the solutions to the system of linear inequalities (i.e., the points inside a polyhedron) modulo 2^N, where N is the number of bits in a variable, with proper adjustment for signed variables. Performing abstract interpretation while using this interpretation works well since linear assignments commute with modular residue. However, linear inequalities do not commute, and therefore an explicit wrapping of the polyhedron is required whenever a linear comparison is made.

The first step of wrapping is to partition $\mathbb{R}^N$ into "windows" as in Figure 1. The linear inequality is repeated in each window to capture the comparison between an integer which might have been wrapped (i.e., lies outside the "base window"). The wrapping is then done by intersecting the partitioned polyhedron parts with the repeated inequalities, shifting the results to the base window, and finally computing the smallest polyhedron that covers all the shifted parts of the polyhedron (see the right part of Figure 1). Unbounded polyhedra are problematic for this wrapping strategy, as they would require intersecting an infinite number of polyhedra. Thus, for any unbounded variable, any relational information is discarded and the variable is simply bounded by the base window size in its dimension. Unfortunately, unbounded polyhedra commonly occur during polyhedral analysis of typical programs.

4 Bounded Polyhedra

Our approach is to use the wrapping strategy to make sound analyses, while preventing unnecessary precision loss due to unbounded polyhedra. Most of the precision loss comes from unbounded polyhedra,

so our approach is to make the polyhedra in the analysis fully bounded. There are three phases of classic polyhedral analysis which can make a polyhedron unbounded: at the initial analysis state, at any non-linear assignment of a variable and at widenings. Widening is an acceleration technique required for most abstract domains in order to terminate the analysis quickly [2].

In classical polyhedral analysis, nothing is assumed to be known about the initial state of the program variables. This is symbolised by an infinite unbounded polyhedron (or equivalently, the system of linear inequalities is empty). To make this state bounded we impose type-bounds on the initial state. Type-bounds are simply the min and max values for any signed or unsigned integer-variable represented by N bits. For example, the type-bounds for an unsigned 8-bit variable x would be: $0 \leq x \leq 255$. Thus, the initial state can safely be bounded.

The polyhedral domain, by nature, has problems with non-linearity. A non-linear assignment of a variable causes the analysis to drop all assumptions about it (i.e., losing all relational information about it). This results in a polyhedron which is unbounded in the dimension the assigned variable represents. However, since relations to other variables in this case are unknown, it is still safe to impose type-bounds for this variable, as we did in the initial state, thus making the polyhedron bounded.

Finally, we have the widening. Widening for polyhedra consists of removing those inequalities, at the widening point, that do not hold in two consecutive iterations of the analysis. Removing inequalities often results in an unbounded polyhedron. Since the polyhedron may still contain relational information, it is not safe to impose type bounds after widening.

Our approach is to do widening in conjunction with wrapping to make sure that the polyhedron remains within all type-bounds. To avoid unnecessary wrapping that could cause imprecision we apply widening only at the points where wrapping must be done - at conditional branches that depend on linear comparisons. This makes it safe to impose the type-bounds on the polyhedron after the widening (this is equivalent to making a limited widening [4] over the type-bounds), thus making the polyhedron fully bounded after widening. This, together with imposing type-bounds for variables with no relational information, results in fully bounded polyhedra in every step of the analysis.

5 Conclusion

We have developed a polyhedral analysis which is safe for wrapping integers while not losing too much precision. The core of the method is to make sure that the analysis derives fully bounded polyhedra in order to avoid unnecessary precision loss when wrapping. An implementation of the method is ongoing and a thorough comparison to Simon and King's original method is planned. Due to the more precise widening, our analysis may take a little longer to stabilise, but potentially with more precision.

References

[1] Stefan Bygde, Björn Lisper, and Niklas Holsti. Fully bounded polyhedral analysis of integers with wrapping. In *International Workshop on Numerical and Symbolic Abstract Domains (NSAD)*, September 2011.

[2] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

[3] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, pages 84–96, 1978.

[4] Nicolas Halbwachs. Delay analysis in synchronous programs. In Costas Courcoubetis, editor, *CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 333–346. Springer, 1993.

[5] Axel Simon and Andy King. Taming the wrapping of integer arithmetic. In *Static Analysis*, volume 4634 of *Lecture Notes in Computer Science*, pages 121–136. Springer Berlin / Heidelberg, 2007.