An Abstract Domain to Infer Ordinal-Valued Ranking Functions[⋆]

Caterina Urban and Antoine Miné

ÉNS & 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 maximum degree of the polynomial, and the types of functions used as polynomial coefficients.
We have implemented a prototype static analyzer for a while-language by instantiating our domain using affine functions as polynomial coefficients. We successfully analyzed small but intricate examples that are out of the reach of existing methods.
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 subsumes existing techniques based on lexicographic ranking functions.

1 Introduction

The traditional method for proving program termination [12] 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 ordinal numbers in an incremental way: we start from the program final states, where the function has value 0 (and is undefined elsewhere); then, we add states to the domain of the function, retracing the program backwards and counting the maximum number of performed program steps as value of the function. In [10], this intuition is formalized into a most precise ranking function that can be expressed in fixpoint form by abstract interpretation [8] of the program maximal trace semantics.

[⋆] 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)

However, the most precise ranking function is not computable. In [22], 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 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 [7], octagons [19], polyhedra [11], ...), the maximum degree of the polynomials, and the type of functions used as polynomial coefficients of ω^k (e.g. affine, quadratic, cubic, exponential, ...). We have implemented an instance of the abstract domain based on interval partitions and affine functions. We successfully analyzed small but intricate examples out of the reach of existing methods.

To our knowledge this is the first abstract domain able to reason about ordinals. We show that handling ordinals leads to a powerful approach for proving program termination of imperative programs which, in particular, subsumes existing techniques based on lexicographic functions.

Motivating Example. In order to motivate the need for ordinal numbers, let us consider the well-known program in Figure 1. At each loop iteration, either it decrements the value of x_2 or it decrements the value of x_1 and resets the value of x_2, until one of the variables becomes less than or equal to zero. The program presents unbounded non-determinism: there is a non-deterministic choice between the branches of the if statement at program point 2, and the value of the variable x_2 is chosen non-deterministically at program point 4 in the first branch of the if statement. The program terminates whatever the initial values for x_1 and x_2 are, and whatever the non-deterministic choices taken during execution.

In the graph of Figure 2, each node represents a state of the program (the nodes with a double outline are final states) and each edge represents a loop iteration. We define a ranking function for the program following the intuition described above: we start from the final states, where we assign value 0 to the function; then, we follow the edges backwards, and for each state that we

```
int : $x_1, x_2$
while $^1$( $x_1 \geq 0 \wedge x_2 \geq 0$ ) do
    if $^2$( ? ) then
        $^3x_1 := x_1 - 1$
        $^4x_2 :=$ ?
    else
        $^5x_2 := x_2 - 1$
od$^6$
```

Fig. 1. Motivating example. The symbol ? stands for a non-deterministic choice.

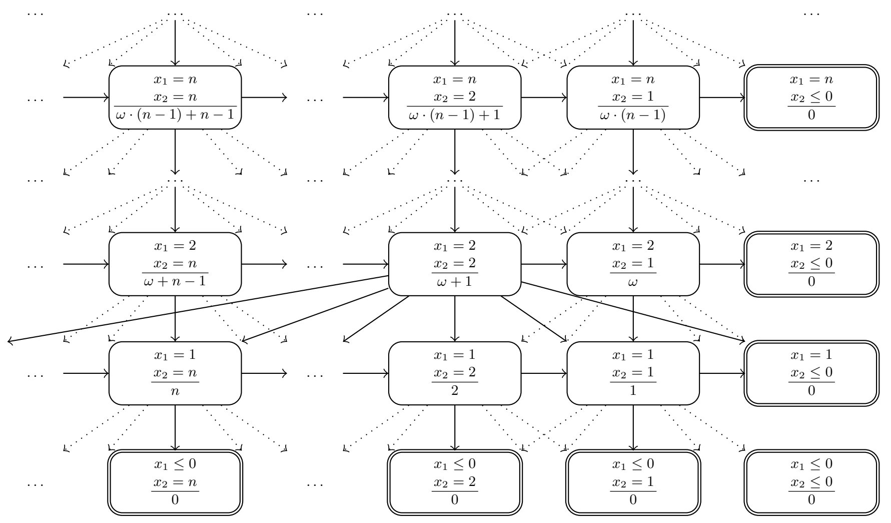
encounter we define the value of the ranking function as the maximum of all values of the function plus 1 for all successors of the state. Hence, we need a transfinite value whenever we encounter a state that leads through unbounded non-determinism to program executions of arbitrary length. In this example, in particular, we need ordinal numbers for all states where $x_1 > 1$ and $x_2 > 0$.

In Section 5 we will detail the analysis of the program by means of our abstract domain of ordinal-valued ranking functions.

It is also possible to prove the termination of the program using a lexicographic ranking function (x_1, x_2). Indeed, a lexicographic tuple $(f_n, \ldots, f_1, f_0)$ of natural numbers is an isomorphic representation of the ordinal $\omega^n \cdot f_n + \cdots + \omega \cdot f_1 + f_0$ [18]. However, reasoning directly with lexicographic ranking functions, poses the additional difficulty of finding an appropriate lexicographic order. Existing methods [1,3,5, etc.] use heuristics to explore the space of possible orders, which grows very large with the number of program variables. Instead, the interesting aspect of ordinal-valued ranking functions is that the coefficients $f_n, \ldots, f_1, f_0$ (and thus their order) are automatically inferred by the analysis. We refer to Section 7 for further discussion on the comparison between lexicographic and ordinal-valued ranking functions.

Our Contribution. In summary, in this paper we propose a parameterized abstract domain for proving termination of imperative programs by abstract interpretation. We introduce the abstract domain of *ordinal-valued* ranking functions, which we subsequently lift to *piecewise-defined* ranking functions. We also describe the implementation of an instance of the abstract domain based on affine functions, and we provide experimental evidence of its expressivity.

Outline of the Paper. Section 2 gives a brief overview of the theory of ordinals and ordinal arithmetic. In Section 3 we recall our concrete semantics, and in Section 4 we introduce the abstract domain of ordinal-valued ranking functions, which we extend to piecewise-defined ranking functions in Section 5. We describe the im-



$x_1 = n$, $x_2 = n$, $\omega \cdot (n-1) + n - 1$
$x_1 = n$, $x_2 = 2$, $\omega \cdot (n-1) + 1$
$x_1 = n$, $x_2 = 1$, $\omega \cdot (n-1)$
$x_1 = n$, $x_2 \leq 0$, 0
$x_1 = 2$, $x_2 = n$, $\omega + n - 1$
$x_1 = 2$, $x_2 = 2$, $\omega + 1$
$x_1 = 2$, $x_2 = 1$, ω
$x_1 = 2$, $x_2 \leq 0$, 0
$x_1 = 1$, $x_2 = n$, n
$x_1 = 1$, $x_2 = 2$, 2
$x_1 = 1$, $x_2 = 1$, 1
$x_1 = 1$, $x_2 \leq 0$, 0
$x_1 \leq 0$, $x_2 = n$, 0
$x_1 \leq 0$, $x_2 = 2$, 0
$x_1 \leq 0$, $x_2 = 1$, 0
$x_1 \leq 0$, $x_2 \leq 0$, 0

Fig. 2. Transitions between states at control point 1 for the program in Figure 1. There is an edge from any node where x_1 has value $k > 0$ (and $x_2 > 0$) to all nodes where x_1 has value $k - 1$ (and x_2 has any value). In every node we indicate the maximum number of loop iterations needed to reach a final state.

plementation of our prototype static analyzer and we experimentally evaluate our approach in Section 6. Section 7 discusses related work and Section 8 concludes.

2 Ordinals

A relation $<$ is *well-founded* if every $<$-decreasing sequence is finite. A *well-ordered set* is a pair $\langle X, \leq \rangle$ where $\leq$ is a *well-ordering*, i.e. a total order whose corresponding strict order $<$ is a well-founded relation over X. Two well-ordered sets $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$ are *order-isomorphic* if there is a bijection $f \colon X \to Y$ such that, for all $x_1, x_2 \in X$, $x_1 \leq_X x_2$ if and only if $f(x_1) \leq_Y f(x_2)$. Two order-isomorphic well-ordered sets are said to have the same *order type*.

An *ordinal number* is defined as the order type of a well-ordered set. In the following, we will use lower case Greek letters to denote ordinals. In particular, a well-ordered set $\langle X, \leq \rangle$ with order type α is order-isomorphic to the set $\{x \in X \mid x < \alpha\}$ of all ordinals strictly less than the ordinal α itself. In fact, this property permits the representation of each ordinal as the set of all ordinals that precede it: the smallest ordinal is $\emptyset$, denoted as 0. The *successor* of an ordinal α is defined as $\alpha \cup \{\alpha\}$ and is denoted as $\alpha + 1$. Thus, the first successor ordinal is $\{0\}$, denoted as 1. The next is $\{0, 1\}$, denoted as 2. Continuing in this manner, we obtain all natural numbers (i.e. all *finite* ordinals). A *limit ordinal* is an ordinal number which is neither zero nor a successor ordinal. The set of all natural numbers, denoted as ω, is the first limit ordinal (and the first *transfinite* ordinal).

In the following we will use $\langle \mathbb{O}, \leq \rangle$ to denote the well-ordered set of ordinals.

Ordinal Arithmetic. We recall the definition and some properties of addition, multiplication and exponentiation on ordinals [15].

Addition. Ordinal addition can be defined by transfinite induction:

$$\begin{aligned} \alpha + 0 &= \alpha && \text{(zero case)} \\ \alpha + (\beta + 1) &= (\alpha + \beta) + 1 && \text{(successor case)} \\ \alpha + \beta &= \bigcup_{\gamma < \beta} (\alpha + \gamma) && \text{(limit case)} \end{aligned}$$

Note that addition is associative, i.e. $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$, but not commutative, e.g. $1 + \omega = \omega \neq \omega + 1$.

Multiplication. Ordinal multiplication can also be defined inductively:

$$\begin{aligned} \alpha \cdot 0 &= 0 && \text{(zero case)} \\ \alpha \cdot (\beta + 1) &= (\alpha \cdot \beta) + \alpha && \text{(successor case)} \\ \alpha \cdot \beta &= \bigcup_{\gamma < \beta} (\alpha \cdot \gamma) && \text{(limit case)} \end{aligned}$$

Multiplication is associative, i.e. $(\alpha \times \beta) \times \gamma = \alpha \times (\beta \times \gamma)$, and left distributive, i.e. $\alpha \times (\beta + \gamma) = (\alpha \times \beta) + (\alpha \times \gamma)$. However, commutativity does not hold, e.g. $2 \times \omega = \omega \neq \omega \times 2$, and neither does right distributivity, e.g. $(\omega + 1) \times \omega = \omega \times \omega \neq \omega \times \omega + \omega$.

Exponentiation. We define ordinal exponentiation again by transfinite induction:

$$\begin{aligned} \alpha^0 &= 1 && \text{(zero case)} \\ \alpha^{\beta+1} &= (\alpha^\beta) \cdot \alpha && \text{(successor case)} \\ \alpha^\beta &= \bigcup_{\gamma < \beta} (\alpha^\gamma) && \text{(limit case)} \end{aligned}$$

Cantor Normal Form. Using ordinal arithmetic, we can build all ordinal numbers up to ε_0 (i.e. the smallest ordinal such that $\varepsilon_0 = \omega^{\varepsilon_0}$):

$$0, 1, 2, \ldots, \omega, \omega + 1, \omega + 2, \ldots, \omega \cdot 2, \omega \cdot 2 + 1, \omega \cdot 2 + 2, \ldots, \omega^2, \ldots, \omega^3, \ldots, \omega^\omega, \ldots$$

In the following, we will use the representation of ordinals based on Cantor Normal Form [15], i.e. every ordinal $\alpha > 0$ can be uniquely written as

$$\omega^{\beta_1} \cdot n_1 + \cdots + \omega^{\beta_k} \cdot n_k$$

where k is a natural number, the coefficients $n_1, \ldots, n_k$ are positive integers and the exponents $\beta_1 > \beta_2 > \cdots > \beta_k \geq 0$ are ordinal numbers. Throughout the rest of the paper we will consider ordinal numbers only up to ω^ω.

3 Termination Semantics

We consider a programming language that allows non-deterministic assignments and non-deterministic tests. The *operational semantics* of a program is described by a transition system $\langle \Sigma, \tau \rangle$, where Σ is the set of program states and $\tau \subseteq \Sigma \times \Sigma$ is the program transition relation. Let $\beta_\tau \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma : \langle s, s' \rangle \notin \tau\}$ denote the set of final states. The *maximal trace semantics* [6] generated by a transition system is the set of all infinite traces over the states in Σ and all finite traces that end with a state in β_τ.

The traditional method for proving program termination [12] 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 have seen that we can define a ranking function from the states of a program to ordinal numbers in an incremental way: starting from the program final states and retracing the program backwards while counting the maximum number of performed program steps as value of the function. In Section 1, we have justified the need for ordinal numbers in case of programs with unbounded non-determinism. In [10], Patrick Cousot and Radhia Cousot formalize this intuition and prove the existence of a *most precise ranking function* that can be expressed in fixpoint form by abstract interpretation of the program trace semantics. This partial function[1] $v \in \Sigma \rightharpoonup \mathbb{O}$ extracts the well-founded part of the transition relation τ: starting from the final states in β_τ and mapping each program state in Σ definitely leading to a final state (i.e. a program state such that all the traces to which it belongs end up at a final state in β_τ) to an ordinal in $\mathbb{O}$ representing an upper bound on the number of program execution steps remaining to termination. It is defined as the least fixpoint of the operator ϕ starting from the totally undefined function $\dot{\emptyset}$:

$$v \triangleq \mathsf{lfp}_{\dot{\emptyset}}^{\preccurlyeq}\ \phi$$

$$\phi(v) \triangleq \lambda s. \begin{cases} 0 & \text{if } s \in \beta_\tau \\ \mathsf{sup}\{v(s') + 1 \mid \langle s, s' \rangle \in \tau\} & \text{if } s \in \widetilde{\mathsf{pre}}(\mathsf{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $v_1 \preccurlyeq v_2 \triangleq \mathsf{dom}(v_1) \subseteq \mathsf{dom}(v_2) \ \wedge\ \forall x \in \mathsf{dom}(v_1) : v_1(x) \leq v_2(x)$ and $\widetilde{\mathsf{pre}}(X) \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma : \langle s, s' \rangle \in \tau \Rightarrow s' \in X\}$. Therefore v is a partial function the domain $\mathsf{dom}(v)$ of which is the set of states definitely leading to program termination: any trace starting in a state $s \in \mathsf{dom}(v)$ must terminate in at most $v(s)$ execution steps, while at least one trace starting in a state $s \notin \mathsf{dom}(v)$ does not terminate. Note that whenever $v(s) \geq \omega$, the programs still terminates, but the number of program execution steps before termination is unbounded.

The ranking function v constitutes a program semantics which is sound and complete to prove program termination (see [10]). However, it is usually not computable. In the following, we will present a decidable abstraction of v. The

[1] $A \rightharpoonup B$ is the set of partial maps from a set A to a set B.

abstraction uses the following approximation order (see [9] for further discussion on approximation and computational orders of an abstract domain):

$$v_1 \sqsubseteq v_2 \triangleq \mathsf{dom}(v_1) \supseteq \mathsf{dom}(v_2) \wedge \forall x \in \mathsf{dom}(v_2) : v_1(x) \leq v_2(x).$$

It computes an over-approximation of the value of the function v but it under-approximates its domain of definition $\mathsf{dom}(v)$. In this way, the abstraction provides sufficient preconditions for program termination: if the abstraction is defined on a program state, then all program execution traces branching from that state are definitely terminating.

4 Ordinal-Valued Ranking Functions

We derive an approximate program semantics by abstract interpretation [8]. First, we introduce the abstract domain of ordinal-valued ranking functions O, which abstracts ranking functions in $\Sigma \rightharpoonup \mathbb{O}$ by abstract ranking functions $o^\# \in \mathcal{O}^\#$ attached to program control points. Then, in the next section, we employ state partitioning to lift this abstraction to piecewise-defined ranking functions.

Let $\mathcal{X}$ be a finite set of program variables. We split the program state space Σ into program control points $\mathcal{L}$ and environments $\mathcal{S} \triangleq \mathcal{X} \to \mathbb{Z}$, 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 function $o \in \mathcal{S} \rightharpoonup \mathbb{O}$ defined on the environments related to the program control point l:

$$\langle \mathcal{S} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle \xleftarrow{\gamma_\mathsf{O}} \langle \mathcal{O}^\#, \sqsubseteq_\mathsf{O} \rangle.$$

Natural-Valued Functions. We assume we are given an abstraction $\langle \mathcal{S}^\#, \sqsubseteq_\mathsf{S} \rangle$ of environments: $\langle \mathcal{P}(\mathcal{S}), \subseteq \rangle \xleftarrow{\gamma_\mathsf{S}} \langle \mathcal{S}^\#, \sqsubseteq_\mathsf{S} \rangle$ (i.e. any abstract domain such as intervals [7], octagons [19], polyhedra [11], ...), and an abstraction $\langle \mathcal{S}^\# \times \mathcal{F}^\#, \sqsubseteq_\mathsf{F} \rangle$ of $\langle \mathcal{S} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$ by means of *natural-valued* functions of the program variables:

$$\langle \mathcal{S} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle \xleftarrow{\gamma_\mathsf{F}} \langle \mathcal{S}^\# \times \mathcal{F}^\#, \sqsubseteq_\mathsf{F} \rangle.$$

More specifically, the abstraction $\langle \mathcal{S}^\# \times \mathcal{F}^\#, \sqsubseteq_\mathsf{F} \rangle$ encodes a partial function $v \in \mathcal{S} \rightharpoonup \mathbb{O}$ by a pair of an abstract state $s^\# \in \mathcal{S}^\#$ and a natural-valued function (e.g. an affine function) of the program variables $f^\# \in \mathcal{F}^\#$ [22]. 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 $\mathcal{O}^\# \triangleq \mathcal{S}^\# \times \mathcal{P}^\#$ where

$$\mathcal{P}^\# \triangleq \{\bot_\mathsf{P}\} \ \cup\ \{p^\# \mid p^\# = \sum_i \omega^i \cdot f_i^\#, f_i^\# \in \mathcal{F}^\#\} \ \cup\ \{\top_\mathsf{P}\}$$

is the set of ordinal-valued ranking functions of the program variables (in addition to the function $\bot_{\mathsf{P}}$ representing potential non-termination, and the function $\top_{\mathsf{P}}$ representing the lack of enough information to conclude). 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 number in Cantor Normal Form) $p^{\#}$:

$$p^{\#} \triangleq \omega^k \cdot f_k^{\#} + \cdots + \omega^2 \cdot f_2^{\#} + \omega \cdot f_1^{\#} + f_0^{\#} \qquad\qquad k > 0$$

where the coefficients $f_0^{\#}, f_1^{\#}, \ldots, f_k^{\#}$ belong to $\mathcal{F}^{\#}$. In the following, with abuse of notation, we use a map $s^{\#} \mapsto p^{\#}$ to denote the pair of $s^{\#} \in \mathcal{S}^{\#}$ and $p^{\#} \in \mathcal{P}^{\#}$.

The abstract domain O is parameterized by the choices of the state abstraction $\langle \mathcal{S}^{\#}, \sqsubseteq_{\mathsf{S}} \rangle$, the maximum degree k of the polynomial, and the type (e.g. affine, quadratic, cubic, exponential, ...) of functions used as polynomial coefficients $f_0^{\#}, f_1^{\#}, f_2^{\#}, \ldots, f_n^{\#}$.

Concretization Function. The concretization function $\gamma_{\mathsf{O}} \in \mathcal{O}^{\#} \to (\mathcal{S} \rightharpoonup \mathbb{O})$ depends on γ_{S}, which maps an abstract state $s^{\#} \in \mathcal{S}^{\#}$ to the corresponding set of program environments, and on γ_{F}, which maps a relation $v^{\#} \in \mathcal{S}^{\#} \times \mathcal{F}^{\#}$ to the corresponding partial function $v \in \mathcal{S} \rightharpoonup \mathbb{O}$:

$$\begin{aligned}
&\gamma_{\mathsf{O}}(s^{\#} \mapsto \bot_{\mathsf{P}}) = \dot{\emptyset} \\
&\gamma_{\mathsf{O}}(s^{\#} \mapsto p\#) = \lambda s \in \gamma_{\mathsf{S}}(s^{\#}).\ p^{\#}(s) \\
&\qquad\qquad \text{where } p^{\#}(s) = \sum_{i \leq k} \omega^i \cdot \gamma_{\mathsf{F}}(s^{\#} \mapsto f_i^{\#})(s) \\
&\gamma_{\mathsf{O}}(s^{\#} \mapsto \top_{\mathsf{P}}) = \dot{\emptyset}
\end{aligned}$$

where $\dot{\emptyset}$ denotes the totally undefined function. Note that the concretization function γ_{O} forgets about all program states that are potentially non-terminating ($\bot_{\mathsf{P}}$) and all program states for which there is not enough information ($\top_{\mathsf{P}}$). This agrees with our goal to under-approximate the domain of definition of the most precise ranking function (cf. Section 3).

Order. To compare two abstract functions, we define the abstract approximation order $\sqsubseteq_{\mathsf{O}}$ as the abstract counterpart of the approximation order $\sqsubseteq$:

$$(s_1^{\#} \mapsto p_1^{\#}) \sqsubseteq_{\mathsf{O}} (s_2^{\#} \mapsto p_2^{\#}) \triangleq s_2^{\#} \sqsubseteq_{\mathsf{S}} s_1^{\#} \wedge p_1^{\#} \sqsubseteq_{\mathsf{P}} p_2^{\#}$$

where $p_1^{\#} \sqsubseteq_{\mathsf{P}} p_2^{\#} \triangleq \forall s \in \gamma_{\mathsf{S}}(s_2^{\#}) : p_1^{\#}(s) \leq p_2^{\#}(s)$.

In order for an abstract function $o_1^{\#}$ to be smaller than an abstract function $o_2^{\#}$, we require the domain of $o_2^{\#}$ to be included in the domain of $o_1^{\#}$ ($s_2^{\#} \sqsubseteq_{\mathsf{S}} s_1^{\#}$) and, for all states in the domain of $o_2^{\#}$, we require $o_1^{\#}$ to have smaller (or equal) value than $o_2^{\#}$ ($p_1^{\#} \sqsubseteq_{\mathsf{P}} p_2^{\#}$). The relative precision between abstract functions is preserved by the concretization function γ_{O}:

$$(s_1^{\#} \mapsto p_1^{\#}) \sqsubseteq_{\mathsf{O}} (s_2^{\#} \mapsto p_2^{\#}) \Rightarrow \gamma_{\mathsf{O}}(s_1^{\#} \mapsto p_1^{\#}) \sqsubseteq \gamma_{\mathsf{O}}(s_2^{\#} \mapsto p_2^{\#})$$

Join. 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 (unless we have already reached the maximum degree for the polynomial, in which case we abandon to $\top_\mathsf{P}$).

Example 1. Let $\mathcal{X} = \{x_1, x_2\}$ and let $\langle \mathcal{S}^\# \times \mathcal{F}^\#, \sqsubseteq_\mathsf{F} \rangle$ be an abstraction of $\langle \mathcal{S} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$ that uses intervals [7] as state abstraction and affine functions as abstract functions $f^\# \in \mathcal{F}^\#$ [22]. We consider the join of the abstract functions:

$$o_1^\# \triangleq s_1^\# \mapsto p_1^\# \triangleq [-\infty, +\infty] \mapsto \omega \cdot x_1 + x_2$$
$$o_2^\# \triangleq s_2^\# \mapsto p_2^\# \triangleq [-\infty, +\infty] \mapsto \omega \cdot (x_1 - 1) - x_2$$

Their common domain is trivially $s^\# \triangleq [-\infty, +\infty]$. The unification of the two polynomials $p_1^\#$ and $p_2^\#$ starts from joining the functions $f_{0_1}^\# \triangleq x_2$ and $f_{0_2}^\# \triangleq -x_2$. However, there does not exist a *natural-valued* affine function $f_0^\#$ greater than $f_{0_1}^\#$ and $f_{0_2}^\#$ for all possible values of x_2 (since $s^\# \triangleq [-\infty, +\infty]$). We force $f_0^\#$ to equal 0 and we carry 1 to the unification of $f_{1_1}^\# \triangleq x_1$ and $f_{1_2}^\# \triangleq x_1 - 1$ which becomes $f_1^\# = x_1 + 1$ (i.e. x_1 after the unification, and $x_1 + 1$ after carrying). The result of the join is $o^\# \triangleq [-\infty, +\infty] \mapsto \omega \cdot (x_1 + 1)$. □

Intuitively, whenever natural-valued functions are not sufficient, we naturally resort to ordinal numbers. 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 (and applying the limit case of ordinal multiplication): $\omega^k \cdot f^\# = \omega^k \cdot \omega = \omega^{k+1} \cdot 1 + \omega^k \cdot 0 = \omega^{k+1}$.

Widening. The widening operator $\triangledown_\mathsf{O}$ summarizes two abstract functions $o_1^\# \triangleq s_1^\# \mapsto p_1^\#$ and $o_2^\# \triangleq s_2^\# \mapsto p_2^\#$ into a single one $o^\# \triangleq s^\# \mapsto p^\#$, where $s^\# \triangleq s_1^\# \sqcup_\mathsf{S} s_2^\#$ and $p^\# \triangleq p_1^\# \sqcup_\mathsf{P} p_2^\#$ (unless the two abstract functions are already defined on the same abstract state — i.e. $s_1^\# =_\mathsf{S} s_2^\#$ — and $p_1^\# \sqsubset_\mathsf{P} p_2^\#$, in which case $p^\# \triangleq \top_\mathsf{P}$ to ensure convergence). Note that $\triangledown_\mathsf{O}$ differs from the join $\sqcup_\mathsf{O}$ in that it widens the abstract state $s^\#$ to the union of $s_1^\#$ and $s_2^\#$. Indeed, the join is an upper-bound with respect to the approximation order $\sqsubseteq$, while the widening is an upper-bound with respect to the computational order $\preccurlyeq$ (cf. Section 3).

Assignments. 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, and is preceded by the addition of 1 to the polynomial constant (to take into account that one more program step is needed before termination). 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.

Example 2. Let $\mathcal{X} = \{x_1, x_2\}$. We consider the result of the non-deterministic assignment $x_1 :=\ ?$ to the polynomial $p^\# \triangleq \omega \cdot x_1 + x_2$. First, we add 1 to the function $f_0^\# \triangleq x_2$ to *count* the assignment as an additional step needed before termination. Then, we perform the assignment on the terms of the polynomial: the function $f_0^\#$ remains unchanged (since the assignment involves only the variable x_1), whereas the coefficient $f_1^\# \triangleq x_1$ of ω is reset to 0 and carries 1 to the term with next higher degree ω^2. In fact, the assignment $x_1 :=\ ?$ allows x_1 (and consequently $f_1^\#$) to take *any* value, but there does not exist a *natural-valued* function that properly abstracts all possible outcomes of the assignment. The resulting polynomial is $\bar{p}^\# \triangleq \omega^2 \cdot 1 + \omega \cdot 0 + x_2 + 1 = \omega^2 + x_2 + 1$. □

Tests. Test statements only affect the abstract states $s^\# \in \mathcal{S}^\#$ (and are managed by the state abstraction) and leave unchanged the polynomials $p^\# \in \mathcal{P}^\#$.

5 Piecewise-Defined Ranking Functions

In the following, we will briefly recall the abstract domain of piecewise-defined ranking functions [22]. Then, we describe our extension of this domain using the ordinal-valued ranking functions we presented in Section 4.

5.1 Piecewise-Defined Natural-Valued Ranking Functions

In [22], a decidable abstraction of the most precise ranking function $v \in \Sigma \rightharpoonup \mathbb{O}$ (cf. Section 3) is provided by the abstract domain $\mathsf{V}(\mathsf{F}(\mathsf{S}))$, where V is a functor abstract domain parameterized by S, an abstract domain for states, and F, an abstract domain based on natural-valued functions of the program variables.

The elements of the abstract domain belong to $\mathcal{V}^\# \triangleq \mathcal{P}(\mathcal{S}^\# \times \mathcal{F}^\#)$, where $\mathcal{S}^\#$ is the set of abstract program states (e.g. intervals [7]) and $\mathcal{F}^\#$ is the set of natural-valued functions of the program variables (e.g. affine functions). More specifically, an element $v^\# \in \mathcal{V}^\#$ has the form:

$$v^\# \triangleq \begin{cases} s_1^\# \mapsto f_1^\# \\ \quad\vdots \\ s_k^\# \mapsto f_k^\# \end{cases}$$

where the abstract states $s_1^\#, \dots, s_k^\#$ induce a partition of the space of environments $\mathcal{S} \triangleq \mathcal{X} \to \mathbb{Z}$, and $f_1^\#, \dots, f_k^\#$ are ranking functions.

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 approximation order $\sqsubseteq_{\mathsf{V}}$ and the computational order $\preccurlyeq_{\mathsf{V}}$ return the conjunction of the piecewise comparisons. The piecewise join $\sqcup_{\mathsf{V}}$ computes the piecewise-defined natural-valued ranking function greater than $v_1^\#$ and $v_2^\#$. The piecewise widening $\triangledown_{\mathsf{V}}$ summarizes *adjacent* pieces of a function joining them into a single one. 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 going to $\top$ on the partitions where the value of the ranking function has increased between iterations.

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 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. As a consequence of the soundness of all abstract operators (see [22]), 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.

However, since the abstract domain V is limited to ranking functions over natural numbers, all program traces with unbounded non-determinism are disregarded by the abstraction. As a result, the abstract domain is not able to prove the termination of programs as the one in Figure 1. In the following, we describe how we extend this abstract domain to ranking functions over ordinal numbers.

5.2 Piecewise-Defined Ordinal-Valued Ranking Functions

We propose the abstract domain $\mathsf{V}(\mathsf{O}(\mathsf{F}(\mathsf{S})))$ obtained by extending $\mathsf{V}(\mathsf{F}(\mathsf{S}))$ with the domain of ordinal-valued ranking functions O presented in Section 4.

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 $s_1^\#, \dots, s_k^\# \in \mathcal{S}^\#$ induce a partition of the space of environments $\mathcal{S}$ and $p_1^\#, \dots, p_k^\#$ 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 $f_0^\#, f_1^\#, f_2^\#, \dots, f_n^\# \in \mathcal{F}^\#$ are natural-valued functions of the program variables.

The partition unification algorithm of $\mathsf{V}(\mathsf{O}(\mathsf{F}(\mathsf{S})))$ works exactly in the same way as that of $\mathsf{V}(\mathsf{F}(\mathsf{S}))$, while the piecewise operators of the domain now use the operators of O (which in turn exploit the operators of F for the polynomial coefficients) instead of using directly those of F. The soundness of all abstract operators of $\mathsf{V}(\mathsf{O}(\mathsf{F}(\mathsf{S})))$ follows by the soundness of all abstract operators of $\mathsf{V}(\mathsf{F}(\mathsf{S}))$, and by the soundness of all abstract operators of O with respect to ordinal arithmetic. Therefore, the abstract domain $\mathsf{V}(\mathsf{O}(\mathsf{F}(\mathsf{S})))$ is suitable to prove program termination and is more powerful than $\mathsf{V}(\mathsf{F}(\mathsf{S}))$ because it overcomes the limitations of natural-valued ranking functions. Indeed, $\mathsf{V}(\mathsf{O}(\mathsf{F}(\mathsf{S})))$ is able to prove the termination of the non-deterministic program in Figure 1.

Motivating Example (continued). Due to space constraints, we describe in some detail only a few interesting iterations of the (backward) analysis of the program in Figure 1. We invite the interested reader to refer to our prototype implementation [21] for a more complete and detailed program analysis.

The starting point is the constant function $f_6(x_1, x_2) = 0$ at the program final control point 6. We use a widening delay of 3 iterations. At the fourth iteration, the ranking function at the loop control point 1 is:

$$f_1^4(x_1, x_2) = \begin{cases} 1 & x_1 \leq 0 \vee x_2 \leq 0 \\ 3x_2 + 2 & x_1 = 1 \\ \bot & \text{otherwise} \end{cases}$$

In the second case, the function $3x_2 + 2$ comes out as a result of the widening between adjacent pieces with consecutive values for x_2 (i.e. between the pieces $(x_1 = 1 \ \wedge \ x2 = 1) \mapsto 5$ and $(x_1 = 1 \ \wedge \ x2 = 2) \mapsto 8$).

Ordinal numbers appear for the first time at program control point 4 due to the non-deterministic assignment to x_2:

$$f_4^4(x_1, x_2) = \begin{cases} 2 & x_1 \leq 0 \\ \omega & x_1 = 1 \\ \bot & \text{otherwise} \end{cases}$$

In the first case, the value of the function is simply increased (to count one more program step before termination) but (since x_2 can now have any value) its domain is modified forgetting all constraints on x_2 (i.e. $x_2 \leq 0$). In the second case, $3x_2 + 2$ is increased to $3x_2 + 3$ (to count one more program step) which then becomes ω (due to approximation of the non-deterministic assignment).

At the seventh iteration, at control point 1, we obtain the ranking function:

$$f_1^7(x_1, x_2) = \begin{cases} 1 & x_1 \leq 0 \vee x_2 \leq 0 \\ 3x_2 + 2 & x_1 = 1 \\ \omega + 3x_2 + 9 & x_1 = 2 \\ \bot & \text{otherwise} \end{cases}$$

as a result of the widening between the preceding iterate f_1^6:

$$f_1^6(x_1, x_2) = \begin{cases} 1 & x_1 \leq 0 \vee x_2 \leq 0 \\ 3x_2 + 2 & x_1 = 1 \\ \omega + 12 & x_1 = 2 \wedge x_2 = 1 \\ \omega + 15 & x_1 = 2 \wedge x_2 = 2 \\ \bot & \text{otherwise} \end{cases}$$

and the ranking function $f_1^{6'}$, obtained from f_1^6 after one loop iteration:

$$f_1^{6'}(x_1, x_2) = \begin{cases} 1 & x_1 \leq 0 \vee x_2 \leq 0 \\ 3x_2 + 2 & x_1 = 1 \\ \omega + 12 & x_1 = 2 \wedge x_2 = 1 \\ \omega + 15 & x_1 = 2 \wedge x_2 = 2 \\ \omega + 18 & x_1 = 2 \wedge 3 \leq x_2 \\ \bot & \text{otherwise} \end{cases}$$

In particular, widening occurs between the pieces where $x_1 = 2$. It is performed in ascending powers of ω: from the constants 12, 15 and 18 (all corresponding to consecutive values for x_2), it infers the affine function $3x_2 + 9$ (by classic join of affine functions); then, since for all pieces the coefficient of ω is equal to 1, the inferred coefficient of ω is again 1. Thus, the result of the widening for $x_1 = 2$ is $\omega + 3x_2 + 9$.

Finally, at the eleventh iteration, we reach a fixpoint f_1^{11}:

$$f_1^{11}(x_1, x_2) = \begin{cases} 1 & x_1 \leq 0 \vee x_2 \leq 0 \\ 3x_2 + 2 & x_1 = 1 \\ \omega + 3x_2 + 9 & x_1 = 2 \\ \omega \cdot (x_1 - 1) + 7x_1 + 3x_2 - 5 & \text{otherwise} \end{cases}$$

Note that the second and third expressions are particular cases (for $x_1 = 1$ and $x_1 = 2$ respectively) of the last expression and are explicitly listed only due to the amount of widening delay we used. The function $f(x_1, x_2) = \omega \cdot (x_1 - 1) + 7x_1 + 3x_2 - 5$ constitutes a ranking function[2] for the program loop, while the first case represents immediate program exit (without even entering the loop). □

[2] The reason why we obtain a different ranking function with respect to Figure 2 is because we count the number of program execution steps whereas, for convenience of presentation, in Figure 2 we count the number of loop iterations.

6 Implementation

We have incorporated the implementation of our abstract domain for ordinal-valued ranking functions O into our prototype static analyzer [21] based on piecewise-defined ranking functions.

The prototype accepts programs written in a small non-deterministic while-language. It is written in OCaml and, at the time of writing, the available abstraction for program states S is based on intervals [7] and the available abstraction for natural-valued functions F is based on affine functions represented as convex polyhedra [11]. The operators for the intervals and convex polyhedra abstract domains are provided by the Apron library [14]. The extension to ordinal-valued ranking functions is optional, but when activated it requires to choose a maximum degree for the abstract polynomials. It is also possible to tune the precision of the analysis by adjusting the widening delay.

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.

6.1 Examples

To illustrate the expressiveness of our domain, we consider two more examples, besides the one shown in Section 5.

Example 3. Let us consider the program in Figure 3 which is an involved variation of the one 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 behaves exactly as in Figure 1. In case x_1 is negative, the program either increments the value of x_1 or it 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 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 at the program control point 5 (in the first branch of the first if statement, i.e. if $x_1 > 0$) while the value of x_1 is reset at the control point 10 (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. Let us consider the graph in Figure 5. Whenever x_2 is reset to any value, we move towards the final states decreasing the value of x_1, and whenever x_1 is reset to any value, we move towards the final states decreasing the value of x_2. Moreover, whenever x_1 is reset to a positive value, its value will only decrease until it reaches zero (or x_2 is reset to a value less than zero).

Our prototype is able to prove the program termination in about 10 seconds (with a widening delay of 3 iterations). We automatically infer the following

```
int : $x_1, x_2$
while $^1$( $x_1 \neq 0 \land x_2 \geq 0$ ) do
    if $^2$( $x_1 > 0$ ) then
        if $^3$( ? ) then
            $^4x_1 := x_1 - 1$
            $^5x_2 :=$ ?
        else
            $^6x_2 := x_2 - 1$
    else
        if $^7$( ? ) then
            $^8x_1 := x_1 + 1$
        else
            $^9x_2 := x_2 - 1$
            $^{10}x_1 :=$ ?
od$^{11}$
```

Fig. 3. Program with no lexicographic ranking function.

```
int : $x_1, x_2$
$^1x_1 := N$
while $^2$( $x_1 \geq 0$ ) do
    $^3x_2 := N$
    while $^4$( $x_2 \geq 0$ ) do
        $^5x_2 := x_2 - 1$
    od
    $^6x_1 := x_1 - 1$
od$^7$
```

Fig. 4. Program with non-linear computational complexity.

piecewise-defined ranking function:

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

In Figure 5, we justify the need for ω^2. Indeed, from any state where $x_1 < 0$ and $x_2 = k_2 > 0$, whenever x_1 is reset at program control point 10, it is possible to jump to any state where $x_2 = k_2 - 1$. In particular, for example from the state where $x_1 = -1$ and $x_2 = 2$, it is possible to jump through unbounded non-determinism to states with value of the most precise ranking function equal to an arbitrary ordinal number between ω and ω^2, which requires ω^2 as upper bound of the maximum number of loop iterations needed to reach a final state.

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

Example 4. Let us consider the program in Figure 4. Since the program has quadratic time complexity, we cannot prove its termination limiting ourselves to piecewise-defined natural-valued *affine* ranking functions.

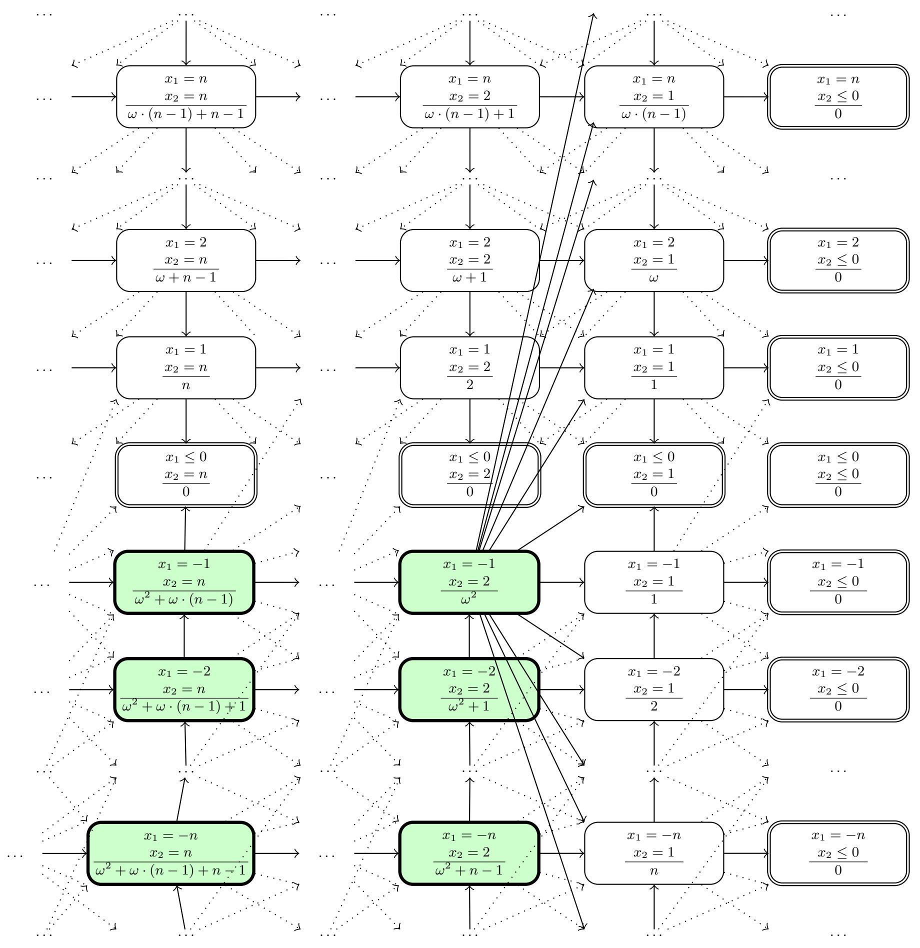

$x_1 = n$, $x_2 = n$, $\omega \cdot (n-1) + n - 1$
$x_1 = n$, $x_2 = 2$, $\omega \cdot (n-1) + 1$
$x_1 = n$, $x_2 = 1$, $\omega \cdot (n-1)$
$x_1 = n$, $x_2 \leq 0$, 0
$x_1 = 2$, $x_2 = n$, $\omega + n - 1$
$x_1 = 2$, $x_2 = 2$, $\omega + 1$
$x_1 = 2$, $x_2 = 1$, ω
$x_1 = 2$, $x_2 \leq 0$, 0
$x_1 = 1$, $x_2 = n$, n
$x_1 = 1$, $x_2 = 2$, 2
$x_1 = 1$, $x_2 = 1$, 1
$x_1 = 1$, $x_2 \leq 0$, 0
$x_1 \leq 0$, $x_2 = n$, 0
$x_1 \leq 0$, $x_2 = 2$, 0
$x_1 \leq 0$, $x_2 = 1$, 0
$x_1 \leq 0$, $x_2 \leq 0$, 0
$x_1 = -1$, $x_2 = n$, $\omega^2 + \omega \cdot (n-1)$
$x_1 = -1$, $x_2 = 2$, ω^2
$x_1 = -1$, $x_2 = 1$, 1
$x_1 = -1$, $x_2 \leq 0$, 0
$x_1 = -2$, $x_2 = n$, $\omega^2 + \omega \cdot (n-1) + 1$
$x_1 = -2$, $x_2 = 2$, $\omega^2 + 1$
$x_1 = -2$, $x_2 = 1$, 2
$x_1 = -2$, $x_2 \leq 0$, 0
$x_1 = -n$, $x_2 = n$, $\omega^2 + \omega \cdot (n-1) + n - 1$
$x_1 = -n$, $x_2 = 2$, $\omega^2 + n - 1$
$x_1 = -n$, $x_2 = 1$, n
$x_1 = -n$, $x_2 \leq 0$, 0

Fig. 5. Transitions between states at control point 1 for the program in Figure 3. There is an edge from any node where x_1 has value $k_1 > 0$ (and $x_2 > 0$) to all nodes where x_1 has value $k_1 - 1$ (and x_2 has any value); there is also an edge from any node where x_2 has value $k_2 > 0$ (and $x_1 < 0$) to all nodes where x_2 has value $k_2 - 1$ (and x_1 has any value). In every node we indicate the maximum number of loop iterations needed to reach a final state: the highlighted nodes require an ordinal greater than ω^2.

Program	Ranking Function
[1, Figure 1]	$f(x, y) = 57x + 3y + 28$ Time: < 3 s (Widening Delay: 2)
[4, Figure 1]	$f(x, y) = 7y + 3x - 5$ Time: < 1 s (Widening Delay: 2)
[5, Figure 7a]	$f(x, y, d) = \omega \cdot (y - 1) + 4x + 9y - 7$ Time: < 60 s (Widening Delay: 3)
[5, Figure 7b]	$f(x, y, z) = \omega^2 \cdot (y - 1) + \omega \cdot (y + z - 2) + 3x + 13y + 8z - 18$ Time: < 240 s (Widening Delay: 3)
[5, Figure 8a]	$f(x) = \begin{cases} -3x + 1 & x < 0 \\ 1 & x = 0 \\ 3x + 1 & x > 0 \end{cases}$ Time: < 1 s (Widening Delay: 2)
[23, `MirrorIntervSim`]	$f(x) = \begin{cases} 4 & x \leq -6 \\ \bot & -5 \leq x \leq -1 \\ 1 & x = 0 \\ 6 & x = 1 \\ 5x + 1 & 2 \leq x \leq 30 \\ \bot & 31 \leq x \leq 35 \\ 4 & x \geq 36 \end{cases}$ Time: < 1 s (Widening Delay: 2)

Fig. 6. Some of the benchmarks used in experiments.

However, with the extension to ordinal-numbers, our prototype analyzes the program in about 2 seconds (with a widening delay of 2 iterations). The inferred ranking function is $f(x_1, x_2) = \omega + 2$, where ω constitutes an upper-bound on the number of program execution steps spent inside the while loop (i.e. it testifies that the number of execution step spent inside the while loop is *finite*), and the constant 2 takes into account the initialization step $x_1 := N$ and the test $x_1 < 0$ that enforces loop exit. This ordinal-valued ranking function represents a rough approximation of the program computational complexity, but allows nonetheless to prove its termination without requiring to handle ranking function more complex than affine functions. □

6.2 Experiments

We have evaluated our prototype implementation against a set of benchmarks collected from publications in the area [1,4,5,23, etc.] or inspired from real code.

We analyzed 38 examples: 25 terminating loops and 13 conditionally terminating loops. Among them, nine are "simple loops" (i.e. loops with only variable updates in the loop body) and seven are nested loops. We successfully proved (conditional) termination for all simple loops but one, and for four out of the seven nested loops. Only two nested loops required the extension to ordinal numbers (to cope with a non-linear computational complexity). 13 of the 38 benchmarks are non-deterministic programs. We proved termination for ten of them, using ordinal numbers in five cases. In the other five cases, piecewise-defined natural-valued ranking functions were sufficient (even when the programs were presented in the literature as requiring a lexicographic ranking function). In summary, our prototype was able to automatically solve 30 of the 38 benchmarks we considered. Failure in the eight missing cases is due to the non-relational nature of partitioning with intervals and, in particular, is not related to the use of ordinal numbers. Almost all examples were analyzed in less than 60 seconds (only one example took aroud 240 seconds), with maximum polynomial degree of two and maximum widening delay of seven iterations. In Figure 6 are depicted some representatives of the benchmarks, together with our results.

7 Related Work

In the recent past, a large body of work has been devoted to proving program termination of imperative programs. To the best of our knowledge, in this setting, the inference of ordinal-valued ranking functions is unique to our work.

Aside from the use of ordinal numbers, the approach presented in this paper is mostly related to [1]: both techniques handle programs with arbitrary structure and infer ranking functions (that also provide information on the program computational complexity in terms of executions steps) attached to program control points. The technique proposed in [1] uses invariants (pre)computed for each program control point to infer lexicographic ranking functions (also attached to program points). On the other hand, with our approach, we infer ordinal-valued ranking functions *directly* as invariants attached to program control points. In [1], the problem of finding an appropriate lexicographic order is handled by a greedy algorithm, whereas dealing with ordinal numbers relieves us from the burden of finding lexicographic orders (cf. Section 1). In contrast, ordinal-valued ranking functions parameterized by functions with limited expressivity (e.g. affine functions) might produce a rough approximation of the program computational complexity (cf. Example 4). We plan to study these issues further and support non-linear functions as part of our future work.

In order to avoid lexicographic ranking functions, many other approaches rely on the transition invariants method introduced in [20]. The advantage of this method is that it only requires to find a *set* of ranking functions, without lexicographic ordering between them. However, the main drawback of the method is the cost of explicitly checking the validity of the termination argument. On the other hand, our approach also avoids explicit lexicographic orders, but in addition it does not suffer from this disadvantage because the validity of the termination argument is automatically enforced at each loop iteration.

In a different context, a large amount of research followed the introduction of size-change termination (SCT) [17]. The SCT approach consists in collecting a set of size-change graphs (representing function calls) and combining them into multipaths (representing program executions) in such a way that at least one variable is guaranteed to decrease. Compared to SCT, our approach avoids the exploration of the combinatorial space of multipaths with the explicit manipulation of ordinal numbers. In [16,2], algorithms are provided to derive explicit ranking functions from size-change graphs, but these ranking functions have a shape quite different from ours which makes it difficult for us to compare their expressiveness. For example, the derived ranking functions use lexicographic orders on variables while our polynomial coefficients are arbitrary linear combinations of variables. In general, an in-depth comparison between such fairly different methods is an open research topic (e.g. see [13] for the comparison of the transition invariants and the size-change termination methods).

Finally, we have seen that there exist programs (e.g., the program in Figure 3) for which there does not exist a lexicographic ranking function. In [5] the authors discuss the problem and propose some heuristics to circumvent it. Interestingly these heuristics rediscover exactly the need for piecewise-defined ranking functions, even if implicitly and in a roundabout way.

8 Conclusion

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.

We also described the implementation of an instance of the abstract domain based on affine functions, and we have provided experimental evidence of its expressivity. In particular, we have seen that inferring ranking functions over ordinals removes the burden of finding lexicographic orders (cf. Section 1 and Section 5), and overcomes the limitations of affine functions in case of programs with non-linear computational complexity (cf. Example 4). Finally, we have seen (cf. Example 3) that piecewise-defined ordinal-valued ranking functions are crucial where lexicographic ranking functions are not powerful enough.

It remains for future work to support *non-linear* functions (e.g. quadratic, cubic, exponential, . . .) and *relational* abstract domains (e.g. octagons [19], polyhedra [11], . . .) for better state partitioning.

Acknowledgments. We are very grateful to Damien Massé for the interesting discussions and his helpful suggestions. We also thank the anonymous reviewers for their careful reviews and their useful comments.

References

1. C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-Dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. In *SAS*, pages 117–133, 2010.
2. A. M. Ben-Amram and C. S. Lee. Ranking Functions for Size-Change Termination II. *Logical Methods in Computer Science*, 5(2), 2009.
3. A. R. Bradley, Z. Manna, and H. B. Sipma. Linear Ranking with Reachability. In *CAV*, pages 491–504, 2005.
4. B. Cook, A. Podelski, and A. Rybalchenko. Proving Program Termination. *Communications of the ACM*, 54(5):88–98, 2011.
5. B. Cook, A. See, and F. Zuleger. Ramsey vs. Lexicographic Termination Proving. In *TACAS*, pages 47–61, 2013.
6. P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Electronic Notes in Theoretical Computer Science*, 6:77–102, 1997.
7. P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Programs. In *Symposium on Programming*, pages 106–130, 1976.
8. P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*, pages 238–252, 1977.
9. P. Cousot and R. Cousot. Higher Order Abstract Interpretation (and Application to Comportment Analysis Generalizing Strictness, Termination, Projection, and PER Analysis. In *ICCL*, pages 95–112, 1994.
10. P. Cousot and R. Cousot. An Abstract Interpretation Framework for Termination. In *POPL*, pages 245–258, 2012.
11. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*, pages 84–96, 1978.
12. R. W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.
13. M. Heizmann, N. D. Jones, and A. Podelski. Size-Change Termination and Transition Invariants. In *SAS*, pages 22–50, 2010.
14. B. Jeannet and A. Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *CAV*, pages 661–667, 2009.
15. K. Kunen. *Set Theory: An Introduction to Independence Proofs.* Studies in Logic and the Foundations of Mathematics. 1980.
16. C. S. Lee. Ranking Functions for Size-Change Termination. *ACM Transactions on Programming Languages and Systems*, 31(3), 2009.
17. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The Size-Change Principle for Program Termination. In *POPL*, pages 81–92, 2001.
18. Z. Manna and A. Pnueli. The Temporal Verification of Reactive Systems: Progress, 1996.
19. A. Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
20. A. Podelski and A. Rybalchenko. Transition Invariants. In *LICS*, pages 32–41, 2004.
21. C. Urban. FuncTion. `http://www.di.ens.fr/~urban/FuncTion.html`.
22. C. Urban. The Abstract Domain of Segmented Ranking Functions. In *SAS*, pages 43–62, 2013.
23. H. Velroyen and P. Rümmer. Non-Termination Checking for Imperative Programs. In *TAP*, pages 154–170, 2008.