Apron: A Library of Numerical Abstract Domains for Static Analysis *

Bertrand Jeannet[1] and Antoine Miné[2]

[1] INRIA Rhône-Alpes, Grenoble, France, Bertrand.Jeannet@inrialpes.fr
[2] CNRS, École Normale Supérieure, Paris, France, mine@di.ens.fr

Abstract. This article describes Apron, a freely available library dedicated to the static analysis of the numerical variables of programs by abstract interpretation. Its goal is threefold: provide analysis implementers with ready-to-use numerical abstractions under a unified API, encourage the research in numerical abstract domains by providing a platform for integration and comparison, and provide teaching and demonstration tools to disseminate knowledge on abstract interpretation.

1 Introduction

Static analysis aims at discovering, at compile-time, properties on all the possible executions of a program, which is useful to, *e.g.*, automatically prove the absence of errors or optimize code. *Numerical* static analysis focuses on properties of numerical program variables. *Abstract Interpretation* [9] is a general theory of semantic approximation which allows constructing static analyses that are *sound* by construction (*i.e.*, always over-approximate the set of program behaviors). In the past 30 years, many *numerical abstract domains* [6, 20, 21, 7, 10, 12, 13, 17] have been proposed, each defining a representation for a set of properties of interest and algorithms to manipulate them. They vary in expressiveness and in the cost/precision trade-off.

However, many abstract domains have no publicly available implementation while others (*e.g.*, early implementations of [10, 20]) have diverging APIs which makes reuse and experimental comparisons difficult. Most are tied to application domains, lacking operators to be useful in more general settings (*e.g.*, [5] does not support non-linear or floating-point expressions).

The Apron library aims at solving these problems and provides a uniform, rich, and user-friendly API, which is given a precise concrete semantics that underlying abstract domains are free to implement with safe approximations. It also provides reference implementations for a growing number of numerical abstract domains — including intervals, octagons [20], and polyhedra [10]. Static analyzers for all kinds of semantics (including non-linear and floating-point arithmetic) can easily use Apron to perform their numerical abstractions. Apron also encourages the development of new domains by providing an abstraction toolbox

* This work is supported by the INRIA project-team Abstraction common to the CNRS and the École Normale Supérieure, and the ANR project ASOPT.

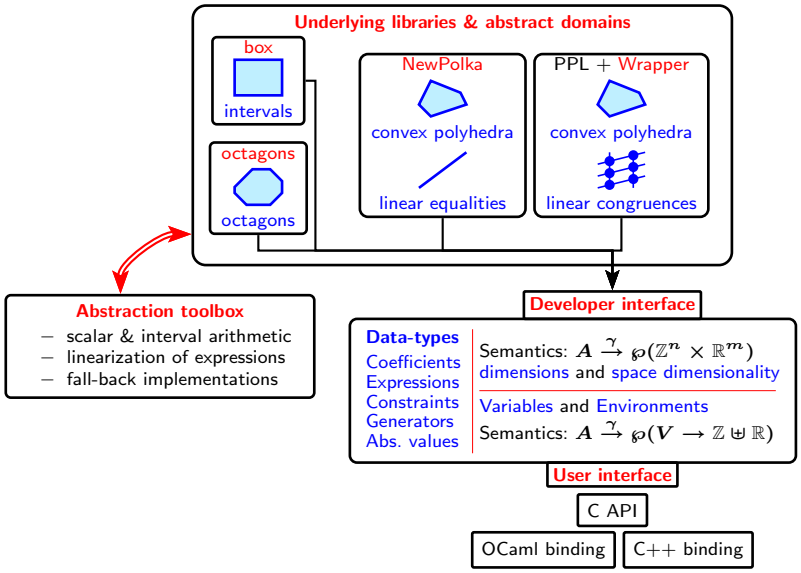

Fig. 1. Architecture of the APRON library.

library. New domains can then be plugged into APRON-powered static analyzers by changing a single line of code, as the API is essentially domain-independent.

The most closely related project is the Parma Polyhedra Library [5], which shares several features with APRON in its very last version (0.10). However, its concrete semantics does not support non-linear and floating-point expressions, and it supports only one global environment for mapping variable names to dimensions in vector spaces. Nevertheless, the domains implemented in the PPL can be interfaced to APRON and can hence benefit from its additional functionalities.

2 Architecture

The architecture of the library is sketched in Fig. 1. A client analysis would first create a context manager object that specifies which abstract domain to use, and then interface to the bottom right part of Fig. 1 by constructing domain-independent operator argument objects and issuing calls to the library. Internally, the library will dispatch the call to the proper domain implementation (top part), which can in turn use built-in generic abstractions and utility libraries (bottom left part). Due to lack of space, it is not possible to present code samples in this paper, but short examples can be found on the web-site [1].

Data-types. To communicate with domains, APRON provides various domain-independent concrete data-types to represent scalars, intervals, linear expressions, expression trees, tests, and generators, as well as associated utility functions — constructors, destructors, conversions, etc. Intervals can appear in lin-

ear and non-linear expressions, where they model a non-deterministic choice. By soundness, all possible outcomes of non-deterministic assignments are taken into account, and a program state passes a non-deterministic test if it does for at least one of its possible values. Non-determinism is useful to model program input but also to abstract complex operators into simpler ones. Operators in expressions include the four standard operations ($+$, $-$, $\times$, $/$), modulo, and square root. Each can be tagged with a rounding target type (*e.g.*, integer, or some IEEE floating-point format [8]) and direction (towards 0, $+\infty$, $-\infty$, to nearest, or non-deterministic). Constants can be provided as integers, rationals, or floats, using either machine types or arbitrary precision types — using the GMP [2] and MPFR [3] libraries.

Abstract elements and operators. Abstract elements are completely opaque data-types that ultimately represent sets of points in $\mathbb{Z}^n \times \mathbb{R}^m$. Floating-point variables can be modeled using real-valued dimensions and expression trees with explicit rounding tags. Standard operators include linear and non-linear assignments, substitutions, and constraint additions, joins, meets, widenings, projections, and various comparisons. Less common operators include n-ary joins, parallel assignments (useful to analyze automata), folds and expands (useful to analyze summarized arrays [11]).

An important rule is that the implementation is free to perform *any* sound approximation. Not imposing precision guarantees allows us to require that *all domains support all operators for all data-types*. Imprecision comes from the well-known limited expressiveness of each domain (*e.g.*, polyhedra cannot encode non-linear constraints nor disjunctions) but also from less obvious implementation details — *e.g.*, internal use of inexact arithmetic, such as floats. Thus, together with a sound result, domains may also return a flag telling whether the result is exact, optimal (w.r.t. the domain expressiveness), or neither. Also, operators have a generic cost-precision parameter that has a domain-specific but documented effect — *e.g.*, polyhedra may gain precision by using integer tightening on integer variables, or lose precision by soundly rounding large constraint coefficients to more memory-efficient ones.

Abstract domains. Fig. 2 presents a few abstract domain examples, organized as a lattice showing their respective expressiveness. The boxed domains are those currently available: intervals, octagons [20] (with either integer, rational or float bounds, using machine representation, or GMP or MPFR arbitrary precision numbers), polyhedra [10] (using GMP integers and the double description method), and linear equalities [17] (implemented as an abstraction of polyhedra). Apron can also act as a middle-ware and make foreign libraries available under its unified API. Linear congruences [13] and an alternate implementation of polyhedra are optionally available through the PPL [5]. Finally, Apron supports a generic reduced product which is instantiated to combine polyhedra and linear congruences. Fig. 2 also shows some of the domains we wish to add in the future to provide either improved expressiveness (*e.g.*, polynomial domains) or additional cost/precision trade-offs (*e.g.*, weakly relational domains).

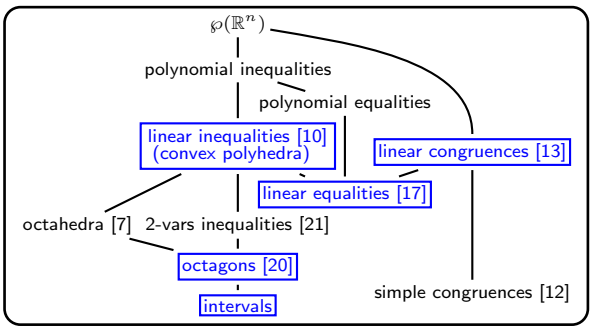

Fig. 2. Some numerical abstract domains, partially ordered by their expressiveness.

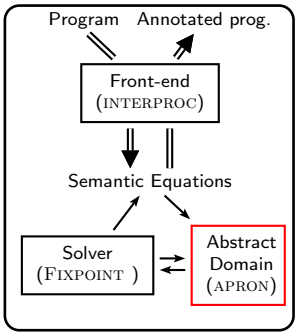

Fig. 3. Architecture of the INTERPROC analyzer.

Abstraction toolbox. APRON exposes a rich, high-level API to domain users that can chose among a large set of abstract operators and various data-type arguments. However, domain designers wish to implement a minimal set of low-level entry-points. To ease the burden of developing new domains, APRON provides several support libraries:

1. APRON provides an *environment* management package, so that developers only have to manipulate dimensions in vector-spaces while the user can manipulate named variables and environments — hence the distinct developer and user interfaces in the right part of Fig. 1.
2. APRON provides a complete arithmetic package for scalar and outward-rounded intervals in a variety of types (integers, rationals, floats; machine types, GMP, and MPFR numbers). Sound conversion functions are provided so that developers can choose the most convenient arithmetic type to use internally and still handle all the API data-types available to the user (*e.g.*, polyhedra are implemented using integers but can handle floating-point constants in expressions).
3. APRON provides *linearization* functions that domains can use to abstract arbitrary expression trees into linear ones (with or without interval coefficients). In particular, they can handle integer and floating-point expressions by abstracting away the non-linearity introduced by rounding, and output linear expressions with $\mathbb{R}$ semantics [19]. Linearization is not compulsory and developers may also access the raw expression trees provided by the user.
4. APRON provides a default implementation for many operators. For instance, compound operators provided for the user convenience (such as n-ary joins or parallel assignments) are implemented in terms of simpler ones (regular joins and assignments). Domain designers are always free to implement those operators if a specialized implementation would provide some efficiency gain.

Bindings. The core API is in C (27 KLOC) to ensure maximum interoperability, but it is designed in an object-oriented way. Additionally, a C++ binding (leveraging the object-orientation aspect and adding intelligent constructors, destructors, and operator overloading for improved user-friendliness) and an OCAML

binding (providing stricter type-checking and fully automated memory management) are provided. Other API bindings may be easily designed. As for the C API, these bindings are fully domain-independent and immediately benefit from the addition of new abstract domains.

Finally, the library is fully thread-safe and several instances (with possibly different domains) can be used concurrently through the use of context managers.

3 Applications

Distribution. The APRON library is freely available [1] and is released under the LGPL license.[1] Currently, we are aware of ten other research teams using APRON. Two of them plan to contribute to its development.

The INTERPROC *static analyzer.* INTERPROC [15] fulfills three goals: provide a library showcase, disseminate abstract interpretation techniques, and help prototype new domains [6] and analyses [18, 4] — *e.g.*, by encoding problems in the input language. Its architecture is depicted in Fig. 3. INTERPROC acts both as a front-end generating semantic equations from the input program, and as a driver interpreting the equations in the chosen abstract domain and solving them with the FIXPOINT [14] equation solver.

INTERPROC takes as input programs written in a small imperative language featuring numerical variables, loops, and recursive procedures. It performs either forward analysis that infers invariants at each program point, or backward analysis that infers necessary conditions to reach a given set of control points, or a combination of both analyses. Procedures are handled in a relational way using [16]. The output of INTERPROC is a program annotated with invariants that can also be interpreted as procedure summaries at the exit point of procedures. Any of the abstract domains provided by APRON can be selected, as well as several options regarding iteration strategies in fixpoint solving, thus allowing for easy experimentation and comparison. INTERPROC is written in OCAML (2 KLOC, excluding the FIXPOINT solver [14]) and released under the LGPL license.

4 Conclusion

The APRON library provides several numerical abstract domains under a unified interface. It defines a precise concrete semantics for a quite complete set of operations — including non-linear and floating-point arithmetic — that domains are free to approximate in a sound way, and that are suitable to build many kind of static analyses. APRON encourages the development and experimental comparison of new abstract domains by providing a toolbox enabling developers to focus on the core algorithms of their domain while benefiting for free from powerful features such as floating-point semantics. Thus, it conciliates two kinds of users with conflicting requirements: analysis designers wishing for a rich, high-level,

[1] However, *optional* dependencies [5] may be released under a more strict license.

and domain-independent API, and domain designers, wishing to implement a minimal set of low-level entry-points. Its effectiveness is illustrated by the number of research teams using it and developing new domains. A sample analyzer, INTERPROC, demonstrates its use.

In the future, we plan to integrate more numerical abstract domains, but also to develop abstract domains combining numerical and finite-type (such as boolean) variables, and extend INTERPROC accordingly.

References

1. The APRON abstract domain library. `http://apron.cri.ensmp.fr/library/`.
2. GMP: The GNU multiple precision arithmetic library. `http://gmplib.org/`.
3. The MPFR library. `http://www.mpfr.org/`.
4. H. Amjad and R. Bornat. Towards automatic stability analysis for rely-guarantee proofs. In *VMCAI'09*, volume 5403 of *LNCS*, 2009.
5. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *SCP*, 72(1–2):3–21, 2008.
6. L. Chen, A. Miné, and P. Cousot. A sound floating-point polyhedra abstract domain. In *APLAS'08*, volume 5356 of *LNCS*, pages 3–18. Springer, 2008.
7. R. Clarisó and J. Cortadella. The octahedron abstract domain. In *SAS'04*, volume 3148 of *LNCS*, pages 312–327. Springer, 2004.
8. IEEE Computer Society. IEEE standard for binary floating-point arithmetic. Technical report, ANSI/IEEE Std. 745-1985, 1985.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252. ACM Press, 1977.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL'78*, pages 84–97. ACM Press, 1978.
11. D. Gopan, F. DiMaio, N. Dor, T. Reps, and S. Sagiv. Numeric domains with summarized dimensions. In *TACAS'04*, pages 512–529, 2004.
12. P. Granger. Static analysis of arithmetical congruences. In *Int. Journal of Computer Mathematics*, volume 30, pages 165–190, 1989.
13. P. Granger. Static analysis of linear congruence equalities among variables of a program. In *TAPSOFT'91*, volume 493 of *LNCS*, pages 169–192. Springer, 1991.
14. B. Jeannet. The Fixpoint solver. `http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint/`.
15. B. Jeannet and al. The Interproc analyzer. `http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`.
16. B. Jeannet and W. Serwe. Abstracting call-stacks for interprocedural verification of imperative programs. In *AMAST'04*, volume 3116 of *LNCS*, July 2004.
17. M. Karr. Affine relationships among variables of a program. *Acta Informatica*, pages 133–151, 1976.
18. R. Majumdar, A. Gupta, and A. Rybalchenko. An efficient invariant generator. In *TACAS'09*, LNCS, 2009.
19. A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In *ESOP'04*, volume 2986 of *LNCS*, pages 3–17. Springer, 2004.
20. A. Miné. The octagon abstract domain. *HOSC*, 19(1):31–100, 2006.
21. A. Simon, A. King, and J. Howe. Two variables per linear inequality as an abstract domain. In *LOPSTR'02*, volume 2664 of *LNCS*, pages 71–89. Springer, 2002.