To Infinity... and Beyond!

Caterina Urban, Antoine Miné

To Infinity… and Beyond!*

Caterina Urban[1] and Antoine Miné[1]

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

Abstract

The traditional method for proving program termination consists in inferring a ranking function. In many cases (i.e., programs with unbounded non-determinism), a single ranking function over natural numbers is not sufficient. Hence, we propose a new abstract domain to automatically infer ranking functions over ordinals. We extend an existing domain for piecewise-defined natural-valued ranking functions to polynomials in ω, where the polynomial coefficients are natural-valued functions of the program variables. The abstract domain is parametric in the choice of the state partitioning inducing the piecewise-definition and the type of functions used as polynomial coefficients. To our knowledge this is the first abstract domain able to reason about ordinals. Handling ordinals leads to a powerful approach for proving termination of imperative programs, which in particular allows us to take a first step in the direction of proving termination under fairness constraints and proving liveness properties of (sequential and) concurrent programs.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs, F.3.2 Semantics of Programming Languages

Keywords and phrases Abstract Interpretation, Ranking Function, Ordinals, Termination

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

1 Introduction

The traditional method for proving program termination [6] consists in inferring ranking functions, namely mappings from program states to elements of a well-ordered set (e.g., ordinals) whose value decreases during program execution.

Intuitively, we can define a partial ranking function from the states of a program to ordinals in an incremental way: we start from the program final states, where the function has value 0 (and is undefined elsewhere); then, we retrace the program backwards enriching the domain of the function with the co-reachable states mapped to the maximum number of program steps until termination. In [3], this intuition is formalized into a most precise ranking function that can be expressed in fixpoint form by abstract interpretation [2] of the program maximal trace semantics.

However, the most precise ranking function is not computable. In [11], we present a decidable abstraction for imperative programs by means of piecewise-defined ranking functions over natural numbers. These functions are attached to the program control points and represent an upper bound on the number of program execution steps remaining before termination. Nonetheless, in many cases (i.e., programs with unbounded non-determinism), natural-valued ranking functions are not powerful enough. For this reason, we propose a new abstract domain to automatically infer ranking functions over ordinals.

We extend the abstract domain of piecewise-defined natural-valued ranking functions to *piecewise-defined ordinal-valued ranking functions* represented as polynomials in ω, where the polynomial coefficients are natural-valued functions of the program variables. The domain automatically infers such ordinal-valued functions through backward invariance analysis. To handle disjunctions arising

* The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no. 269335 (ARTEMIS project MBAT) (see Article II.9. of the JU Grant Agreement)

from tests and loops, the analysis automatically partitions the space of values for the program variables into abstract program states, inducing a piecewise definition of the functions. Moreover, the domain naturally infers sufficient preconditions for program termination. The analysis is sound: all program executions respecting these sufficient preconditions are indeed terminating, while an execution that does not respect these conditions might not terminate.

The abstract domain is parametric in the choices of the state abstraction used for partitioning (in particular, we can abstract the program states using any convex abstract domain such as intervals [1], octagons [9], polyhedra [4], ...) and the type of functions used as polynomial coefficients (e.g., affine, quadratic, cubic, exponential, ...).

To our knowledge this is the first abstract domain able to reason about ordinals. Handling ordinals leads to a powerful approach for proving termination of imperative programs, which in particular allows us to take a first step in the direction of proving termination under fairness constraints and proving liveness properties of (sequential and) concurrent programs.

2 Ordinal-Valued Ranking Functions

We derive a decidable program termination semantics by abstract interpretation of the program most precise ranking function [3]: first, we introduce the abstract domain O of ordinal-valued functions; then, in the next section, we employ state partitioning to lift this abstraction to piecewise-defined functions [11].

Let $\mathcal{X}$ be a finite set of program variables. We split the program state space $\Sigma \triangleq \mathcal{L} \times \mathcal{E}$ into program control points $\mathcal{L}$ and environments $\mathcal{E} \triangleq \mathcal{X} \to \mathbb{Z}$, which map each program variable to an integer value. No approximation is made on $\mathcal{L}$. On the other hand, each program control point $l \in \mathcal{L}$ is associated with an element $o \in \mathcal{O}$ of the abstract domain O. Specifically, o represents an abstraction of the partial function $\gamma_{\mathsf{O}}(o) \in \mathcal{E} \rightharpoonup \mathbb{O}$ defined on the environments related to the program control point l: $\langle \mathcal{E} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle \xleftarrow{\gamma_{\mathsf{O}}} \langle \mathcal{O}, \sqsubseteq_{\mathsf{O}} \rangle$. Intuitively, where defined, the partial function provides a ranking function proving (definite) termination; where undefined, it denotes (potential) non-termination.

Natural-Valued Functions. We assume we are given an abstraction $\langle \mathcal{S}, \sqsubseteq_{\mathsf{S}} \rangle$ of environments: $\langle \mathcal{P}(\mathcal{E}), \subseteq \rangle \xleftarrow{\gamma_{\mathsf{S}}} \langle \mathcal{S}, \sqsubseteq_{\mathsf{S}} \rangle$ (i.e., any abstract domain such as intervals [1], octagons [9], convex polyhedra [4], ...), and an abstraction $\langle \mathcal{S} \times \mathcal{F}, \sqsubseteq_{\mathsf{F}} \rangle$ of $\langle \mathcal{E} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$ by means of *natural-valued* functions of the program variables: $\langle \mathcal{E} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle \xleftarrow{\gamma_{\mathsf{F}}} \langle \mathcal{S} \times \mathcal{F}, \sqsubseteq_{\mathsf{F}} \rangle$. More specifically, the abstraction $\langle \mathcal{S} \times \mathcal{F}, \sqsubseteq_{\mathsf{F}} \rangle$ encodes a partial function $v \in \mathcal{E} \rightharpoonup \mathbb{O}$ by a pair $\langle s, f \rangle$ of a natural-valued (total) function (e.g., an affine function [11]) $f \in \mathcal{F}$ and an abstract state $s \in \mathcal{S}$ which restricts its domain. For instance, $\langle [1,5], 3x+2 \rangle$ denotes the affine function $3x+2$ restricted to the interval $[1,5]$. We can now use the abstractions $\mathcal{S}$ and $\mathcal{F}$ to build the abstract domain O.

Ordinal-Valued Functions. The elements of the abstract domain O belong to $\mathcal{O} \triangleq \mathcal{S} \times \mathcal{W}$ where $\mathcal{W} \triangleq \{\bot_{\mathsf{W}}\} \cup \{\sum_i \omega^i \cdot f_i \mid f_i \in \mathcal{F}\} \cup \{\top_{\mathsf{W}}\}$ is the set of ordinal-valued ranking functions of the program variables (in addition to the function $\bot_{\mathsf{W}}$ representing potential non-termination, and the function $\top_{\mathsf{W}}$ representing the lack of enough information to conclude[1]). More specifically, an abstract function $o \in \mathcal{O}$ is a pair of an abstract state $s \in \mathcal{S}$ and a polynomial in ω (i.e., an ordinal in Cantor Normal Form) $\omega^k \cdot f_k + \cdots + \omega^2 \cdot f_2 + \omega \cdot f_1 + f_0$ where the coefficients $f_0, f_1, f_2, \ldots, f_k$ belong to $\mathcal{F}$. Note that the ordinal $\omega^k \cdot f_k + \cdots + \omega^2 \cdot f_2 + \omega \cdot f_1 + f_0$ is isomorphic to the lexicographic tuple $(f_k, \ldots, f_1, f_0)$. In fact, our abstract domain is isomorphic to the set of all lexicographic ranking functions with finite (but unbounded) number of components. In the following, with abuse of notation, we use a map $s \mapsto p$ to denote the pair of $s \in \mathcal{S}$ and $p \in \mathcal{W}$, i.e., p restricted to s. The abstract domain O is parameterized

[1] In fact, our abstract domain is equipped with an approximation and a computational ordering (here not discussed) which respectively do and do not distinguish between $\bot_{\mathsf{W}}$ and $\top_{\mathsf{W}}$. We refer to [11] for further discussion.

by the choices of the abstraction $\langle \mathcal{S}, \sqsubseteq_{\mathsf{S}} \rangle$ and the type (e.g., affine, quadratic, cubic, exponential, ...) of natural-valued functions used as polynomial coefficients $f_0, f_1, f_2, \ldots, f_n \in \mathcal{F}$. As an example, $[1,5] \mapsto \omega \cdot (3x+2)+(2x)$ uses intervals and affine functions respectively.

As for the operators of the abstract domain, we briefly describe only the join operator and the assignment transfer function. We refer to [12] for more details and examples.

The join operator $\sqcup_{\mathsf{O}}$, given two abstract functions $o_1 \triangleq s_1 \mapsto p_1$ and $o_2 \triangleq s_2 \mapsto p_2$, determines the function $o \triangleq s \mapsto p$, defined on their common domain $s \triangleq s_1 \sqcap_{\mathsf{S}} s_2$ with value $p \triangleq p_1 \sqcup_{\mathsf{P}} p_2$. Specifically, the unification $p_1 \sqcup_{\mathsf{P}} p_2$ of two polynomials p_1 and p_2 is done in ascending powers of ω, joining the coefficients of similar terms (i.e., terms with the same power of ω). The join of two coefficients f_1 and f_2 is provided by $f \triangleq f_1 \sqcup_{\mathsf{F}} f_2$ and is defined as a *natural-valued* function (of the same type of f_1 and f_2) greater than f_1 and f_2 (on the domain s). Whenever such function does not exist, we force f to equal 0 and we carry 1 to the unification of terms with next higher degree. Intuitively, whenever natural-valued functions are not sufficient, we naturally resort to ordinals. Let us consider the join $\omega^k \cdot f$ of two terms $\omega^k \cdot f_1$ and $\omega^k \cdot f_2$. Forcing f to equal 0 and carrying 1 to the terms with next higher degree is exactly the same as considering f equal to ω: $\omega^k \cdot f = \omega^k \cdot \omega = \omega^{k+1} \cdot 1 + \omega^k \cdot 0 = \omega^{k+1}$. To avoid computing infinite increasing chains of abstract functions, to analyze loops we use a widening operator [1] $\triangledown_{\mathsf{O}}$ which is similar to the join $\sqcup_{\mathsf{O}}$ but defaults to $\top_{\mathsf{P}}$ when the abstract function has increased between iterates.

In order to handle assignments, the abstract domain is equipped with an operation to substitute an arithmetic expression for a variable within a function $f \in \mathcal{F}$. Given an abstract function $o \triangleq s \mapsto p$, an assignment is carried out independently on the abstract state s and on the polynomial p. In particular, an assignment on p is performed in ascending powers of ω, possibly carrying 1 to the term with next higher degree. The need for carrying might occur in case of non-deterministic assignments: it is necessary to take into account all possible outcomes of the assignment, possibly using ω as approximation.

3 Piecewise-Defined Ordinal-Valued Ranking Functions

In the following, we lift the abstract domain O to piecewise-defined ranking functions [11].

The elements of the abstract domain belong to $\mathcal{V} \triangleq \mathcal{P}(\mathcal{S} \times \mathcal{W})$. More specifically, an element $v \in \mathcal{V}$ of the abstract domain has now the form:

$$v \triangleq \begin{cases} s_1 \mapsto p_1 \\ \quad \vdots \\ s_k \mapsto p_k \end{cases}$$

where the abstract states $s_1, \ldots, s_k \in \mathcal{S}$ induce a partition of the space of environments $\mathcal{E}$ and $p_1, \ldots, p_k$ are ranking functions represented as polynomials $\omega^k \cdot f_k + \cdots + \omega^2 \cdot f_2 + \omega \cdot f_1 + f_0$ whose coefficients $f_0, f_1, f_2, \ldots, f_n \in \mathcal{F}$ are natural-valued functions of the program variables.

The binary operators of the abstract domain rely on a partition unification algorithm that, given two piecewise-defined ranking functions v_1 and v_2, modifies the partitions on which they are defined into a common refined partition of the space of program environments. For example, in case of partitions determined by intervals with constant bounds, the unification simply introduces new bounds consequently splitting intervals in both partitions. Then, the binary operators are applied piecewise: the piecewise join $\sqcup_{\mathsf{V}}$ computes the piecewise-defined natural-valued ranking function greater than v_1 and v_2 using $\sqcup_{\mathsf{O}}$. The piecewise widening $\triangledown_{\mathsf{V}}$ keeps only the partition of the domain of the first function. In this way, it prevents the number of pieces of an abstract function from growing indefinitely. It also prevents the indefinite growth of the *value* of an abstract function by using $\triangledown_{\mathsf{O}}$.

The unary operators for assignments and tests are also applied piecewise. In particular, assignments are carried out independently on each abstract state and each ranking function. Then, the resulting covering induced by the over-approximated abstract states is refined (joining overlapping pieces) to obtain once again a partition.

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 equal

```
int : x1, x2
while ( x1≠0 ∧ x2≥0 ) do
   if ( x1>0 ) then
      if ( ? ) then
         x1 := x1 - 1
         x2 := [−∞,∞]
      else
         x2 := x2 - 1
   else /* x1<0 */
      if ( ? ) then
         x1 := x1 + 1
      else
         x2 := x2 - 1
         x1 := [−∞,∞]
```

Figure 1 Program with no lexicographic ranking function.

```
int : x, b
x := 0, b := 1
[ while ( b > 0 ) do x := x + 1 ] ||
[ b := 0 ]
```

```
int : x, b, z1, z2
z1 := [0,+∞], z2 := [0,+∞], x := 0, b := 1
while ( b > 0 ) do
   if ( z1≤z2 ) then
      x := x + 1
      z1 := [0,+∞]
      z2 := z2 - 1
   else
      b := 0
```

Figure 2 Concurrent variant (above) and non-deterministic variant (below) of Dijkstra's random number generator [5].

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 using join and widening for loops. As a consequence of the soundness of all abstract operators (see [11]), 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 incorporated the implementation of the abstract domain O of ordinal-valued ranking functions into our prototype static analyzer [10] based on piecewise-defined ranking functions. The prototype accepts programs written in (a subset of) C. It is written in OCaml and, at the time of writing, the available abstractions for program environments $\mathcal{S}$ are based on intervals [1], octagons [9] or convex polyhedra [4], and the available abstraction for natural-valued functions $\mathcal{F}$ is based on affine functions. The operators for the intervals, octagons and convex polyhedra abstract domains are provided by the APRON library [8]. The analysis proceeds by structural induction on the program syntax, iterating loops until an abstract fixpoint is reached. In case of nested loops, a fixpoint on the inner loop is computed for each iteration of the outer loop.

▶ **Example 1.** Let us consider the program in Figure 1. The variables x_1 and x_2 can have any initial integer value, and the program behaves differently depending on whether x_1 is positive or negative. In case x_1 is positive, the program either decrements the value of x_2 or decrements the value of x_1 and resets x_2 to any value. In case x_1 is negative, the program either increments the value of x_1 or decrements the value of x_2 and resets x_1 to any value (possibly positive). The loop exits when x_1 is equal to zero or x_2 is less than zero.

Note that there does not exist a lexicographic affine ranking function for the loop. In fact, the variables x_1 and x_2 can be alternatively reset to any value at each loop iteration: the value of x_2 is reset in the first branch of the first if statement (i.e., if $x_1 > 0$) while the value of x_1 is reset in the second branch of the first if statement (i.e., if $x_1 < 0$).

Nonetheless, the program always terminates, regardless of the initial values for x_1 and x_2, and regardless of the non-deterministic choices taken during execution. Our prototype is able to prove the program terminating in about 10 seconds (with a widening delay of 3 iterations). We automatically infer the following piecewise-defined ranking function (at loop entry):

$$f(x_1,x_2) = \begin{cases} \omega^2+\omega\cdot(x_2-1)-4x_1+9x_2-2 & x_1<0 \wedge x_2>0 \\ 1 & x_1=0 \vee x_2\leq 0 \\ \omega\cdot(x_1-1)+9x_1+4x_2-7 & x_1>0 \wedge x_2>0 \end{cases}$$

Note that, from any state where $x_1 < 0$ and $x_2 = k_2 > 0$, whenever the value of x_1 is reset, it is possible to jump to any state where $x_2 = k_2 - 1$. Thus, f must go up to ω^2 (... and beyond!) as it is possible to jump through unbounded non-determinism to states with value of the most precise ranking function equal to an arbitrary ordinal between ω and ω^2.

Finally, note the expressions identified as coefficients of ω: when $x_1 < 0$, the coefficient of ω is an expression in x_2 (since x_2 guides the progress towards the final states), and when $x_1 > 0$, the coefficient of ω is an expression in x_1 (because x_1 now rules the progress towards termination). The expressions are automatically inferred by the analysis without requiring assistance from the user. ◀

4 Conclusion and Future Work

In this paper, we proposed a parameterized abstract domain for proving termination of imperative programs. The domain automatically infers sufficient conditions for program termination, and synthesizes piecewise-defined ordinal-valued ranking functions through backward invariance analysis.

The full version of this short paper has been published in [12]. Due to space constraints, we refer to [11, 12] for a comparison with related work.

It remains for future work to extend our research to proving termination under fairness constraints and thus proving liveness properties of (sequential and) concurrent programs. However, as shown in the following example, handling ordinals already allows us to take a first step in this direction.

▶ **Example 2.** Let us consider the concurrent variant of Dijkstra's random number generator [5] in Figure 2. The program is terminating *under fairness assumptions*. Nonetheless, a program transformation can be applied in order to introduce *unbounded non-determinism* and thus explicitly represent the fair scheduler within the program [7]. Once this transformation is carried out, the resulting non-deterministic program (in Figure 2) can be proved terminating by our ordinal-valued ranking functions.

However, more abstractions are to be expected to handle all practical cases.

References

1 Patrick Cousot and Radhia Cousot. Static Determination of Dynamic Properties of Programs. In *Symposium on Programming*, pages 106–130, 1976.
2 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.
3 Patrick Cousot and Radhia Cousot. An Abstract Interpretation Framework for Termination. In *POPL*, pages 245–258, 2012.
4 Patrick Cousot and Nicolas Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*, pages 84–96, 1978.
5 Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
6 Robert W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.
7 Nissim Francez. *Fairness*. Springer, 1986.
8 Bertrand Jeannet and Antoine Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *CAV*, pages 661–667, 2009.
9 Antoine Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
10 Caterina Urban. FuncTion. `http://www.di.ens.fr/~urban/FuncTion.html`.
11 Caterina Urban. The Abstract Domain of Segmented Ranking Functions. In *SAS*, pages 43–62, 2013.
12 Caterina Urban and Antoine Miné. An Abstract Domain to Infer Ordinal-Valued Ranking Functions. In *ESOP*, 2014.