To Infinity... and Beyond!
Caterina Urban, Antoine Miné

To cite this version:
Caterina Urban, Antoine Miné. To Infinity... and Beyond!. 14th International Workshop on Termination (WST’14), Jul 2014, Vienne, Austria. pp.5. hal-01105216

HAL Id: hal-01105216
https://inria.hal.science/hal-01105216
Submitted on 20 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
To Infinity... and Beyond!*

Caterina Urban and Antoine Miné

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 $\omega$, 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 $\omega$, 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 $X$ be a finite set of program variables. We split the program state space $\Sigma \triangleq X \rightarrow \mathbb{Z}$ into program control points $L$ and environments $E \triangleq X \rightarrow \mathbb{Z}$, which map each program variable to an integer value.

No approximation is made on $L$. On the other hand, each program control point $l \in L$ is associated with an element $o \in O$ of the abstract domain $O$. Specifically, $o$ represents an abstraction of the partial function $\gamma_O(o) \in L \rightarrow O$ defined on the environments related to the program control point $l$: $\langle E \rightarrow O, o \rangle \triangleq (O, \subseteq O)$. 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 $\langle S, \subseteq S \rangle$ of environments: $\langle P(E), \subseteq E \rangle \triangleq (S, \subseteq S)$ (i.e., any abstract domain such as intervals [1], octagons [9], convex polyhedra [4], . . . ), and an abstraction $\langle S \times F, \subseteq F \rangle$ of $\langle E \rightarrow O, \subseteq E \rangle$ by means of natural-valued functions of the program variables: $\langle E \rightarrow O, \subseteq E \rangle \mapsto (S \times F, \subseteq F)$. More specifically, the abstraction $\langle S \times F, \subseteq F \rangle$ encodes a partial function $o \in E \rightarrow O$ by a pair $(s, f)$ of a natural-valued (total) function (e.g., an affine function [11]) $f \in F$ and an abstract state $s \in S$ 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 $S$ and $F$ to build the abstract domain $O$.

Ordinal-Valued Functions. The elements of the abstract domain $O$ belong to $O \triangleq S \times W$ where $W \triangleq \{\bot_W\} \cup \{\sum_{i} \omega^i \cdot f_i \mid f_i \in F\} \cup \{\top_W\}$ is the set of ordinal-valued ranking functions of the program variables (in addition to the function $\bot_W$ representing potential non-termination, and the function $\top_W$ representing the lack of enough information to conclude¹). More specifically, an abstract function $o \in O$ is a pair of an abstract state $s \in S$ and a polynomial in $\omega$ (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 $F$. 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 $s \in S$ and $p \in W$, 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 $\bot_W$ and $\top_W$. We refer to [11] for further discussion.
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
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 $S$ are based on intervals [1], octagons [9] or convex polyhedra [4], and the available abstraction for natural-valued functions $F$ 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):

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

Finally, note the expressions identified as coefficients of $\omega$: when $x_1 < 0$, the coefficient of $\omega$ is an expression in $x_2$ (since $x_2$ guides the progress towards the final states), and when $x_1 > 0$, the coefficient of $\omega$ 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