Polyhedral Abstract Interpretation

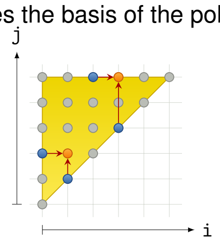
Laure Gonnord – Maître de conférences Université Lyon1/LIP

Keywords Compilation, Polyhedral Model, Abstract Interpretation.

Context

The polyhedral model is a collection of techniques developed around a common intermediate representation of programs : integer polyhedra. Such a mathematical representation of programs inherits nice structural properties. For instance, when loop transformations are represented as affine functions, compositions of transformations are also affine functions due to their closure properties.

The polyhedral representation was linked to loop programs by an analysis proposed by Feautrier [1]. This analysis provides exact dependence analysis information where statement instances (i.e., statements executed at different loop iterations) and array elements are distinguished. The exact dependence information obtained through this analysis and the use of linear programming techniques to explore the space of legal schedules [2] is what constitutes the basis of the polyhedral model for loop transformations.



```
for (i = 0; i < N; i++)
   for (j = i; j < N; j++)
S:    A[j] = f(A[i], A[j]);
```

$$\text{Domain}(S) = \{i,j \mid 0 \le i \le j < N\}$$
$$\text{Read}(S \mapsto \texttt{A}) = (i,j \to i); (i,j \to j)$$
$$\text{Write}(S \mapsto \texttt{A}) = (i,j \to i)$$
$$\text{Dep}(S \mapsto S) = \begin{cases} (i,i \to i,j) & : i < j \\ (i-1,j \to i,j) & : 0 < i \\ \textit{corner case} & : \cdots \end{cases}$$

FIGURE 1 – An example of polyhedral representation. Loop nests that fit the polyhedral model can be viewed as mathematical (constraint-based) objects, which can also be visualized geometrically.

Figure 1 illustrates the polyhedral representation with an example. The statement S is executed approximately $\frac{N^2}{2}$ times during the execution of this loop. The triangular region expressed as a set of constraints, called the *domain* of S, represents this set of dynamic execution instances. Accesses to array A from each of these statement instances can be succinctly captured through affine functions of the loop iterators. The dependences are also expressed as a function between two statement instances. The key insight is that although the specific dependences at a statement instance may differ, they are all captured by a function due to the regularity in the control flow. The figure illustrates dependences for two instances, where one of the dependences $(i, i \to i, j)$ has different lengths depending on the instance.

The "traditional" use of polyhedral techniques in optimizing compilers typically focuses on loop transformations. PLuTo [3] is a now widely used push-button tool for automatically parallelizing polyhedral loop nests that tries to optimize locality in addition to parallelization. There is also significant work in data layout optimization for polyhedral programs where analyses are performed to minimize the memory requirement [4]. Polyhedral techniques for loop transformations are now adopted by many production level compilers, such as GCC, IBM XL, and LLVM.

Recently, polyhedral techniques are being applied to many different areas besides loop transformations. One natural application of automatic parallelization techniques is in verification of given parallelizations where the tools take parallelized programs as inputs, and use polyhedral analysis to guarantee the absence of parallel bugs [5, 6]. Another application of scheduling techniques is in synthesis of ranking functions for proving program termination [7].

Subject

Although the polyhedral model provides strong analysis capabilities with many different applications, its main limitation is its applicability. The program must have regular control flow, and in addition, it has to be fully affine. Specifically, loop bounds, array accesses, and `if` conditions must be affine functions of the surrounding loop iterators and runtime constants.

In previous work [7], we demonstrated that abstract interpretation is one key to encode non regular flows, and used it to prove termination of general flowcharts programs. In this particular work we want to go a step further and express the whole polyhedral domain in a more general formal framework where the syntactic restrictions are replaced by over-approximation.

The candidate will work on expressing the concepts of the polyhedral model in a more general setting that should permit to use classical tools from abstractions to logic solving. The expected result is a unified “model” for reasoning about program flows like in the polyhedral model, namely :

- Loop iterators as elements of an abstract domain ;
- Dependencies between read statements, and write statements of cells of arrays as relations between elements of different abstract domains.
- An algorithm to compute and store an over-approximation of dependencies, implemented in a prototype.

The candidate should have knowledge in Abstract Interpretation and other formal methods such as SMT-solving and perhaps rewriting.

Remark : This internship proposal is the first step of a more general research project financed by the National Agency of Research (ANR) and for which we already have a PhD funding.

Références

[1] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1) :23–53, 1991.

[2] Paul Feautrier. Some efficient solutions to the affine scheduling problem, II, multi-dimensional time. *International Journal of Parallel Programming*, 21(6) :389–420, December 1992.

[3] Uday Bondhugula, Albert Hartono, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI ’08, pages 101–113, 2008.

[4] Alain Darte, Robert Schreiber, and Gilles Villard. Lattice-based memory allocation. *IEEE Transactions on Computers*, 54(10) :1242–1257, 2005.

[5] Vamshi Basupalli, Tomofumi Yuki, Sanjay Rajopadhye, Antoine Morvan, Steven Derrien, Patrice Quinton, and Dave Wonnacott. ompVerify : Polyhedral analysis for the OpenMP programmer. In *Proceedings of the 7th International Workshop on OpenMP*, IWOMP ’11, pages 37–53, June 2011.

[6] Tomofumi Yuki, Paul Feautrier, Sanjay Rajopadhye, and Vijay Saraswat. Array dataflow analysis for polyhedral X10 programs. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP ’13, pages 23–34, February 2013.

[7] Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *Proceedings of the 17th International Conference on Static Analysis*, SAS ’10, pages 117–133, 2010.