Relational Analysis of Floating-Point Arithmetic

Axel Simon

Computing Laboratory, University of Kent Canterbury, CT2 7NF, UK
A.Simon@kent.ac.uk

Abstract. A set of abstract operations is given that allows the uniform analysis of integer and floating point arithmetic in the abstract domain of convex polyhedra. We show how to implement rounding faithfully and show that the modelling of rounding is necessary even for the analysis of integer variables. Different rounding modes occurring in floating point implementations are discussed and their implementation is given.

1 Introduction

Inferring possible valuations for program variables is a classic application of abstract interpretation. Convex polyhedra [2] or subclasses [3, 4] are commonly used as an abstract domain due to their ability to express relational dependencies. Most concrete arithmetic operations can be approximated naturally as a transformation on convex polyhedra. One notable exceptions is the division operation (and division-like operations such as right shifts) that truncate their result. Truncation is an implicit rounding at the concrete level that has to be reflected explicitly in the abstract domain to ensure a sound analysis. This paper addresses several standard rounding modes and details their implementation in terms of polyhedral operations.

The ability to express rounding within the domain of polyhedra opens up the possibility to analyse floating point arithmetic side-by-side with integral values using general polyhedra. Up to now, value range analyses that are to capture floating point valuations (e.g. [1, 5]) are forced to use polyhedral sub-domains where a restricted set of coefficients makes it possible to round appropriately.

2 Modelling Division

This section demonstrates how division can be expressed abstractly by transforming polyhedra. In the sequel, let $Lin_n := \{\langle a_1, \ldots a_n, c\rangle \mid a_i, c \in \mathbb{N}\}$ represent the set of linear inequalities $\sum_{i=1}^{n} a_i x_i \leq c$. Let $Poly_n \subset Lin_n$ denote all finite subsets of Lin_n. For any given $P \in Poly_n$, define $[\![P]\!] := \{\langle x_1, \ldots, x_n\rangle \mid \sum_{i=1}^{n} a_i x_i \leq c$ where $\langle a_1, \ldots a_n, c\rangle \in P\}$, that is, $[\![P]\!]$ denotes the feasible space of the polyhedron. Note that $P \in Poly_n$ can be interpreted in $Poly_{n+1}$ by adjoining a constant zero to the vectors in P. Each (integral) program variable v is represented by some abstract variable $\sigma(v) = x_i$ in the polyhedron $P \in Poly_n$,

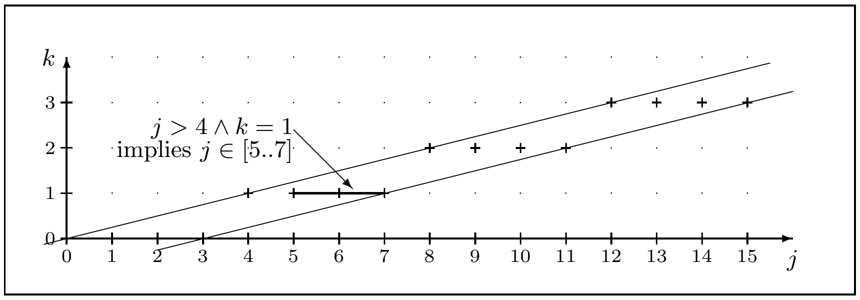

Fig. 1. The smallest convex state space that represents $k = j/4$ where k and j are integers. The crosses mark the feasible variable valuations.

$1 \leq i \leq n$. To execute the update $v := v + u + 7$, the polyhedron P is interpreted in Poly_{n+1} and two vectors representing $\sigma'(v) \leq \sigma(v) + \sigma(u) + 7$ and $\sigma'(v) \geq \sigma(v) + \sigma(u) + 7$ are added to P, where σ' concurs with σ except for $\sigma(v) = x_{n+1}$. In short, we write $P \rhd v = v + u + 7$.

The assignment of a linear expression such as v=4*u+7 can be directly expressed in the polyhedral domain. A division operations such as `k = j/4` evaluated with the classic two's-complement semantics rounds the result of the calculation towards $-\infty$. A naïve simulation in the polyhedral domain as $P' = P \rhd 4k = j$ is therefore unsound: Assume that the division assignment is followed by a conditional that tests for $j > 4 \wedge k = 1$. The abstract polyhedron P' guarantees that this condition can never hold true while, in fact, it can for $j \in [5..7]$ as shown in Figure 1. Here, the upper line represents the line $4k = j$.

3 Modelling Rounding

To capture all possible integral values that arise during a division, the polyhedron has to include all possible pairs of integral points that are possible outcomes of the division. The space between the two parallel lines in Figure 1 contains all possible values of j and k after the division. In general, a division $k = j/d$ where $d \in \mathbb{N}$ can be modelled using different rounding modes:

- Round down: $P'_{-\infty} = P \rhd dk \leq j, dk \geq j - (d-1)$
- Round up: $P'_{+\infty} = dk \leq j + (d-1), dk \geq j$
- Round to nearest: $P'_N = P \rhd dk \leq j + \left\lfloor \frac{d-1}{2} \right\rfloor, dk \geq j - \left\lceil \frac{d-1}{2} \right\rceil$
- Round towards zero: $P'_0 = P \rhd k \leq j + (d-1), dk \geq j - (d-1)$

Here the notation $P \rhd dk \leq \mathit{expr}_1, dk \geq \mathit{expr}_2$ indicates that the variable k is to be updated by adding the two given inequalities. With the exception of "round towards zero", all updates yield the smallest polyhedron for any input values of j. The last update can be refined to rounding down if the input value is known to be positive and to rounding up if the value is negative. These abstract operations on integer division serve as basis for abstract floating point calculations as shown in the next section.

4 Modelling Floating-Point Arithmetic

A floating point number consists of a mantissa (including the sign) m and an exponent e. Intuitively it can be thought of as a fraction $m/2^{-e}$. The mantissa is forced into a fixed number of bits which implies that the result of a calculation has to be approximated with the number of bits available. Consider two floating point values $m_1 = 0b1110, e_1 = 2$ and $m_2 = 0b1101, e_2 = 0$, assuming that the mantissa is restricted to four bits. Adding these two numbers yields $0b111000 + 0b1101 = 0b1000101$ which has to be rounded to four bits, i.e. $m_{res} = 0b1001, e_{res} = 3$ assuming rounding to nearest is in effect.

An abstract representation of a floating point variable consists of one variable x_m in the polyhedral domain and an independent value $x_e \in \mathbb{N}$ representing the exponent e. Consider the addition of the abstract floating-point values x_{m1}, x_{e1} and x_{m2}, x_{e2}. Let $x_{e1} \leq x_{e2}$, i.e. let the first value contain a smaller value. Firstly, the mantissas are aligned by scaling the first value: $P_1 = P \rhd x_{m1} = (x_{e2} - x_{e1})x_{m1}$. The resulting mantissa x_{mres} can then be calculated as $P_2 = P_1 \rhd x_{mres} = x_{m1} + x_{m2}$. Suppose that the maximum value of the x_{mres} is d bits larger that can be represented in a floating-point mantissa. A division of the mantissa by d with the appropriate rounding ensures that the resulting floating-point value is a sound over-approximation of the concrete, rounded value.

Conclusion and Future Work

We sketched abstract operations to model division within general polyhedral domains. Many questions remain to be discussed regarding the handling of negative values, whether exponents can and should be kept in the polyhedral domain and how to deal best with special cases like denormalised numbers and NaNs (not a number, e.g. infinities).

Practical experiments are needed to determine how the abstract division operation affects the growth of the coefficients in the polyhedral domain: While division in the concrete setting decreases the magnitude of the resulting values, the coefficients of linear inequalities are natural numbers and a fraction of one variable is expressed by increasing the coefficient in front of another. Observe that only integral points in the feasible space are of interest to the analysis sketched here. To ensure that coefficients do not grow excessively, tightening of inequalities around the integral grid can be applied. In general, the cost of tightening might outweigh the benefit of restricting coefficient growth. One solution might be to tighten only inequalities with large coefficients.

References

1. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A Static Analyzer for Large Safety-Critical Software. In *Program-*

ming Language Design and Implementation, San Diego, California, USA, June 2003. ACM.

2. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Constraints among Variables of a Program. In *Principles of Programming Languages*, pages 84–97, Tucson, Arizona, USA, January 1978. ACM.
3. A. Miné. The Octagon Abstract Domain. In *Conference on Reverse Engineering*, pages 310–319, Stuttgart, Germany, October 2001. IEEE Computer Society.
4. A. Simon, A. King, and J. M. Howe. Two Variables per Linear Inequality as an Abstract Domain. In M. Leuschel, editor, *Logic-Based Program Synthesis and Transformation*, volume 2664 of *LNCS*, pages 71–89, Madrid, Spain, September 2003. Springer.
5. A. Venet and G. Brat. Precise and Efficient Static Array Bound Checking for Large Embedded C Programs. In *Programming Language Design and Implementation*, pages 231–242, Washington DC, USA, June 2004. ACM.