Random: R-based Analyzer for Numerical DOMains

Gianluca Amato and Francesca Scozzari

Università "G. d'Annunzio" di Chieti–Pescara — Dipartimento di Scienze

Abstract. We present the tool `Random` (R-based Analyzer for Numerical DOMains) for static analysis of imperative programs. The tool is based on the theory of abstract interpretation and implements several abstract domains for detecting numerical properties, in particular integer loop invariants. The tool combines a statistical dynamic analysis with a static analysis on the new domain of parallelotopes. The tool has a graphical interface for tuning the parameters of the analysis and visualizing partial traces.

1 Introduction

In the abstract interpretation framework [14], the expressive power of an analyzer strictly depends on the choice of the abstract domain. In the last 20 years, many abstract interpretation frameworks have been proposed based on different semantics (see, for instance, [19, 10, 2]), equipped with many abstract domains, with different trade-offs between expressivity and efficiency. The expressivity of an abstract domain mostly depends on the kind of constraints (assertions) that the abstract domain can represent. The simplest constraint is a constant bound on the value of a program variable, such as $-20 \leq x \leq 100$. The abstract domain of intervals [13], which can handle conjunctions of these constraints, is very efficient but not very expressive, since it cannot prove relationships between variables, such as $x_1 + x_2 \leq 100$. On the contrary, the abstract domain of (convex) polyhedra [15] can represent any linear constraint between program variables, such as $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$, where $x_1, \ldots, x_n$ are program variables and $a_1, \ldots, a_n, b$ are numerical constants (which may be integer, rational or floating point). The abstract domain of polyhedra is very precise for linear constraints but its computational cost is very high.

Many other abstract domains, which reduce the expressive power of general polyhedra while improving efficiency, have been proposed. In most cases, new abstract domains are derived by considering linear constraints subject to syntactic restrictions. This is the case of the difference bound matrices domain [20], which allows only the constraints $a \leq x_1 \leq b$ and $a \leq x_1 - x_2 \leq b$, and for the octagon domain [21] which allows the constraints $a \leq x_1 + x_2 \leq b$ and $a \leq x_1 - x_2 \leq b$. A slight generalization is the two-variables per-inequality domain [26] whose constraints may only contain two variables, such as $a_1x_1 + a_2x_2 \leq b$, and the octahedron abstract domain [12], which can handle constraints whose coefficients are $0, 1, -1$, that is $\pm x_1 \pm x_2 + \ldots \pm x_n \leq b$.

A different approach has been followed in the template polyhedra abstract domain [24]. This is a parametric domain which, given a finite set $\{a_{1i}x_1 + \ldots + a_{ni}x_n\}_i$ of linear forms fixed *a priori* (the template), allows the constraints $a_{1i}x_1 + \ldots + a_{ni}x_n \leq b_i$. The template approach may be viewed as a generalization of difference bound matrices, octagon and octahedron, since it allows any kind of linear constraints. However, the computational cost of its abstract operators is higher, since any algorithm should be able to deal with any kind and any number of constraints. Moreover, it remains the key problem of how to find the template.

The recently proposed abstract domain of template parallelotopes [3, 5] tries to retain the advantages of both approaches. Like template polyhedra, it allows to represent any kind of linear constraint, but the number of constraints in the template is n – the number of variables in the program – and the constraints are required to be linearly independent. Bounding the number of constraints is the key to find very efficient algorithms for the abstract operators. In fact, parallelotopes can be thought of as intervals expressed in a non-standard basis in the vector space of variable's values.

Our tool `Random` (R-based Analyzer for Numerical DOMains) implements three different ways of using parallelotopes. First, we have implemented the template parallelotope domain, using a fixed template to analyze the whole program. In order to find the coefficients in the template, we use two statistical tools, namely the *Principal Component Analysis* (PCA) and the *Independent Component Analysis* (ICA) [17]. Second, we have implemented a template parallelotope approach where we can associate a template to each program point (or to some selected program points). Third, we provide an implementation of the parallelotope abstract domain (without templates), which exploits the full expressive power of parallelotopes. At the end, `Random` annotates each program point of the input program with the constraints discovered by the static analysis.

In the tool, we have privileged the implementation of efficient (and obviously correct) operators, disregarding optimality and completeness (see, for instance, [14, 6, 16, 25]), in particular in the parallelotope abstract domain.

1.1 Template parallelotopes

The tool can analyze a given program by using the domain of template parallelotopes. In order to find the template, we have implemented a technique based on the pre-analysis of the partial execution traces of the program. Consider the example program in Figure 1. Initially, we run an instrumented version of the program to collect the values of numerical variables in some specific program points for different inputs. Figure 2 shows the set of values collected at the program points ① and ②, where the grey rectangle is the abstraction on the interval domain. Then, we apply to the sample data a statistical technique in order to find the template. The tool implements two different techniques: the Principal Component Analysis and a new technique combining the PCA with the Independent Component Analysis.

```
x = 42
y = 0
while (x>y) {
 ①  x = x-1
 ②  y = y+2
}
```

Fig. 1. The example program.

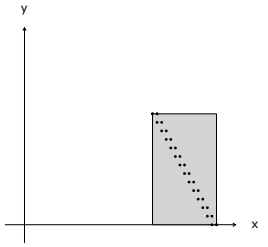

Fig. 2. Interval abstraction of a partial execution trace, observed at program points ① and ②.

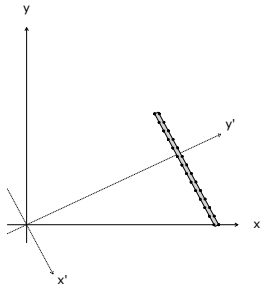

Fig. 3. Parallelotope abstraction with axes rotated according to PCA.

The principal component analysis finds a new orthonormal coordinate system maximizing the variance of the collected values. More explicitly, PCA finds new axes such that the variance of the projection of the data points on the first axis is the maximum among all possible directions, the variance of the projection of the data points on the second axis is the maximum among all possible directions which are orthogonal to the first axis, and so on. For instance, if we apply PCA to the values collected from partial executions traces of the program in Figure 1, we get the new basis (x', y') in Figure 3.

On the contrary, the independent component analysis looks for components that are both independent and non-Gaussian. Two variables are independent if knowing something about the value of one variable does not yield any information about the value of the other one. Independence is a stronger property than uncorrelatedness, and it is immediate to see that if two variables are independent, then their covariance is zero. In practice, ICA cannot find a representation whose components are really independent, but it can at least find components that are as independent as possible. We have implemented a combination of PCA and ICA, where we combine the most promising PCA components (those with very low variance) with the most promising ICA components.

The result of the statistical analysis is further refined by a simplification procedure, which stabilizes the result and avoids approximation errors. We supply two simplification procedures which return an approximation of the principal components which are proportional to vectors of small integers. The first procedure, namely the *Orthogonal Simple Component Analysis* (OSCA) [7], minimizes the angle between a principal component and its approximation, while the second procedure analyzes the ratio between the coefficients in a single component.

For the program shown in Figure 1, OSCA finds the change of basis matrix $\begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}$ whose columns correspond to the axes (x', y') in Figure 3. With this template, we can represent the constraints $a \leq x - 2y \leq b$ and $c \leq 2x + y \leq d$,

```
{
    [ x=0 , y=0 : -x+2*y=0 , 2*x+y=0 ( )  ]
    x = 42
    [ x=42 , y=0 : -x+2*y=-42 , 2*x+y=84 ( ) ]
    y = 0
    [ x=42 , y=0 : -x+2*y=-42 , 2*x+y=84 ( ) ]
    while ({
        [ 27<=x<=42 , 0<=y<=30 : -42<=-x+2*y<=33 , 2*x+y=84 ( ) ]
        x > y
    }) {
        [ 28<=x<=42 , 0<=y<=28 : -42<=-x+2*y<=28 , 2*x+y=84 ( ) ]
        x = x - 1
        [ 27<=x<=41 , 0<=y<=28 : -41<=-x+2*y<=29 , 2*x+y=82 ( ) ]
        y = y + 2
        [ 27<=x<=41 , 2<=y<=30 : -37<=-x+2*y<=33 , 2*x+y=84 ( ) ]
    }
    [ 27<=x<=28 , 28<=y<=30 : 28<=-x+2*y<=33 , 2*x+y=84 ( ) ]
}
```

Fig. 4. The result of the static analysis

thus proving that $2x + y = 84$ holds at the program point ①. Note that none of these constraints may be expressed either in the interval domain or in octagon. The result of the analysis is shown in Figure 4.

1.2 Multiple template parallelotopes

The tool is able to change the template in specific program points, marked by a call to the function `.tag(n)`. The parameter `n` is optional and can be used to glue the collected data coming from different program points, thus creating a virtual program point.

`Random` uses statistical tools to compute different templates for any (virtual) program point, and then selects each template in the corresponding program point. In order to switch from a template to the next one, the tool projects the data on the new template with standard algebraic operations. In practice, this approach amounts to partition the source code in fragments, and to apply a different template to each fragment. Thus, in each fragment we can represent up to n different constraints, which must be linearly independent, and do not need to be related to the constraints in other fragments.

Using multiple templates improves the precision of the template parallelotope domain, but can significantly reduce efficiency.

1.3 Parallelotope abstract domain

We have implemented the full domain of parallelotopes. The domain changes the template at each program point, according to the operation to be performed. The

key points in the design of this domain are the join and widening operators. Both are implemented as a variant of the inverse join [23], which allows us to discover new constraints at a reasonable computational cost. It is also crucial, at least in our implementation, to use delayed widening, so that new invariants may be discovered without being immediately discarded. For instance, consider the following program.

```
x = 1
y = 1
while (y < 100) {
   y = y + y
   y = y + y
   x = x + x
   x = x + x
}
```

The tool starts the analysis with the standard interval domain. After the first two lines, the constraints are $x = 1, y = 1$. At the end of the first while iteration we obtain the constraints $x = 4, y = 4$. Since we use delayed widening, in the first iteration we simply join the two constraints. The inverse join of $x = 1, y = 1$ and $x = 4, y = 4$ yields $-x + y = 0$, $1 \leq x \leq 4$ and $1 \leq y \leq 4$. The heuristic we have developed chooses two linearly independent constraints from the result, in this case $-x + y = 0$ and $1 \leq x \leq 4$. The constraint $-x + y = 0$ is preferred to $1 \leq y \leq 4$ since it is saturated by both constraints $x = 1, y = 1$ and $x = 4, y = 4$.

When processing the assignment $y = y+y$ the analyzer changes the template by transforming $-x+y = 0$ in the new constraint $-2x+y = 0$. After the second assignment $y = y + y$, we get $-4x + y = 0$. Now we process the assignment $x = x + x$ and get the constraints $-2x + y = 0, 2 \leq x \leq 8$, and after the last assignment we get the constraints $-x + y = 0, 4 \leq x \leq 16$. By applying the widening operator, we discard all the constraints which are changed w.r.t. the previous iteration, and we get $-x + y = 0, 1 \leq x$ which is the final result of the analysis.

2 The tool interface

Figure 5 shows the tool's interface. On the left side, there are the four main panels. The Source code panel allows to upload and edit a program, while the Analysis result shows the result of the analysis. The Matrix panel shows the templates used in the analysis, while the Partial Trace panel shows the collected values to be analyzed with the statistical engines.

On the right side we may choose the abstract domains and tune the precision. In the first section Partial Execution Traces, we instruct the tool on the program points to be considered and we may select a subset of the variables for the analysis. In the Trace Analysis section we choose the statistical engine (either PCA or ICA), or we can provide a user-defined template matrix. We can also choose whether to use a single template for the whole program or to change the template at the program points selected in the previous section.

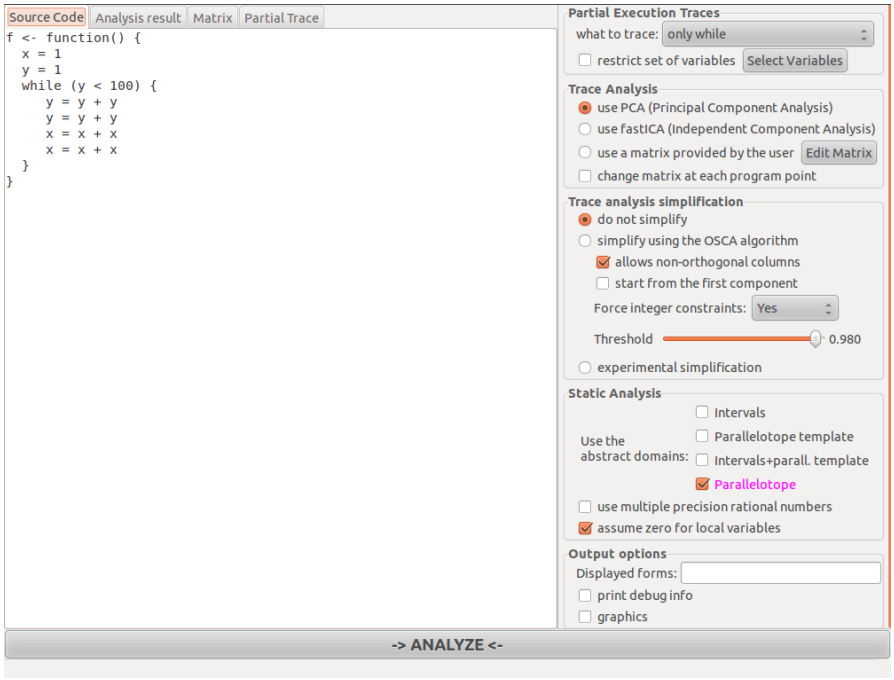

Fig. 5. A screenshot of `Random`

In the Trace Analysis Simplification section we choose the simplification strategy to be applied to the result of the statistical engine. The OSCA procedure can be fine-tuned by choosing several parameters. The most important is the Threshold, which misures the distance between the original matrix and the simplified one. In the Static Analysis section we choose the abstract domains to be used in the analysis: the intervals, the template parallelotopes, a combination of the two, or the parallelotope domain. In the last section Output options we can tune the output of the analysis. The Displayed forms textbox allows to insert a list of linear forms (such as `3*x+2*z`). The result of the analysis is projected on these linear forms. This may be useful to compare the result of analyses performed with different templates. The print debug info option shows, on the console, the intermediate computations of the static analysis, and the graphics option draws the values in the partial execution traces and the principal components.

3 Implementation details

The tool is available at `http://www.sci.unich.it/~amato/random` under the terms of the GNU GPLv3. A previous and partial version of the tool appeared in [4], without the multiple template parallelotopes, the general parallelotope abstract domain, the ICA analysis, the graphical user interface, the graphical display of partial execution traces and most of the options.

The tool is written in R, a language and environment for statistical computing [22]. It is a functional language with powerful meta-programming features and a vast library of statistical functions. However, it does not excel in efficiency and convenience of debugging facilities.

We analyze programs written in an imperative fragment of the R language, which includes assignments, conditionals and while loops. In addition, the programmer can use the built-in functions `brandom()`, which returns a random boolean value, and `assume(·)`, in order to make assumptions on program variables, such as `assume(x>0)`. The function `.tag(·)` allows to declare specific program points to be traced. The programmer can insert in the source code the calls `.tag(0)`, `.tag(1)`,`.tag(2)`, ..., even multiple times, in order to create virtual program points. In case of programs with parameters, the user should provide some input values, in order to generate the partial execution traces. The analyzer instruments the program to record the values of the variables in specific program points, computes the partial execution traces starting from the input values, performs the PCA or ICA statistical analysis, simplifies the results, and finally executes the static analysis. PCA and ICA are computed using respectively the standard built-in functions of R and the `fastICA` package.

The static analyzer uses a *recursive chaotic iteration strategy* on the *weak topological ordering* induced by the program structure (see [9]). Correctness of all the abstract operators is ensured by using either rational arithmetic through the *GNU Multiple Precision Arithmetic Library* or, when it is possible, by changing the rounding mode of the floating point arithmetic. To this aim, we have written an auxiliary R package `ieeeround` [1] to control the rounding mode of the CPU.

4 Conclusion and future work

In order to improve usefulness and to ease further developments of `Random`, many changes are necessary. First of all, the domains should be ported to `C/C++`, preferably inside well known libraries such as PPL [8] or APRON [18]. The analyzer engine and the program tracer should be ported to a faster and more robust language than R. Finally, the analyzer should support a mainstream target language. To this aim, we could exploit Frama-C [11], an extensible platform for source-code analysis of `C` programs, or the Clang static analyzer (`http://clang-analyzer.llvm.org/`).

References

1. G. Amato. `ieeeround`*: Functions to set and get the IEEE rounding mode*, 2011. R package version 0.2-0. `http://CRAN.R-project.org/package=ieeeround`.
2. G. Amato, J. Lipton, and R. McGrail. On the algebraic structure of declarative programming languages. *Theoretical Computer Science*, 410(46):4626–4671, 2009.
3. G. Amato, M. Parton, and F. Scozzari. Deriving numerical abstract domains via principal component analysis. In *SAS 2010, Proc.*, vol. 6337 of *LNCS*, pp. 134–150.
4. G. Amato, M. Parton, and F. Scozzari. A tool which mines partial execution traces to improve static analysis. In *RV 2010, Proc.*, vol. 6418 of *LNCS*, pp. 475–479.

5. G. Amato, M. Parton, and F. Scozzari. Discovering invariants via simple component analysis. *Journal of Symbolic Computation*, 2012. To appear. doi: 10.1016/j.jsc.2011.12.052.
6. G. Amato and F. Scozzari. Optimality in goal-dependent analysis of sharing. *Theory and Practice of Logic Programming*, 9(5):617–689, 2009.
7. K. Anaya-Izquierdo, F. Critchley, and K. Vines. Orthogonal simple component analysis: a new, exploratory approach. *Annals of Applied Statistics*, 5(1):486–522, 2011.
8. 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. *Science of Computer Programming*, 72(1–2):3–21, 2008.
9. F. Bourdoncle. Efficient chaotic iteration strategies with widenings. In *Formal Methods in Prog. and Their Appl., 1993 Proc.*, vol. 735 of *LNCS*, pp. 128–141.
10. M. Bruynooghe. A practical framework for the abstract interpretation of logic programs. *The Journal of Logic Programming*, 10(1/2/3 & 4):91–124, 1991.
11. G. Canet, P. Cuoq, and B. Monate. A value analysis for C programs. In *SCAM 2009, Proceedings*, pages 123–124, 2009. IEEE Computer Society Press.
12. R. Clarisó and J. Cortadella. The octahedron abstract domain. *Science of Compututer Programming*, 64:115–139, 2007.
13. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. Second Int'l Symposium on Programming*, pp. 106–130, 1976. Dunod.
14. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL 1979, Proc.*, pp. 269–282. ACM Press, 1979.
15. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL 1978, Proc.*, pp. 84–97, 1978. ACM Press.
16. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract domains condensing. *ACM Transactions on Computational Logic*, 6(1):33–60, 2005.
17. A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. John Wiley & Sons, 2001.
18. B. Jeannet and A. Miné. APRON: A library of numerical abstract domains for static analysis. In *CAV 2009, Proc.*, vol. 5643 of *LNCS*, pp. 661–667.
19. K. Marriott, H. Søndergaard, and N. D. Jones. Denotational abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, 16(3):607–648, 1994.
20. A. Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO 2001, Proc.*, vol. 2053 of *LNCS*, pp. 155–172.
21. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
22. R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. http://www.R-project.org/.
23. S. Sankaranarayanan, M. Colón, H. B. Sipma, and Z. Manna. Efficient strongly relational polyhedral analysis. In *VMCAI 2006, Proc.*, vol. 3855 of *LNCS*, pp. 111–125.
24. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI 2005, Proc.*, vol. 3385 of *LNCS*, pp. 25–41.
25. F. Scozzari. Abstract domains for sharing analysis by optimal semantics. In *SAS 2000, Proc.*, vol. 1824 of *LNCS*, pp. 397–412.
26. A. Simon, A. King, and J. M. Howe. Two variables per linear inequality as an abstract domain. In *LOPSTR 2002, Proc.*, vol. 2664 of *LNCS*, pp. 71–89. 2003.