Static analysis by abstract interpretation of the functional correctness of vector and matrix manipulating programs

Internship proposal, Master 2 MPRI, year 2014–2015

Supervisor:	Antoine Miné (`mine@di.ens.fr`)
Internship location:	Département d'informatique, École normale supérieure, Paris, France
Relevant course:	M2–6: Abstract interpretation: application to verification and static analysis

Note: Other internships are possible on the topic of numeric program analysis, static analysis, and abstract interpretation in general. Please contact the internship supervisor for more information.

Motivation

Static analysis by abstract interpretation is a popular method to verify non-functional correctness properties (i.e., nothing very wrong happens: there is no arithmetic overflow, pointer error, etc.), but it can also be used to prove *functional properties* (i.e., the program actually computes the function it is supposed to compute). For instance, recent works [2, 1] have shown that it is possible to design symbolic abstract domains able to reason about arrays of unbounded or parametric size and verify the correctness of array subroutines such as array initialization, array search, and sorting. The goal of the internship is to design new abstract domains able to prove the correctness of classic linear algebra functions, such as matrix addition, multiplication, GEMM (general matrix multiplication), or more generally BLAS (Basic Linear Algebra Subprograms).
A very simple example of matrix multiplication routine could be:

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++) {
    c[i][j] = 0;
    for (k=0; k<N; k++)
      c[i][j] += a[i][k] * b[k][j];
  }
```

and the goal is to prove that, at the end of the program, matrix `c` indeed contains the product of `a` and `b`. A successful static analysis would require designing a symbolic domain able not only to express that `c` is the product of `a` and `b`, but also able to express the loop invariants for each nested loop (i.e., `c` is a partial matrix product at each iteration, parameterized by the values of `i`, `j`, and `k`). It should also feature a precise set of transfer functions, including the modeling of assignments such as `c[i][j] = 0` and `c[i][j] += a[i][k] * b[k][j]`, as well as a join and a widening operator. We would, as a first approximation, consider that programs manipulate real numbers and ignore rounding-errors induced by the floating-point representation actually used.

From there, several directions of research are possible:

1. Analyze more complex matrix algorithms, such as Gauss elimination, matrix inversion, linear system solving, etc.
2. Analyze algorithms form realistic BLAS libraries, such as the C implementations provided by the BLIS library [3] or other implementations. Realistic implementations employ hand-crafted optimization techniques (such as loop unrolling, software pipelining, specialization, etc.) which may require more complex loop invariants (and so, a more complex abstract domain) to analyze precisely.
3. Analyze programs output from automated source-to-source optimizers, such as Pluto [4], that perform complex loop transformations (such as tiling). The generated code is much larger and more complex than the original one (including in particular extra levels of loops), and it is far from obvious to prove that it still performs the intended computation. Interestingly, such optimizers use internally their own flavor of static analysis, including polyhedral representations. The theoretical and technical connections between the field of loop optimization and program verification by abstract interpretation could be studied in this internship.
4. Due to floating-point errors, the actual result of a correct algorithm may be significantly different from the mathematical result. A possible extension to this work would be to bound the difference between the mathematical and the real results, possibly using an array version of the abstract domain proposed in [5].

Expected work

The intern shall select a set of matrix programs and their implementations, and design symbolic abstract domains (including all the necessary abstract operators) capable of proving that the implementation performs the expected mathematical computation. The proposed domains and operators shall be proved correct. An implementation and an experimental evaluation of the proposed analysis (e.g., on a toy programming language manipulating arrays) is a strong plus.

References

[1] P. Cousot and R. Cousot and F. Logozzo. A Parametric Segmentation Functor for Fully Automatic and Scalable Array Content Analysis. In *Conference Record of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 105–118, Austin, Texas, January 14-16, 2011. ACM Press, New York, NY.

[2] P. Cousot. Verification by Abstract Interpretation. *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, N. Dershowitz (Ed.), Taormina, Italy, June 29 – July 4, 2003. Lecture Notes in Computer Science, vol. 2772, pp. 243–268. © Springer-Verlag, Berlin, Germany, 2003.

[3] Field G. Van Zee and Robert A. van de Geijn. BLIS: A Framework for Rapidly Instantiating BLAS Functionality. In *ACM Transactions on Mathematical Software*, 2013.

[4] U. Bondhugula, A. Hartono, J. Ramanujan, and P. Sadayappan. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer. In *ACM SIGPLAN Programming Languages Design and Implementation (PLDI)*, Jun. 2008, Tucson, Arizona.

[5] É. Goubault and S. Putot. Static Analysis of Numerical Algorithms. In *Proceedings of Static Analysis Symposium (SAS)*, Aug. 2006.