Piecewise-Defined Ranking Functions*

Caterina Urban[1]

1 **École Normale Supérieure - CNRS - INRIA**
Paris, France
urban@di.ens.fr

Abstract

We present the design and implementation of an abstract domain for proving program termination by abstract interpretation. The domain automatically synthesizes piecewise-defined ranking functions and infers sufficient conditions for program termination. The analysis is sound, meaning that all program executions respecting these sufficient conditions are indeed terminating.

We discuss the limitations of the proposed framework, and we investigate possible future work. In particular, we explore potential extensions of the abstract domain considering piecewise-defined non-linear ranking functions such as polynomials or exponentials.

1998 ACM Subject Classification D.1.4 Sequential Programming, D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Abstract Interpretation, Ranking Function, Termination

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

The traditional method for proving program termination [6] is based on the synthesis of ranking functions, which map program states to elements of a well-ordered set. A program terminates if a ranking function that decreases during program execution is found. In [4], Patrick Cousot and Radhia Cousot introduced the idea of the computation of a ranking function by Abstract Interpretation [3], a general theory of programs semantics approximation. In a recent work [10], we built on their proposed general framework, to design and implement a suitable abstract domain for proving termination of imperative programs.

Intuitively, we can define a ranking function from the states of a program to ordinal numbers, in an incremental way: we start from the program final states, where the function has value 0; then, we add states to the domain of the function, retracing the program backwards and counting the number of performed program steps as value of the function.

However, such ranking function is obviously not computable. Hence, we resort to abstract interpretation to automatically compute an abstract ranking function, which consists of abstract invariants attached to program points. These abstract invariants are represented by elements of an abstract domain and state properties about the program variables whenever control reaches that program point. More specifically, the elements of the abstract domain are piecewise-defined affine functions of the program variables, representing an upper bound on the number of program execution steps remaining before termination.

The domain automatically synthesizes such piecewise-defined ranking functions through backward invariance analysis. The analysis does not rely on assumptions about the structure

* The research leading to these results was partially funded by the MBAT project (EU ARTEMIS Joint Undertaking under grant agreement no. 269335).

	1st iteration	2nd iteration	3rd/4th iteration
4	$f(x) = 0$	$f(x) = 0$	$f(x) = 0$
1	$f(x) = \begin{cases} 1 & x > 10 \\ \bot & x \leq 10 \end{cases}$	$f(x) = \begin{cases} 1 & x > 10 \\ 4 & 9 \leq x \leq 10 \\ \bot & x \leq 8 \end{cases}$	$f(x) = \begin{cases} 1 & x > 10 \\ 4 & 9 \leq x \leq 10 \\ -3x + 38 & 7 \leq x \leq 8 \\ \bot & x \leq 6 \end{cases}$
3	$f(x) = \begin{cases} 2 & x > 8 \\ \bot & x \leq 8 \end{cases}$	$f(x) = \begin{cases} 2 & x > 8 \\ 5 & 7 \leq x \leq 8 \\ \bot & x \leq 6 \end{cases}$	$f(x) = \begin{cases} 2 & x > 8 \\ 5 & 7 \leq x \leq 8 \\ -3x + 33 & x \leq 6 \end{cases}$
2	$f(x) = \begin{cases} 3 & x > 8 \\ \bot & 7 \leq x \leq 8 \\ \bot & x \leq 6 \end{cases}$	$f(x) = \begin{cases} 3 & x > 8 \\ 6 & 7 \leq x \leq 8 \\ \bot & x \leq 6 \end{cases}$	$f(x) = \begin{cases} 3 & x > 8 \\ 6 & 7 \leq x \leq 8 \\ \bot & x \leq 6 \end{cases}$

Figure 2 Simple Example Analysis.

of the analyzed program: for example, is not limited to simple loops, as in [8]. To handle disjunctions arising from tests and loops, the analysis automatically partitions the space of values for the program variables into intervals, inducing a piecewise-definition of the affine ranking functions. During the analysis, pieces are split by tests, modified by assignments and joined when merging control flows. Widening limits the number of pieces of a ranking function to a maximum given as a parameter of the analysis.

Moreover, the domain naturally infers sufficient conditions for program termination. The analysis is sound: all program executions respecting these sufficient conditions are indeed terminating, while an execution that does not respect these conditions might not terminate.

Example Let us consider a small sequential programming language with no procedures, no pointers and no recursion. The language statements include assignments, branches and while loops. All program variables have (mathematical) integer values. In particular, let us consider the simple program in Figure 1. Figure 2 illustrates the details of the backward invariance analysis. We map each program control point to a function $f \in \mathbb{Z} \mapsto \mathbb{N}$ of the variable x, representing an upper bound on the number of execution steps before termination.

```
int : x
while [1](x <= 10) do
   if [2](x > 6) then
      [3]x := x + 2
   fi
od[4]
```

Figure 1 Simple Example

The analysis is performed backwards starting from the total function $f(x) = 0$ at program point 4. At program point 1, the loop test $x \leq 10$ splits the domain of the function and enforces termination in 1 step. At program point 3, the assignment $x := x + 2$ modifies the domain of the function and increases its value to 2. The, the test $x > 6$ further splits the domain of the function. Finally, a second iteration starts joining the function at program point 4 after $x > 10$ with the function at program point 2 after $x \leq 10$.

At the fourth iteration, a fix-point is reached yielding the following ranking function

$f \in \mathbb{Z} \mapsto \mathbb{N}$ as loop invariant at program point 1:

$$f(x) = \begin{cases} \bot_{\mathsf{F}} & x \leq 6 \\ -3x + 38 & 7 \leq x \leq 8 \\ 4 & 9 \leq x \leq 10 \\ 1 & x \geq 11 \end{cases}$$

The analysis provides $x > 6$ as a sufficient condition for termination, revealing potential non-termination for $x \leq 6$. Indeed, for $x \leq 6$, the program is non-terminating. ◀

2 Piecewise-Defined Affine Ranking Functions Abstract Domain

In the following, due to space constraints, we do not recall the results presented in [4] and we introduce straightaway our abstract domain of piecewise-defined affine ranking functions. Most definitions will be only hinted, we refer to [10] for more details and examples.

The elements of the abstract domain belong to $\mathcal{V}^\# \triangleq \mathcal{S}^\# \mapsto \mathcal{F}^\#$, where $\mathcal{S}^\#$ is the set of abstract program states (in particular, we abstract the program states using the intervals abstract domain [2]) and $\mathcal{F}^\# \triangleq \{\bot_{\mathsf{F}}\} \cup \{f^\# \mid f^\# \in \mathbb{Z}^n \mapsto \mathbb{N}\} \cup \{\top_{\mathsf{F}}\}$ is the set of natural-valued ranking functions of the integer-valued program variables (in addition to the function $\bot_{\mathsf{F}}$ representing potential non-termination, and the function $\top_{\mathsf{F}}$ representing the lack of enough information to conclude). More specifically, an abstract function $v^\# \in \mathcal{V}^\#$ has the form:

$$v^\# \equiv \begin{cases} s_1^\# \mapsto f_1^\# \\ s_2^\# \mapsto f_2^\# \\ \dots \\ s_k^\# \mapsto f_k^\# \end{cases}$$

where the abstract states $s_1^\#, \dots, s_k^\#$ induce a partition of the space of values for the program variables, and the ranking functions $f_1^\#, \dots, f_k^\#$ are affine functions of the program variables.

The concretization function $\gamma \in (\mathcal{S}^\# \mapsto \mathcal{F}^\#) \mapsto (\mathcal{S} \hookrightarrow \mathbb{O})$ is applied piecewise and maps an abstract function to a partial function from program states to ordinals:

$$\gamma(s^\# \mapsto \bot_{\mathsf{F}}) = \dot{\emptyset}$$
$$\gamma(s^\# \mapsto f^\#) = \lambda s \in \gamma_{\mathsf{S}}(s^\#).\ f^\#(s(x_1), \dots, s(x_n))$$
$$\gamma(s^\# \mapsto \top_{\mathsf{F}}) = \dot{\emptyset}$$

where $\dot{\emptyset}$ denotes the totally undefined function, and the function $\gamma_{\mathsf{S}} \in \mathcal{S}^\# \mapsto \mathfrak{P}(\mathcal{S})$ maps an abstract state to the corresponding set of program states.

The domain operators for the abstract approximation order $\sqsubseteq$, the abstract computational order $\preccurlyeq$ and the abstract join $\sqcup$ rely on a partition unification algorithm that, given two abstract functions $v_1^\#$ and $v_2^\#$, modifies the partitions on which they are defined, into a common refined partition of the space of values for each program variable. In particular, since the partitions are determined by intervals with constant bounds, the unification simply introduces new bounds consequently splitting intervals in both partitions. Then, the binary operators can be applied piecewise: the abstract orders, first compare the abstract states on which each function is defined, and then compare the values of the ranking functions on each abstract state; the join operator $\sqcup$ reuses the convex-hull of polyhedra [5].

The widening operator $\triangledown$ prevents the number of pieces of an abstract function from growing indefinitely. First, it performs a partition unification that keeps only the interval

bounds occurring in the first abstract function. Then, it widens the functions piecewise, reusing the convex-hull and the widening of polyhedra.

In order to handle assignments, the abstract domain is equipped with an operation to substitute an arithmetic expression for a variable within an affine function. An assignment is carried on piecewise and independently on each abstract state and each ranking function. Then, the resulting covering induced by the abstract states is refined to obtain a partition.

Finally, to deal with tests, the abstract domain merely applies piecewise to each abstract state the abstract filter operator from the intervals domain.

The operators of the abstract domain are combined together, to compute an abstract ranking function for a program, through backward invariance analysis. The starting point is the constant function equals to 0 at the program final control point. The ranking function is then propagated backwards towards the program initial control point taking assignments and tests into account with join and widening around loops [1].

Thanks to the soundness of all abstract operators, we can establish the soundness of the analysis for proving program termination: the program states, for which the analysis finds a ranking function, are states from which the program indeed terminates.

Implementation We have implemented a research prototype static analyzer [9], based on our abstract domain of piecewise-defined affine ranking functions, and we have used it to analyze programs written in a small non-deterministic imperative language. The prototype is written in OCaml, and the operators from the intervals and convex polyhedra abstract domains are provided by the Apron library [7].

The analysis proceeds by structural induction on the program syntax, iterating loops until an abstract fix-point is reached. In case of nested loops, a fix-point on the inner loop is computed for each iteration of the outer loop, following [1].

3 Future Work

As might be expected, the implemented domain has a limited expressiveness that translates into an imprecision of the analysis especially in the case of nested loops (and, in general, of programs with non-linear complexity). For this reason, we would like to design other abstract domains, based on more sophisticated abstract states and on non-linear ranking functions such as polynomials or exponentials.

Piecewise-Defined Non-Linear Ranking Functions Let us consider the program in Figure 3: it is the (skeleton of) the Bubble Sort algorithm for an array of length 10, once we have removed all tests and assignments on the array. Since the program has a polynomial time complexity, we need non-linear (polynomial) ranking functions to prove its termination.

```
int : x1, x2
while 1(x1 <= 10) do
  2x2 := 10
  while 3(x2 > 1) do
    4x2 := x2 - 1
  od5
  6x1 := x1 + 1
od7
```

Figure 3 Bubble Sort

Figure 4 illustrates the iterates of the backward invariance analysis limited at program control point 1. At the third iteration, the analysis tries to synthesize an affine ranking function for the program. However, such function is not a fix-point: at the next iteration, for $x_1 \leq 8$, we obtain an affine function $f_2(x_1, x_2) = -24x_1 + 259$ with greater slope than $f_1(x_1, x_2) = -22x_1 + 243$; this manifests the need for a polynomial function and, in particular, it leads to the parabola $f(x_1, x_2) = \frac{1}{2}x_1^2 - 31x_1 + \frac{567}{2}$ tangent to both $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$ and passing through

	1
1st iteration	$f(x_1,x_2)=\begin{cases}1 & x_1>10\\ \bot & x_1\leq 10\end{cases}$
2nd iteration	$f(x_1,x_2)=\begin{cases}1 & x_1>10\\ 23 & x_1=10\\ \bot & x_1\leq 9\end{cases}$
3rd iteration	$f(x_1,x_2)=\begin{cases}1 & x_1>10\\ 23 & x_1=10\\ -22x_1+243 & x_1\leq 9\end{cases}$
4th iteration	$f(x_1,x_2)=\begin{cases}1 & x_1>10\\ 23 & x_1=10\\ 43 & x_1=9\\ -24x_1+259 & x_1\leq 8\end{cases}$
4th/5th iteration	$f(x_1,x_2)=\begin{cases}1 & x_1>10\\ 23 & x_1=10\\ \frac{1}{2}x_1^2-31x_1+\frac{567}{2} & x_1\leq 9\end{cases}$

Figure 4 Bubble Sort Analysis.

the point $x_1 = 9$ of $f_1(x_1) = -22x_1 + 243$. At the fifth iteration, a fix-point is reached, proving program termination for all values of x_1 and x_2. ◀

It also remains to investigate the possibility of structuring computations as suggested by [4]. In addition, we plan to extend our research to proving other liveness properties.

References

1 François Bourdoncle. Efficient Chaotic Iteration Strategies with Widenings. In *FMPA*, pages 128–141, 1993.

2 Patrick Cousot and Radhia Cousot. Static Determination of Dynamic Properties of Programs. In *Symposium on Programming*, pages 106–130, 1976.

3 Patrick Cousot and Radhia Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*, pages 238–252, 1977.

4 Patrick Cousot and Radhia Cousot. An Abstract Interpretation Framework for Termination. In *POPL*, pages 245–258, 2012.

5 Patrick Cousot and Nicolas Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*, pages 84–96, 1978.

6 Robert W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

7 Bertrand Jeannet and Antoine Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *CAV*, pages 661–667, 2009.

8 Andreas Podelski and Andrey Rybalchenko. A Complete Method for the Synthesis of Linear Ranking Functions. In *VMCAI*, pages 239–251, 2004.

9 Caterina Urban. FuncTion. `http://www.di.ens.fr/~urban/FuncTion.html`.

10 Caterina Urban. The Abstract Domain of Segmented Ranking Functions. In *SAS*, pages 43–62, 2013.