Devliverable D2: A Prototype Implementation of the Bounded Polyhedra Method

Stefan Bygde

December 21, 2011

1 Introduction and Motivation

In order to validate the methods developed in deliverable D1, we have made an implementation of our bounded polyhedral method [1](from now on referred to as BD) as well as the method it is built upon by Simon and King [5](from now on referred to as SK). The motivation behind the implementation is to be able to compare BD and SK and quantify the advantages of our proposed method in a convincing manner.

2 The Static Analyser SWEET

SWEET (SWEdish Execution time Tool) is a prototype static analysis tool developed at MDH [6]. The tool is mainly used for WCET analysis, but can perform a number of various analyses such as program slicing, abstract interpretation with various domains, reaching definitions etc. The tool analyses code in an intermediate language called ALF [3]. ALF was designed specifically to be used for program analysis and to be able be represent both source and object code.

SWEET has recently incorporated abstract interpretation with the polyhedral domain. The polyhedral domain uses the Parma Polyhedra Library (PPL), an open-source library for creating and manipulating convex polyhedra [4]. This domain makes it possible to perform classic polyhedral abstract interpretation very much like the original method suggested in [2]. The maturity of the tool and its support for various analyses and the flexibility of the Parma Polyhedra Library makes it very suitable to implement and evaluate our methods by implementing them into SWEET.

2.1 Implementation in SWEET

We have added two new domains to SWEET: the SK domain and the BD domain. The SK domain runs Simon and King's method as described in their paper and the BD domain runs our bounded polyhedral method. In addition, it

is possible to run the two methods simultaneously to make comparison easier. The widening placement, which is an integral part of the bounded polyhedral method (see D1), can be set independently of which method is used. So in order to use BD properly, one must be sure to select the correct widening placement. If the analyses are run simultaneously, both have to use the same widening placement.

2.2 Implementation Details

The implementation of the SK domain is fairly straightforward as the only difference is in the handling of conditionals. As mentioned, PPL has a built-in wrapping operation which is implemented exactly as in Simon and King's method. Since SWEET is object oriented and its abstract domains are implemented as classes, this domain is implemented as a subclass of the classical polyhedral domain, overriding the behaviour of applying constraints (that is, handling conditionals).

The implementation of the BD domain inherits from the SK domain and it additionally overrides the behaviour of creating polyhedra, projecting variables and widening. The implementation features things like computing the range constraint for variables. The widening in BD uses limited widening (see D1), which fortunately also is available as an operation in PPL.

2.3 Challenges and Obstacles

The theoretical model used in D1, on which SK and BD are built, is not the same model as SWEET uses. This has led to a few practical issues.

2.3.1 Signedness Information

Both SK and BD assume that all variables have a fixed size and a fixed signedness. However, SWEET operates on the intermediate representation ALF. This format stores program variables in memory frames. Frames can be accessed in arbitrary ways and there is no fixed interpretation of the data in the frames (i.e., it is not stored as signed or unsigned). Our temporary solution to this problem is to manually provide information of the interpretation of the variables in the program to be analysed. In the future we plan to do a pre-analysis to find out how the different variables are used (i.e., if they are used as signed or unsigned in the operations).

2.3.2 Separate Widening and Conditional Handling

As mentioned, a fundamental of BD is the placement of widenings, in particular whether the widening should be done at conditional branches. SWEET operates on CFGs rather than flow charts (as is used in D1). Conditionals and branches are resolved at the end of a basic block and widening is performed either in the beginning of a basic block or at the end of a basic block, just before the branching

is resolved. However, widening and computation of conditionals are separate processes which do not by default "know" about each other. We solve this by letting each bounded polyhedron "remember" the last linear constraint that was applied to it. This means that when the widening occurs in the beginning of a basic block, the last remembered constraint is available in the memory of the widened polyhedron.

2.3.3 Placement Widening Points

BD does not dictate an exact placement of widenings, but requires that the widenings should:

1. Be in conjunction with a conditional branch
2. Occur at least once per cycle in the CFG

SWEET does label an arc that goes from a basic block to a basic block which is earlier in the CFG as a *back edge*. Thus, every cycle in a program has a back edge in SWEET. A basic block does in most cases have multiple exits guarded by a conditional. Our strategy is to give SWEET the option to place widening points at the beginning of each basic block that has an outgoing back-edge. The rationale behind this placement is that we have observed that the back-edge itself is often an unconditional branch, , while we are interested in making the widening in conjunction with a conditional branch. Thus, placing the widening point in the beginning of the basic block containing the unconditional branch is more likely to be conditional.

2.4 Evaluation and Future Work

The implementation can output a variety of statistics of the analysis results (both final and intermediate results) to facilitate a comparison. If the analyses are run simultaneously, the interrelationship between the derived polyhedra can be measured (in inclusion relationship, number of constraints, number of bounded variables etc.). Currently, we are working on measuring (i.e., counting) the number of integer points inside polyhedra, as we believe that this will be the most accurate measurement for comparing the methods.

3 References

References

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

[2] Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM Symposium on Principles of Programming Languages*, pages 84–97, 1978.

[3] Jan Gustafsson, Andreas Ermedahl, Björn Lisper, Christer Sandberg, and Linus Källberg. ALF – a language for WCET flow analysis. In Niklas Holsti, editor, *Proc. 9th International Workshop on Worst-Case Execution Time Analysis (WCET'2009)*, pages 1–11, Dublin, Ireland, June 2009. OCG.

[4] The parma polyhedra library.
URL: `http://bugseng.com/products/ppl`, December 2011.

[5] Axel Simon and Andy King. Taming the wrapping of integer arithmetic. In Hanne Riis Nielson and Gilberto Filé, editors, *Proc. 14th International Static Analysis Symposium*, volume 4634 of *Lecture Notes in Comput. Sci.*, pages 121–136, Kongens Lyngby, Denmark, 2007. Springer.

[6] SWEET execution time tool.
URL: `http://www.mrtc.mdh.se/projects/wcet/sweet.html`, December 2011.