# *isl*: An Integer Set Library for the Polyhedral Model

Sven Verdoolaege

Department of Computer Science, Katholieke Universiteit Leuven, Belgium and
Team ALCHEMY, INRIA Saclay, France
Sven.Verdoolaege@{cs.kuleuven.be,inria.fr}

## 1 Introduction and Motivation

In compiler research, polytopes and related mathematical objects have been successfully used for several decades to represent and manipulate computer programs in an approach that has become known as the polyhedral model. The key insight is that the kernels of many compute-intensive applications are composed of loops with bounds that are affine combinations of symbolic constants and outer loop iterators. The iterations of a loop nest can then be represented as the integer points in a (parametric) polytope and manipulated as a whole, rather than as individual iterations. A similar reasoning holds for the elements of an array and for mappings between loop iterations and array elements.

For most types of program transformations, it is safe to approximate the set of integer points in a polytope by the polytope itself. Many researchers therefore use polyhedral libraries such as `PolyLib` [18] and `PPL` [1] that exploit the double description of polytopes in terms of both facets and vertices. In particular, some operations can be performed a lot more efficiently on one representation than on the other. However, the computation of one representation from the other may also be very costly, as in the worst case the size of the output may be exponential in that of the input. In practice, polyhedra that arise from compiler applications are typically close to hypercubes, i.e., they have few facets and many vertices.

A different approach is taken by the `Omega` library [16], which specifically handles sets of integer tuples satisfying affine constraints. There is also explicit support for parameters, existentially quantified variables and relations between pairs of integer tuples, making the library not only more accurate, but also more convenient to use. Note that polyhedral libraries have no need for existentially quantified variables since the projection of a rational polyhedron is again a rational polyhedron. The internal representation is based on the constraints of the sets (although vertices are implicitly constructed during the convex hull computation) and most operations are built on top of an extension of Fourier-Motzkin elimination [20] and a series of heuristics. The library is very fast on simple problems, but rather unpredictable on larger problems. Furthermore, it is not thread-safe and only supports machine precision. The library had also been left unmaintained for many years and had grown a reputation of being unreliable due to various unimplemented corner cases. Only recently have most, if not all, of these corner cases been resolved in the `Omega+` library [7].

We present `isl`, an LGPL, thread-safe, C library for manipulating sets and relations of integer tuples bounded by affine constraints using `GMP` [13] based arbitrary precision integer arithmetic. The interface of the library draws inspiration from that of `Omega`, but the underlying implementation is completely different, favoring the use of a collection of targeted and efficient algorithms. The internal representation is also different, with `Omega` transforming sets with existentially quantified variables to unions of intersections of polyhedra and lattices in order to be able to perform some set operations, while `isl` uses a representation in terms of integer divisions inspired by the output format of `PipLib`, a library for performing parametric integer programming [11]. The `isl` library is available from `http://freshmeat.net/projects/isl/`.

The `isl` library is mainly intended to be used in the polyhedral model for program analysis and transformation, but some of the many operations it supports can and have been used outside of this model. From inception, one of the primary long-term objectives has been to provide all set and polynomial manipulations required by the `barvinok` library, which, at that time, used a combination of `PolyLib`, `PipLib`, `Omega` and `GiNaC` [4]. We have already achieved the short-term objectives of replacing `PolyLib` in the loop generator `CLooG` [3], producing better code by eliminating constraints that are redundant over the integers but not over the rationals, and of forming the basis of an equivalence checker [22] of programs that can be represented in the polyhedral model.

## 2 Internals

The main objects of interest are sets and binary relations over tuples of integers bounded by affine constraints, which we will call polyhedral sets and maps, respectively. Each map $R$ is a finite union of basic maps $R = \bigcup_i R_i$, each mapping a tuple of $n$ integer parameters to a binary relation on tuples of integers, i.e., $R_i : \mathbb{Z}^n \to 2^{\mathbb{Z}^{d_1+d_2}} : \boldsymbol{s} \mapsto R_i(\boldsymbol{s})$, with

$$R_i(\boldsymbol{s}) = \{\, \boldsymbol{x}_1 \to \boldsymbol{x}_2 \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \boldsymbol{z} \in \mathbb{Z}^e : A_1\boldsymbol{x}_1 + A_2\boldsymbol{x}_2 + B\boldsymbol{s} + D\boldsymbol{z} + \boldsymbol{c} \geq \boldsymbol{0} \,\}$$

and $A_i \in \mathbb{Z}^{m \times d_i}$, $B \in \mathbb{Z}^{m \times n}$, $D \in \mathbb{Z}^{m \times e}$ and $\boldsymbol{c} \in \mathbb{Z}^m$. Sets are defined similarly. The difference between sets and maps lies only in their use. Maps have domains and ranges, can be composed with each other and can be applied to sets. Basic sets are essentially projections of the integer points in a polyhedron and include intersections of polyhedra and lattices as a special case. Note that in practice and for reasons of efficiency, equality constraints are represented separately. For some operations, it is convenient to have explicit representations for the existentially quantified variables. In particular, we use greatest integer parts of rational affine combinations of the parameters and the domain and range variables.

The core of the library is formed by an incremental LP solver modeled after that of `Simplify` [10]. This solver is used in practically every operation of the library. In particular, it is used in an ILP feasibility solver based on generalized basis reduction [9], which is in turn used to check the emptiness of a set, producing a sample element if not. Such sample elements are used during the

computation of the integer affine hull using the algorithm of [15], which is very useful for reducing the dimension of a set by detecting implicit equalities and for eliminating redundant existentially quantified variables. Finally, parametric integer programming [11] is built on top of these LP and ILP solvers. Parametric integer programming is used to compute the *lexicographic minimum* of a map and to compute a unique (lexicographically minimal) representation for the existentially quantified variables. The lexicographic minimum of a map $R$ is a map $R'$ that maps each domain element $\boldsymbol{x} \in \operatorname{dom} R$, to the unique lexicographically minimal element in its image, i.e., $R'(\boldsymbol{s}) = \{ \boldsymbol{x} \to \boldsymbol{y} \in R(\boldsymbol{s}) \mid \boldsymbol{y} = \operatorname{lexmin} R(\boldsymbol{s}, \boldsymbol{x}) \}$, with $R(\boldsymbol{s}, \boldsymbol{x}) = \{ \boldsymbol{y} \mid \boldsymbol{x} \to \boldsymbol{y} \in R(\boldsymbol{s}) \}$.

The above algorithms are used to implement the basic operations on sets and maps such as *intersection*, *union*, *difference*, *projection* and *emptiness check*. Other operations require additional algorithms, some of which are listed below.

- *convex hull*, a very "rational" operation, which therefore does not fit in very well in an integer set library and is not implemented very efficiently. Still, it is provided as it is used in `CLooG`. The algorithm is based on [14], extended to handle unbounded polyhedra. The library also provides a "*simple hull*" operation, which computes the smallest basic set that contains the input set and that can be described using only translates of the constraints of the input set. The result is an overapproximation of the convex hull, but it is much more efficient to compute.
- *set coalescing* changes the representation of a set (without changing its meaning) by replacing pairs of basic sets by a single basic set. The algorithm is based on a variation of the constraints based technique of [6], but extended to handle sets of integers. It is different from the algorithm of [2], which uses both constraints and vertices and considers only rational sets.
- the *transitive closure* of a map $R$ is the map $R^+ = \bigcup_{k \geq 1} R^k$, with $R^1 = R$ and $R^k = R \circ R^{k-1}$ for $k \geq 2$. It is computed approximatively using an algorithm that improves upon both [17] and [5].
- *dependence analysis* [12] is a crucial operation for the polyhedral model. Given a list of write and read accesses in a program, dependence analysis determines which write instance is the last to precede a given read instance. The algorithm relies heavily on the computation of lexicographic maxima.
- *parametric vertex enumeration* computes the parametric vertices of a parametric polytope and is essential for the computation in `barvinok` of the number of elements in a polyhedral set. The algorithm for the actual vertex enumeration is essentially that of [19], but the corresponding chamber decomposition is implemented much more efficiently. Preliminary experiments on a couple of non-trivial cases show that the implementation is orders of magnitude faster than that of `PolyLib` and as fast as or slightly faster than TOPCOM [21] (version 0.16.2).
- *bounds on piecewise step-polynomials* are computed in an approximative, but usually fairly accurate, way using the algorithm of [8]. Step-polynomials are polynomial expressions in greatest integer parts of affine expressions and appear as the result of (weighted) counting problems over polyhedral sets.

## References


1. Bagnara, R., Ricci, E., Zaffanella, E., Hill, P.M.: Possibly not closed convex polyhedra and the Parma Polyhedra Library. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS '09. LNCS, vol. 2477, pp. 213–229. Springer-Verlag, Berlin (2002)
2. Bagnara, R., Hill, P., Zaffanella, E.: Exact join detection for convex polyhedra and other numerical abstractions. Comput. Geom. Theory Appl. 43(5), 453–473 (2010)
3. Bastoul, C.: Code generation in the polyhedral model is easier than you think. In: PACT '04. pp. 7–16. IEEE Computer Society (2004)
4. Bauer, C., Frink, A., Kreckel, R.: Introduction to the GiNaC framework for symbolic computation within the C++ programming language. J. Symb. Comput. 33(1), 1–12 (2002)
5. Beletska, A., Barthou, D., Bielecki, W., Cohen, A.: Computing the transitive closure of a union of affine integer tuple relations. In: COCOA '09. pp. 98–109. Springer-Verlag, Berlin, Heidelberg (2009)
6. Bemporad, A., Fukuda, K., Torrisi, F.D.: Convexity recognition of the union of polyhedra. Comput. Geom. 18(3), 141–154 (2001)
7. Chen, C.: Omega+ library (2009), `http://www.cs.utah.edu/~chunchen/omega/`
8. Clauss, P., Fernandez, F.J., Gabervetsky, D., Verdoolaege, S.: Symbolic polynomial maximization over convex sets and its application to memory requirement estimation. IEEE Transactions on VLSI Systems 17(8), 983–996 (Aug 2009)
9. Cook, W., Rutherford, T., Scarf, H.E., Shallcross, D.F.: An implementation of the generalized basis reduction algorithm for integer programming. Cowles Foundation Discussion Papers 990, Cowles Foundation, Yale University (Aug 1991)
10. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM 52(3), 365–473 (2005)
11. Feautrier, P.: Parametric integer programming. Operationnelle/Operations Research 22(3), 243–268 (1988)
12. Feautrier, P.: Dataflow analysis of array and scalar references. International Journal of Parallel Programming 20(1), 23–53 (1991)
13. Free Software Foundation, Inc.: GMP, available from `ftp://ftp.gnu.org/gnu/gmp`
14. Fukuda, K., Liebling, T.M., Lütolf, C.: Extended convex hull. In: Proceedings of the 12th Canadian Conference on Computational Geometry. pp. 57–63 (2000)
15. Karr, M.: Affine relationships among variables of a program. Acta Informatica 6, 133–151 (1976)
16. Kelly, W., Maslov, V., Pugh, W., Rosser, E., Shpeisman, T., Wonnacott, D.: The Omega library. Tech. rep., University of Maryland (Nov 1996)
17. Kelly, W., Pugh, W., Rosser, E., Shpeisman, T.: Transitive closure of infinite graphs and its applications. Int. J. Parallel Program. 24(6), 579–598 (1996)
18. Loechner, V.: PolyLib: A library for manipulating parameterized polyhedra. Tech. rep., ICPS, Université Louis Pasteur de Strasbourg, France (Mar 1999)
19. Loechner, V., Wilde, D.K.: Parameterized polyhedra and their vertices. International Journal of Parallel Programming 25(6), 525–549 (Dec 1997)
20. Pugh, W.: The Omega test: a fast and practical integer programming algorithm for dependence analysis. Communications of the ACM 8, 102–114 (Aug 1992)
21. Rambau, J.: TOPCOM: Triangulations of point configurations and oriented matroids. In: Cohen, A.M., Gao, X.S., Takayama, N. (eds.) ICMS 2002. pp. 330–340 (2002)
22. Verdoolaege, S., Janssens, G., Bruynooghe, M.: Equivalence checking of static affine programs using widening to handle recurrences. In: Computer Aided Verification 21. pp. 599–613. Springer (Jun 2009)