A Decision Tree Abstract Domain for Proving Conditional Termination

Caterina Urban, Antoine Miné

A Decision Tree Abstract Domain for Proving Conditional Termination*

Caterina Urban and Antoine Miné

ÉNS & CNRS & INRIA, France
{urban,mine}@di.ens.fr

Abstract. We present a new parameterized abstract domain able to refine existing numerical abstract domains with finite disjunctions. The elements of the abstract domain are decision trees where the decision nodes are labeled with linear constraints, and the leaf nodes belong to a numerical abstract domain.
The abstract domain is parametric in the choice between the expressivity and the cost of the linear constraints for the decision nodes (e.g., polyhedral or octagonal constraints), and the choice of the abstract domain for the leaf nodes. We describe an instance of this domain based on piecewise-defined ranking functions for the automatic inference of sufficient preconditions for program termination.
We have implemented a static analyzer for proving conditional termination of programs written in (a subset of) C and, using experimental evidence, we show that it performs well on a wide variety of benchmarks, it is competitive with the state of the art and is able to analyze programs that are out of the reach of existing methods.

1 Introduction

Numerical abstract domains are widely used in static program analysis and verification to maintain information about the set of possible values of the program variables along with the possible numerical relationships between them. The most common abstract domains — intervals [10], octagons [27] and convex polyhedra [14] — maintain this information using convex sets consisting of conjunctions of linear constraints. The convexity of these abstract domains makes the analysis scalable. On the other hand, it might lead to too harsh approximations and imprecisions in the analysis, ultimately yielding false alarms and a failure of the analyzer to prove the desired program property.

The key for an adequate cost versus precision trade-off is the handling of disjunctions arising during the analysis (e.g., from program tests and loops). In practice, numerical abstract domains are usually refined by adding weak forms of disjunctions to increase the expressivity while minimizing the cost of the analysis [13, 18, 20, 29, etc.].

In this paper, we propose a novel parameterized abstract domain for the disjunctive refinement of numerical abstract domains which is particularly well-suited for proving conditional termination of imperative programs.

The elements of the abstract domain are inspired by the space partitioning trees [16] developed in the context of 3D computer graphics and the use of decision trees

* 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)

in program analysis and verification [3, 23]: they are decision trees where the decision nodes are labeled with linear constraints, and the leaf nodes belong to a numerical abstract domain. These decision trees recursively partition the space of possible values of the program variables inducing disjunctions into the numerical abstract domain.

The partitioning is *dynamic*: during the analysis, partitions (respectively, decision nodes and constraints) are split (respectively, added) by tests, modified by assignments and joined (respectively, removed) when merging control flows. In order to minimize the cost of the analysis, a widening limits the height of the decision trees and the number of maintained disjunctions.

We also emphasize that the partitioning is *semantic-based* rather than syntactic-based: the linear constraints labeling the decision nodes are automatically inferred by the analysis and do not necessarily appear in the program.

The abstract domain is parametric in the choice between the expressivity and the cost of the linear constraints for the decision nodes (e.g., polyhedral or octagonal constraints), and the choice of the numerical abstract domain for the leaf nodes. As a result of its adaptability, the abstract domain is well-suited to be used for the inference of different program properties, from program invariants to ranking functions.

```
int f (int x, int y, int r) {
    while 1( r > 0 ) {
        2r = r + x;
        3r = r − y;
    }4
    return 0;
}
```

Fig. 1: Simple C function. It terminates if $x < y$.

In the following, we describe an instance of this domain based on piecewise-defined ranking functions [30, 31] for the inference of sufficient preconditions for program termination.

Through this instance we propose an approach to termination analysis of imperative programs which is *modular*, i.e., which allows reasoning on a portion of the code (e.g., a function) at a time — without any knowledge about the complete program — and reusing the analysis result whenever the same function is called.

To illustrate the potential of our approach, let us consider the simple C function in Figure 1: at each loop iteration, the value of r is increased by the value of x and decreased by the value of y. Our abstract domain, parameterized by polyhedral constraints at the decision nodes and affine ranking functions at the leaf nodes and using a widening with thresholds, is able to automatically infer that the program terminates in at most r loop iterations (i.e., in at most $3r + 1$ program steps) if $x < y$ (the constraint $x < y$ not appearing in the program).

Our Contribution. In summary, in this paper we make several contributions. First, we propose a parameterized abstract domain for the disjunctive refinement of numerical abstract domains. We show its adaptability to different abstractions, focusing in particular on piecewise-defined ranking functions for proving program conditional termination. Second, we thoroughly discuss the widening operator for ranking functions, which is non trivial and of independent interest. Finally, we evaluate our approach for termination against state-of-the-art implementations [5, 19, 22].

2 Termination Semantics

We consider a programming language with non-deterministic statements. 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 the program transition relation $\tau \subseteq \Sigma \times \Sigma$ describes the possible transitions between states during program execution. 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 Floyd/Turing traditional method for proving program termination [15] consists in inferring ranking functions, namely mappings from program states to elements of a well-ordered set (e.g., $\langle \mathbb{O}, < \rangle$, the well-ordered set of ordinals) whose value decreases during program execution.

Intuitively, 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 [12], this intuition is formalized into a *most precise ranking function*[1] $w \in \Sigma \rightharpoonup \mathbb{O}$ that can be expressed as the least fixpoint of the operator ϕ starting from the totally undefined function $\dot{\emptyset}$:

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

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

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 $\widetilde{\mathsf{pre}}(X) \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma : \langle s, s' \rangle \in \tau \Rightarrow s' \in X\}$. The domain $\mathsf{dom}(w)$ of w is the set of states definitely leading to program termination: any trace starting in a state $s \in \mathsf{dom}(w)$ must terminate in at most $w(s)$ execution steps, while at least one trace starting in a state $s \notin \mathsf{dom}(w)$ does not terminate.

The most precise ranking function w is sound and complete to prove program termination (see [12]). However, it is usually not computable. In [30, 31], we present decidable abstractions of w by means of piecewise-defined ranking functions over natural numbers [30] and ordinals [31]. The abstractions refer to the following approximation order (see [11] for further discussion on approximation and computational orders of an abstract domain):

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

They compute an *over-approximation* of the value of the function w and an *under-approximation* of its domain of definition $\mathsf{dom}(w)$. In this way, an 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 terminating.

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

3 Piecewise-defined Ranking Functions

We derive new decidable abstractions of w by introducing the abstract domain of constraint-based decision trees $\mathbb{T}$ and combining it with the piecewise-defined ranking functions abstractions from [30, 31].

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a finite and totally ordered set of program variables with value in $\mathbb{Z}$. We split the program state space Σ into program control points $\mathcal{P}$ and environments $\mathcal{E} \triangleq \mathcal{X} \to \mathbb{Z}$, which map each program variable to its integer value at a given program control point. No approximation is made on $\mathcal{P}$. On the other hand, each program control point $p \in \mathcal{P}$ is associated with an element $t \in \mathcal{T}$ of the abstract domain $\mathbb{T}$. Specifically, t represents an abstraction of the function $v \in \mathcal{E} \rightharpoonup \mathbb{O}$ defined on the environments related to the program control point p:

$$\langle \mathcal{E} \rightharpoonup \mathbb{O}, \sqsubseteq \rangle \xleftarrow{\gamma_{\mathsf{T}}} \langle \mathcal{T}, \sqsubseteq_{\mathsf{T}} \rangle.$$

(we postpone the formal definition of γ_{T} to Section 3.2).

We assume we are given as parameter a (possibly infinite) set $\mathcal{L}$ of linear constraints over $\mathcal{X}$ (e.g., interval [10], octagonal [27] or polyhedral [14] constraints). We also assume we are given an abstraction of partial functions from environments to ordinals by means of a numerical abstract domain for functions $\mathbb{F} \triangleq \langle \mathcal{F}, \sqsubseteq_{\mathsf{F}} \rangle$ [30, 31], equipped with a bottom element $\bot_{\mathsf{F}}$ representing the totally undefined function $\dot{\emptyset}$.

The elements of the abstract domain $\mathbb{T}$ are disjunctive refinements of those of $\mathbb{F}$ (i.e., piecewise-defined functions) in the form of *constraint-based decision trees*, i.e., decision trees where the decision nodes are labeled by linear constraints in $\mathcal{L}$, and the leaf nodes belong to $\mathcal{F}$. As an example, in Figure 2, the constraint-based decision tree represents the piecewise-defined partial ranking function of the program in Figure 1:

$$f(x, y, r) = \begin{cases} 1 & r \leq 0 \\ 3r + 1 & r > 0 \wedge x < y \\ \text{undefined} & r > 0 \wedge x \geq y \end{cases}$$

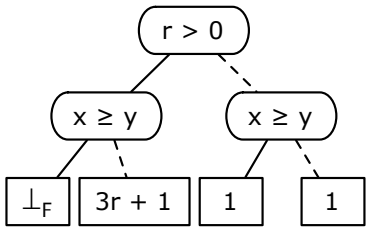


Fig. 2: Example of constraint-based decision tree abstracting a function. The leaves of the tree represent the value of the function for the satisfied constraints on the variables.

In the following, we first dive into some more details on the functions abstract domain. Then, we give a more formal presentation of our constraint-based decision trees and all abstract operators, including widening to ensure convergence.

3.1 Functions Abstract Domain

The functions abstract domain $\mathbb{F}$ abstracts a partial ranking function $v \in \mathcal{E} \rightharpoonup \mathbb{O}$ from environments to ordinals by an element $f \in \mathcal{F}$ which is a function of the program variables, or the element $\bot_{\mathsf{F}}$ representing potential non-termination, or the element $\top_{\mathsf{F}}$ representing the lack of enough information to conclude. In the following, the leaf nodes

belonging to $\mathcal{F} \setminus \{\bot_\mathsf{F}, \top_\mathsf{F}\}$ and $\{\bot_\mathsf{F}, \top_\mathsf{F}\}$ will be referred to as *defined* and *undefined* leaf nodes, respectively.

In order to under-approximate the domain of definition of the most precise ranking function, the concretization function γ_F maps all undefined leaf nodes to the totally undefined function $\dot{\emptyset}$:

$$\gamma_\mathsf{F}(\bot_\mathsf{F}) = \gamma_\mathsf{F}(\top_\mathsf{F}) = \dot{\emptyset}$$

In fact, the computational and approximation ordering of the abstract domain respectively do and do not distinguish between $\bot_\mathsf{F}$ and $\top_\mathsf{F}$. In particular, the element $\top_\mathsf{F}$ is produced and used only by the widening operator discussed in the upcoming Section 3.3.

In [30], we considered instances of the abstract domain $\mathbb{F}$ based on affine functions $f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$, where $m_1, \ldots, m_n, q$ are constants. In [31] we extended the abstraction to functions over ordinals.

Remark 1. In this paper, we are mainly focusing on instances of $\mathbb{T}$ for program termination. However, we emphasize that $\mathbb{T}$ is also well-suited to be instantiated with other numerical abstract domains. In fact, analogously to [20], we can have $\mathcal{F} \triangleq \{\mathbf{0}, \mathbf{1}\}$ and interpret the abstract domain $\mathbb{T}$ as the disjunctive refinement of numerical abstract domains such as intervals [10], octagons [27] and convex polyhedra [14].

We assume that the abstract domain $\mathbb{F}$ is equipped with sound binary operators for approximation ordering $\sqsubseteq_\mathsf{F}$, join $\sqcup_\mathsf{F}$ and widening $\triangledown_\mathsf{F}$, as well as sound transfer functions for assignments $\mathsf{ASSIGN}_\mathsf{F}$ and tests $\mathsf{FILTER}_\mathsf{F}$. We refer to [30, 31] for examples.

3.2 Constraint-based Decision Trees

The decision tree abstract domain $\mathbb{T}$ is parametric in the choice of the abstract domain $\mathbb{F}$ and the set of linear constraints $\mathcal{L} \subseteq \{k_1 x_1 + \cdots + k_n x_n \leq k_{n+1} \mid k_1, \ldots, k_n, k_{n+1} \in \mathbb{Z}\}$. As for boolean decision trees where an ordering is imposed on all decision variables, let $<_\mathsf{L} \in \mathcal{L} \times \mathcal{L}$ be a total order on $\mathcal{L}$. As an example, we can define $<_\mathsf{L}$ to be the lexicographic order on the coefficients $k_1, \ldots, k_n$ and constant k_{n+1} of the linear constraints.

An element t of the abstract domain $\mathbb{T}$ belongs to a set $\mathcal{T}$ and is either a leaf node $\mathsf{LEAF} : f$, with f an element of $\mathcal{F}$, or a decision node $\mathsf{NODE}\{c\} : t_1; t_2$, such that c is a linear constraint in $\mathcal{L}$ (in the following denoted by $t.c$) and the left subtree t_1 and the right subtree t_2 (in the following denoted by $t.l$ and $t.r$, respectively) belong to $\mathcal{T}$. In particular, given a decision tree $\mathsf{NODE}\{c\} : t_1; t_2$, the linear constraint c is always the smallest constraint appearing in the tree, and the left and right subtrees t_1 and t_2 are either both leaf nodes or both decision nodes labeled with the same linear constraint c' (such that $c <_\mathsf{L} c'$), i.e., two decision nodes at the same height in the decision tree are always labeled with the same linear constraint. In order to ensure a canonical representation, we also avoid a constraint c and its negation $\neg c$ simultaneously appearing in a constraint-based decision tree (e.g., by keeping only the largest constraint with respect to $<_\mathsf{L}$ between c and $\neg c$).

Remark 2. The choice of maintaining the same constraints at the same height in the decision trees is important for the design of the widening operator as explained in the following Section 3.3.

Algorithm 1 : Tree Unification

```
1: function UNIFICATION(t1,t2)
2:     if ISLEAF(t1) ∧ ISLEAF(t2) then
3:         return (t1, t2)
4:     else if ISLEAF(t1) ∨ (ISNODE(t1) ∧ ISNODE(t2) ∧ t2.c <L t1.c) then
5:         (l1, l2) ← UNIFICATION(t1, t2.l)
6:         (r1, r2) ← UNIFICATION(t1, t2.r)
7:         return (NODE{t2.c} : l1; r1, NODE{t2.c} : l2; r2)
8:     else if ISLEAF(t2) ∨ (ISNODE(t1) ∧ ISNODE(t2) ∧ t1.c <L t2.c) then
9:         (l1, l2) ← UNIFICATION(t1.l, t2)
10:        (r1, r2) ← UNIFICATION(t1.r, t2)
11:        return (NODE{t1.c} : l1; r1, NODE{t1.c} : l2; r2)
12:    else
13:        (l1, l2) ← UNIFICATION(t1.l, t2.l)
14:        (r2, r2) ← UNIFICATION(t1.r, t2.r)
15:        return (NODE{t1.c} : l1; r1, NODE{t2.c} : l2; r2)
```

A constraint-based decision tree $t \in \mathcal{T}$, recursively partitions the space of values of the program variables inducing disjunctions into the abstract domain $\mathbb{F}$. Moreover, since two decision nodes at the same height in the decision tree are always labeled with the same linear constraint, they induce the same partition on their left and right subtrees.

Concretization Function. The concretization function γ_{T} depends on the concretization function γ_{F} of the abstract domain $\mathbb{F}$ and produces a (partial) ranking function:

$$\gamma_{\mathsf{T}}(\mathsf{LEAF} : f) = \gamma_{\mathsf{F}}(f)$$
$$\gamma_{\mathsf{T}}(\mathsf{NODE}\{c\} : t_1; t_2) = \gamma_{\mathsf{T}}(t_1)|_c \;\dot\cup\; \gamma_{\mathsf{T}}(t_2)|_{\neg c}$$

where $v|_c$ is the partial ranking function $v \in \mathcal{E} \rightharpoonup \mathbb{O}$ whose domain $\mathsf{dom}(v)$ is restricted to the environments satisfying the constraint c and $\dot\cup$ joins partial functions with disjoint domains: $(f_1 \dot\cup f_2)(x) \triangleq f_1(x)$, if $x \in \mathsf{dom}(f_1)$, and $(f_1 \dot\cup f_2)(x) \triangleq f_2(x)$, if $x \in \mathsf{dom}(f_2)$, where $\mathsf{dom}(f_1) \cap \mathsf{dom}(f_2) = \emptyset$.

Ordering, Join. The binary operators for the approximation ordering $\sqsubseteq_{\mathsf{T}}$ and join $\sqcup_{\mathsf{T}}$ of constraint-based decision trees rely on Algorithm 1 for tree unification. Given two decision trees $t_1 \in \mathcal{T}$ and $t_2 \in \mathcal{T}$, the tree unification algorithm finds a common refinement for the trees, possibly adding decision nodes (cf. Lines 5-7 and Lines 9-11). Note that the tree unification does not loose any information. Then, the binary operations are carried out "leaf-wise" on the unified constraint-based decision trees.

Ordering. Given two unified constraint-based decision trees, their approximation ordering is decided by the approximation ordering $\sqsubseteq_{\mathsf{F}}$ of the abstract domain $\mathbb{F}$:

$$(\mathsf{LEAF} : f_1) \sqsubseteq_{\mathsf{T}} (\mathsf{LEAF} : f_2) = f_1 \sqsubseteq_{\mathsf{F}} f_2$$
$$(\mathsf{NODE}\{c\} : l_1; r_1) \sqsubseteq_{\mathsf{T}} (\mathsf{NODE}\{c\} : l_2; r_2) = (l_1 \sqsubseteq_{\mathsf{T}} l_2) \wedge (r_1 \sqsubseteq_{\mathsf{T}} r_2)$$

Algorithm 2 : Tree Augment

```
1: function AUGMENT(t,C)
2:     if ISEMPTY(C) then return t
3:     else
4:         c ← min_{<L} C                    /* c is the smallest constraint appearing in C */
5:         return (NODE{c} : AUGMENT(t, C \ {c}); AUGMENT(t, C \ {c}))
```

Algorithm 3 : Tree Assign

```
1: function ASSIGN(t,x := a)
2:     if ISLEAF(t) then return LEAF : ASSIGN_F(f, x := a)        /* t ≜ LEAF : f */
3:     else
4:         C ← ASSIGN_L(t.c, x := a)
5:         if ISEMPTY(C) then return ASSIGN(t.l, x := a) ⊔_T ASSIGN(t.r, x := a)
6:         else if ISUNSAT(C) then return ASSIGN(t.r, x := a)
7:         else
8:             l ← AUGMENT(ASSIGN(t.l, x := a), C)
9:             r ← AUGMENT(ASSIGN(t.r, x := a), C)
10:            return NODE{l.c} : l; r
```

Join. Similarly, given two unified constraint-based decision trees, their join is built using the join operator $\sqcup_{\mathsf{F}}$ of the abstract domain $\mathbb{F}$:

$$(\mathsf{LEAF} : f_1) \sqcup_{\mathsf{T}} (\mathsf{LEAF} : f_2) = \mathsf{LEAF} : (f_1 \sqcup_{\mathsf{F}} f_2)$$
$$(\mathsf{NODE}\{c\} : l_1; r_1) \sqcup_{\mathsf{T}} (\mathsf{NODE}\{c\} : l_2; r_2) = \mathsf{NODE}\{c\} : (l_1 \sqcup_{\mathsf{T}} l_2); (r_1 \sqcup_{\mathsf{T}} r_2)$$

Assignments, Tests. The transfer functions for assignments $\mathsf{ASSIGN}_{\mathsf{T}}$ and tests $\mathsf{FILTER}_{\mathsf{T}}$ add, modify or delete decision nodes of a constraint-based decision tree. In particular, both operators rely on Algorithm 2 for the extension of a constraint-based decision tree $t \in \mathcal{T}$ with decision nodes built from linear constraints in $C \subseteq \mathcal{L}$.

Assignments. We recall that the most precise ranking function w defined in Section 2 is a *backward semantics*. Consequently, we consider *backward assignments* to a constraint-based decision tree. The transfer function $\mathsf{ASSIGN}_{\mathsf{T}}$ is described by Algorithm 3. An assignment $x := a$ to a tree $t \in \mathcal{T}$ is carried out independently on each constraint $c \in \mathcal{L}$ appearing in t (cf. Line 4): given a constraint c, the primitive $\mathsf{ASSIGN}_{\mathsf{L}}$ substitutes the expression a for the variable x within the constraint c. Since the modified constraint may not be representable exactly in $\mathcal{L}$, $\mathsf{ASSIGN}_{\mathsf{L}}$ produces a set of constraints $C \subseteq \mathcal{L}$ approximating it. For instance, non-linear assignments can be modeled using standard linearization techniques [3]. In case C is empty, it means that the constraint c does not exist anymore and the subtrees of t should be joined (cf. Line 5). In case C is an unsatisfiable set of constraints, it means that c is no longer satisfiable and we should keep only the right subtree of t (cf. Line 6). Otherwise, C is a set of constraints that should be substituted to c in t (cf. Lines 8-10). Finally, an assignment to a leaf node is carried out by the operator $\mathsf{ASSIGN}_{\mathsf{F}}$ of the abstract domain $\mathbb{F}$ (cf. Line 2).

Algorithm 4 : Tree Filter

1: **function** FILTER-AUX(t,c)
2: **if** ISLEAF(t) **then return** LEAF : $\mathsf{FILTER}_{\mathsf{F}}(f, c)$ $/* \; t \triangleq \mathsf{LEAF} : f \; */$
3: **else return** $\mathsf{NODE}\{t.c\}$: FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

4: **function** FILTER(t,c)
5: $C \leftarrow \mathsf{FILTER}_{\mathsf{L}}(c)$
6: **return** AUGMENT(FILTER-AUX(t, c), C)

Remark 3. Note that Algorithm 3 is general enough to also handle forward assignments, in case the abstract domain $\mathbb{T}$ is instantiated with other numerical abstract domains as mentioned in Remark 1. In fact, it is sufficient to modify the primitive $\mathsf{ASSIGN}_{\mathsf{L}}$ accordingly in order to handle forward assignments.

Example 1. Let us consider the constraint-based decision tree $\mathsf{NODE}\{x - y \leq 0\}$: $(\mathsf{NODE}\{y \leq 0\} : \alpha; \beta); (\mathsf{NODE}\{y \leq 0\} : \gamma; \delta)$, where greek letters denote leaf nodes. The forward non-invertible assignment $y = 3$, modifies the constraint $x - y \leq 0$ to $x \leq 3$ and removes the constraint $y \leq 0$ which is no longer satisfiable: $\mathsf{NODE}\{x \leq 3\} : \beta; \delta$. Instead, the backward non-deterministic assignment $y = \; ?$ removes y from any constraint appearing in the tree, enforcing the join of the leaf nodes α and β and the leaf nodes γ and δ: $\mathsf{NODE}\{x \leq 0\} : (\alpha \sqcup_{\mathsf{T}} \beta); (\gamma \sqcup_{\mathsf{T}} \delta)$. □

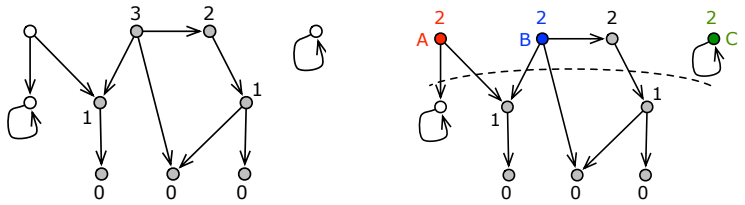
Tests. The transfer function $\mathsf{FILTER}_{\mathsf{T}}$ for test statements is described by Algorithm 4. First, a test statement c is handled independently on each leaf node (cf. Line 2). The primitive $\mathsf{FILTER}_{\mathsf{L}}$ approximates c producing a set of constraints $C \subseteq \mathcal{L}$ (cf. Line 5). Then, the constraint-based decision tree $t \in \mathcal{T}$ is augmented with C (cf. Line 6).

Note that, following an assignment or a test, the decision trees must be *sorted* and *normalized* in order to remove possible multiple occurrences of a constraint c and possible occurrences of both a constraint c and its negation $\neg c$ (e.g., by keeping only the largest constraint with respect to $<_{\mathsf{L}}$ between c and $\neg c$): for example, $\mathsf{NODE}\{y \leq 1\} : (\mathsf{NODE}\{y \leq 0\} : \alpha; \beta); (\mathsf{NODE}\{y \leq 0\} : \gamma; \delta)$ is sorted as $\mathsf{NODE}\{y \leq 0\} : (\mathsf{NODE}\{y \leq 1\} : \alpha; \gamma); (\mathsf{NODE}\{y \leq 1\} : \beta; \delta)$ and $\mathsf{NODE}\{-y \leq -1\} : (\mathsf{NODE}\{y \leq 0\} : \alpha; \beta); (\mathsf{NODE}\{y \leq 0\} : \gamma; \delta)$ is normalized as $\mathsf{NODE}\{y \leq 0\} : \gamma; \beta$.

The soundness of all the abstract operators of $\mathbb{T}$ follows immediately from the soundness of the corresponding abstract operators of $\mathbb{F}$.

3.3 Widening

The widening operator ∇_{T} requires a more thorough discussion. The widening is allowed more freedom than the other operators, in the sense that it is *temporary* allowed to *under-approximate* the value of the most precise ranking function or *over-approximate* its domain of definition, or both — in contrast with the approximation order $\sqsubseteq$. This is necessary in order to extrapolate a ranking function over the program states on which



(a) Most precise ranking function. (b) Unsound abstraction.

Fig. 3: Example of unsound abstraction (b) of a most precise ranking function (a).

it is not yet defined. This is possible because the only requirement of a static analysis is that, when the iteration sequence with widening is stable for the computational order, its limit is a sound abstraction of the program semantics of interest with respect to the approximation order. For this reason, the widening ∇_{T} consists of many steps that need to be performed in order to guarantee the soundness of the analysis with respect to the most precise ranking function w. In the following, we will go through these steps and we will discuss them in some detail.

As running example, let us consider Figure 3. In Figure 3a we depict a transition system and the value of the most precise ranking function for the well-founded part of the transition relation. In Figure 3b we represent the concretization of a possible abstract analysis iterate. In this case the abstraction both under-approximates the value of the most precise ranking function (on the second state from the left — case B) and over-approximates its domain of definition (including the first and the last state from the left — case A and C, respectively). In case A, the loop causing non-termination is *outside* the domain of definition of the (unsound) abstract function, while in case C the loop is *inside*.

Step 1: Check for Case A. The first step that the widening operator ∇_{T} has to do is to check for cases like case A, where the domain of definition of the most precise ranking function has been over-approximated including a program state from which a non-terminating loop is reachable. In cases like case A, at the next iteration due to the soundness of all the other abstract operators (cf. Section 3.2) the value of the abstract function will become $\bot_{\mathsf{F}}$. In order to handle such situations, the widening ∇_{T} has to look for leaf nodes whose value is now $\bot_{\mathsf{F}}$ and that previously belonged to a defined subtree (i.e., a subtree with only defined leaf nodes). Then, it has to substitute their value with $\top_{\mathsf{F}}$ in order to prevent successive iterates from mistakenly including again the same program states into the abstract function.

Step 2: Domain Widening - Tree Left Unification. At this point, the widening operator ∇_{T} calls Algorithm 5 for tree unification. Algorithm 5 is a slight modification of Algorithm 1: given two constraint-based decision trees[2] the left unification algorithm enforces the refinement of the first tree on the second, possibly removing decision nodes (by joining

[2] Algorithm 5 requires the constraints appearing in the first tree to be a subset of those appearing in the second, which can always be ensured by computing $t_1 \nabla_{\mathsf{T}} (t_1 \sqcup_{\mathsf{T}} t_2)$ instead of $t_1 \nabla_{\mathsf{T}} t_2$.

Algorithm 5 : Tree Left Unification

```
1: function LEFT-UNIFICATION(t1,t2)
2:     if ISLEAF(t1) ∧ ISLEAF(t2) then
3:         return (t1, t2)
4:     else
5:         if ISLEAF(t1) ∨ (ISNODE(t1) ∧ ISNODE(t2) ∧ t2.c <L t1.c) then
6:             return LEFT-UNIFICATION(t1, t2.l ⊔T t2.r)
7:         else
8:             (l1, l2) ← LEFT-UNIFICATION(t1.l, t2.l)
9:             (r2, r2) ← LEFT-UNIFICATION(t1.r, t2.r)
10:            return (NODE{t1.c} : l1; r1, NODE{t2.c} : l2; r2)
```

subtrees, cf. Line 6) and thus *extrapolating the domain* of the abstract ranking function over program states on which it is not yet defined. In this way we might loose information but we ensure convergence limiting the size of the constraint-based decision trees.

Note that it is important to check for cases like case A before the tree left unification. Otherwise, since leaf nodes whose value is $\bot_{\mathsf{F}}$ might disappear when joining subtrees, we would not be able to detect them.

Step 3: Check for Case B and C. The third step that the widening operator $\triangledown_{\mathsf{T}}$ has to do is to check for cases like case B, where the value of the most precise ranking function has been under-approximated, and cases like case C, where its domain of definition has been over-approximated including a non-terminating loop. In cases like B and C, at the next iteration the value of the abstract function will increase. In order to handle such situations, the widening $\triangledown_{\mathsf{T}}$ has to look for leaf nodes whose value has increased between two iterates and it has to substitute their value with $\top_{\mathsf{F}}$ in order to prevent an indefinite growth. Note that the widening is not able to distinguish between an under-approximation of the value of the most precise ranking function (as in case B) and an over-approximation of its domain of definition as in case C.

The following lemma establishes that the widening operator $\triangledown_{\mathsf{T}}$ always recovers from the inclusion of non-terminating program states into the domain of an abstract ranking function at some iterate X_i (i.e., it always recovers from an over-approximation of the domain $\mathsf{dom}(w)$ of the most precise ranking function w — cases A and C):

Lemma 1. $\mathsf{dom}(\gamma_{\mathsf{T}}(X_i)) \not\subseteq \mathsf{dom}(w) \Rightarrow \mathsf{dom}(\gamma_{\mathsf{T}}(X_{i+1})) \subset \mathsf{dom}(\gamma_{\mathsf{T}}(X_i))$

It follows that the domain of the limit w_{T} of the iteration sequence with widening is a sound *under-approximation* of the domain of the most precise ranking function w:

Corollary 1. $\mathsf{dom}(\gamma_{\mathsf{T}}(w_{\mathsf{T}})) \subseteq \mathsf{dom}(w)$

In addition, the next lemma establishes that, if at some iterate X_i the value of the most precise ranking function w is under-approximated (case B), the iteration sequence with widening $\triangledown_{\mathsf{T}}$ is not stable:

Lemma 2. $\exists s \in \mathsf{dom}(\gamma_{\mathsf{T}}(X_i)) \cap \mathsf{dom}(w) : w(s) > \gamma_{\mathsf{T}}(X_i)(s) \Rightarrow s \notin \mathsf{dom}(\gamma_{\mathsf{T}}(X_{i+1}))$

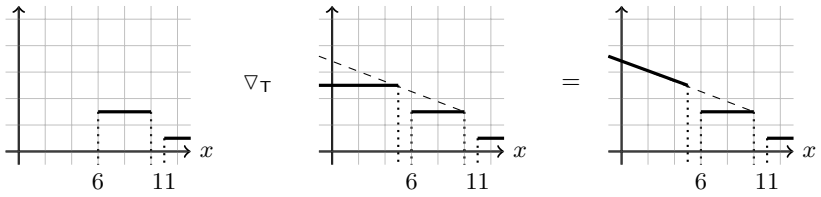


Fig. 4: Example of *Value Widening*.

It follows that the value of the limit w_{T} of the iteration sequence with widening is a sound *over-approximation* of the value of the most precise ranking function w:

Corollary 2. $\forall s \in \mathsf{dom}(w) \cap \mathsf{dom}(\gamma_{\mathsf{T}}(w_{\mathsf{T}})) : w(s) \leq \gamma_{\mathsf{T}}(w_{\mathsf{T}})(s)$

Step 4: Value Widening. Once the widening operator ∇_{T} has checked for possible violations of the soundness and the domain of the abstract ranking function has been extrapolated, the last step is devoted to *extrapolating the value* of the ranking function over the program states on which it was not yet defined. The heuristic that we used in [30] has proved to be rather effective and justifies our choice to maintain the same linear constraint at the same height in the decision trees. We decided to widen the leaf nodes defined only in the second tree with respect to their *adjacent* leaf nodes. The rationale being that programs often loop over consecutive values of a variable, we use the information available in adjacent partitions of the domain of the ranking function to infer the shape of the ranking function for the current partitions, i.e., the leaf nodes defined only in the second tree (cf. Figure 4). Since we maintain the same linear constraint at the same height in the decisions tree, the adjacency between leaf nodes is pretty straightforward to define: two leaf nodes in a constraint-based decision tree are adjacent if their paths from the root differ for exactly one constraint satisfaction.

Remark 4. In establishing relationships only between adjacent leaf nodes, we are considering a rather naïve heuristic. Another possibility would be establishing relationships between leaf nodes based on the parity of some variable, or based on numerical relationships between variables. It is also possible to improve the widening by introducing thresholds in the left unification (in order to limit the loss of precision). We plan to investigate these possibilities as part of our future work.

Example 2. Let $\mathcal{F}$ be the set of *affine functions* of the program variables (plus $\bot_{\mathsf{F}}$ and $\top_{\mathsf{F}}$) [30]. We consider the widening between[3] $t_1 \triangleq \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); t'_1$ and $t_2 \triangleq \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); t'_2$ where the decision (sub)trees t'_1 and t'_2 are:

$$t'_1 \triangleq \mathsf{NODE}\{x - y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 3)$$
$$t'_2 \triangleq \mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{LEAF} : 5); (\mathsf{LEAF} : \bot_{\mathsf{F}})); (\mathsf{LEAF} : 3)$$

[3] Redundant constraints in the decision trees are omitted for conciseness.

First, the left unification modifies t_2' into: $\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{LEAF} : 3); (\mathsf{LEAF} : 5)$. Then, the leaf node $\mathsf{LEAF} : 5$, defined only in t_2', is widened with respect to its adjacent leaf node $\mathsf{LEAF} : 3$. This produces the leaf node $\mathsf{LEAF} : 2x + 1$. □

Since Algorithm 5 limits the height of constraint-based decision trees (cf. *Step* 2) and we prevent the indefinite growth of the value of the functions inside the leaf nodes (cf. *Step* 3), the iteration sequence with widening is eventually stable after finitely many steps. Its limit w_{T} is a sound abstraction of the most precise ranking function w:

Theorem 1. $w \sqsubseteq \gamma_{\mathsf{T}}(w_{\mathsf{T}})$.

Proof. Follows by definition of $\sqsubseteq$ from Corollary 1 and Corollary 2. □

Remark 5. The reason for the complexity of the widening operator $\triangledown_{\mathsf{T}}$ is the coexistence of an approximation and a computational order in the termination semantics domain (cf. Section 2) as well as in the abstract domain. We believe that ours is the first widening in the two-order settings. In case the abstract domain $\mathbb{T}$ is instantiated with other numerical abstract domains as mentioned in Remark 1, the widening $\triangledown_{\mathsf{T}}$ becomes straightforward (only the tree left unification and "leaf-wise" widening $\triangledown_{\mathsf{F}}$ being needed).

3.4 Abstract Termination Semantics

The operators of the abstract domain are combined together to compute an abstraction of the most precise 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 and the soundness (and termination) of the iteration sequence with widening, 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.

Example 3. Let us consider the following simple C program:

$$\mathsf{while}\ ^1(\ x > 0\ \wedge\ y > 0\ )\ \{\ ^2x = x - y;\ \}^3$$

At each loop iteration, the value of x is decreased until it becomes less than or equal to zero. The program always terminates whatever the initial values for x and y are.

We analyze this program using our abstract domain of constraint-based decision trees, parameterized with polyhedral [14] constraints at the decision nodes and affine functions [30] at the leaf nodes. The starting point is $t_3 = \mathsf{LEAF} : 0$ at the final control point 3. We use a widening delay of two iterations. At the first iteration, at the program control point 1 we obtain the decision tree $t_1^1 = \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : \bot_{\mathsf{F}}))$ which, taking into account the assignment $x = x - y$, becomes $t_2^1 = \mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 2); (\mathsf{LEAF} : \bot_{\mathsf{F}})); (\mathsf{LEAF} : 2)$ at the program control point 2. At the third iteration the widening comes into action between the decision trees of Example 2 yielding a fixpoint: $t_1^3 = \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : 2x + 1)); (\mathsf{LEAF} : 3))$ (i.e., the ranking function $f_1(x, y) = 2x + 1$) which proves that the program terminates in at most $2x + 1$ program steps, whatever the initial values for x and y are. □

	Tot	FuncTion-OCT	FuncTion-POLY	FuncTion [31]	Time
FuncTion-OCT	39	–	0	18	4s
FuncTion-POLY	46	7	–	24	11s
FuncTion [31]	27	6	5	–	13s

Fig. 5: Overview of the experimental evaluation for FuncTion.

4 Implementation

We have implemented our abstract domain of constraint-based decision trees $\mathbb{T}$ into our prototype static analyzer FuncTion[4] based on piecewise-defined ranking functions. A preliminary version of FuncTion [31] (without relational partitioning) participated in the *3rd International Competition on Software Verification (SV-COMP 2014)*, which featured a category for termination of C programs for the first time.

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 handling linear constraints in decision trees are based on intervals [10], octagons [27] and convex polyhedra [14], and the available abstractions for ranking functions are based on affine functions. The operators from the intervals, octagons and convex polyhedra abstract domains are provided by the APRON library [24]. It is also possible to activate the extension to ordinal-valued ranking functions [31] and 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.

Experiments. We have evaluated our prototype implementation against a set of 87 terminating C programs collected from the *SV-COMP 2014* termination category and from various publications in the area [1, 6, 9, etc.]. All the experiments were performed on a 1.30GHz Core i5 system with 4GB of RAM and running Ubuntu 12.04.

In Figure 5, we compared FuncTion to its preliminary version [31]. In particular, we evaluated the expressiveness and efficiency of two instances of our abstract domain of constraint-based decision trees: FuncTion-OCT (which uses octagonal constraints for labeling the decision nodes) and FuncTion-POLY (which uses polyhedral constraints). In the first column we report the total number of programs that each tool was able to prove termination for. In the second to the fourth column, we consider each tool and we report the number of programs that every other tool was able to prove terminating among the programs that the tool was not able to prove termination for. Finally, the last column reports the average running time in seconds for the programs where the tool proved termination. The results match the expectations: FuncTion-OCT is faster than FuncTion-POLY but less precise in seven examples; also, both FuncTion-OCT and FuncTion-POLY are more precise than FuncTion in its preliminary version. Note that, to improve precision, FuncTion [31] avoids trying to infer a ranking function for the non-reachable states

[4] `http://www.di.ens.fr/~urban/FuncTion.html`

	Tot	FuncTion	AProVE [19]	T2 [5]	Ultimate [22]	Time	Timeouts
FuncTion	51	–	8	8	3	6s	5
AProVE [19]	60	17	–	7	2	35s	19
T2 [5]	73	30	20	–	3	2s	0
Ultimate [22]	79	31	21	9	–	9s	1

Fig. 6: Overview of the experimental evaluation for termination.

while FuncTion-OCT and FuncTion-POLY do not apply yet this kind of optimizations: for this reason, FuncTion [31] was able to prove termination for respectively six and five programs that FuncTion-OCT and FuncTion-POLY were not able to prove terminating.

We also compared FuncTion (using all the available abstractions) to some of the other tools that participated to the termination category of *SV-COMP 2014*: AProVE [19], T2 [5], and Ultimate Büchi Automizer [22]. Figure 6 shows an overview of the experimental evaluation when using a time limit of 300 seconds for each example. In the first column we report the total number of programs that each tool was able to prove termination for. In the second to the fifth column, similarly to Figure 5, we consider each tool and we report the number of programs that every other tool was able to prove terminating among the programs that the tool was not able to prove termination for. Finally, the last columns report the average running time in seconds for the programs where the tool proved termination and the number of time outs (i.e., programs for which the analysis took more than 300 seconds). We observe that FuncTion proved termination of 51 of the 87 programs considered, while the other tools get better results. We noticed that the main reason for this is the value widening heuristic (cf. Section 3.3) used by FuncTion. We plan to study these issues further and improve the widening operator as part of our future work. However, we also observe that FuncTion was able to prove termination for eight programs that AProVE and T2 were not able to prove terminating, and for three programs that Ultimate Büchi Automizer was not able to prove termination for. We noticed that all these programs are characterized by the presence of multiple paths with unrelated or conflicting rankings inside loops: these programs are handled in a natural way by the inherent partitioning at the basis of our tool while the other tools must often resort to heuristics or specific workarounds [9].

5 Related Work

The use of (binary) decision trees (Binary Decision Diagrams, in particular) for verification has been devoted a large body of work, especially in the area of timed-systems and hybrid-system verification [23]. In this paper, we focus on common program analysis applications and, in this sense, our abstract domain is mostly related to the ones presented in [13, 20]: both ours and these abstract domains are a disjunctive refinement of an abstract domain based on decision trees extended with linear constraints. However, the abstract domain proposed in [20] is designed specifically for the disjunctive refinement of the intervals abstract domain [10], while our abstract domain is parameterized by the (possibly relational) abstract domain we want to build the disjunctive refinement for.

Moreover, while our abstract domain is based on binary decision trees where we impose the same linear constraint at the same tree level, in [13] the choices at the decision nodes may differ at each node and their number is not bounded a priori.

In general, despite all the available alternatives [3, 13, 18, 20, 21, 29, etc.], it seems to us that in the literature there is no disjunctive abstract domain well-suited for program termination. A first (minor) reason is the fact that most of the existing disjunctive abstract domains are designed specifically for forward analyses while ranking functions are inferred through backward analysis (cf. Section 3.4). However, the main reason is that adapting existing widening operators to ranking functions is not obvious due to the coexistence of an approximation and computational ordering in the termination semantics domain (cf. Section 2 and Section 3.3).

As for related work on termination, we emphasize that our method is able to *directly* manipulate ranking functions which are dealt with as any other kind of invariants associated to program control points. In this sense, it differs from the majority of the literature based on the *indirect* use of invariants for synthesizing ranking functions or just proving termination [1, 2, 5, 7, 25, etc.]. Moreover, our approach is at the same time *modular* (i.e., able to reason on a portion of the code without any knowledge of the complete program) and able to deal with arbitrary control structures (i.e., it is not limited to simple loops as [28] or to non nested loops as [4]).

Finally, in the literature, we found only few works that have addressed the problem of automatically finding preconditions for program termination. In [8], the authors proposed a method based on preconditions generating ranking functions from potential rankings (i.e., mappings to elements of a well-ordered set whose value does not necessarily decrease during program execution), while our preconditions are inherently obtained from the inferred ranking functions as the set of programs state for which the ranking function is defined. Thus, our preconditions are derived by *under-approximation* of the set of terminating states as opposed to the approaches presented in [17, 26] where the preconditions are derived by (complementing an) over-approximation of the non-terminating states.

6 Conclusion and Future Work

In this paper, we proposed a novel parameterized abstract domain for the disjunctive refinement of numerical abstract domains. We have shown its adaptability to different abstractions, focusing in particular on piecewise-defined ranking functions for automatically proving conditional termination. Our approach to program termination is semantic-based and approximate in a provably sound way. It is able to analyze programs that are out of the reach of state-of-the-art methods.

It remains for future work to improve the widening between ranking functions establishing cleverer relationships between leaf nodes of decision trees and introducing thresholds in order to limit the loss of precision. We also plan to design more abstract domains in order to support *non-linear* ranking functions (e.g., quadratic, cubic, exponential, ...).

Acknowledgements. We are grateful to the developers of AProVE [19], T2 [5], and Ultimate Büchi Automizer [22] for their help with the experiments.

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. J. Berdine, A. Chawdhary, B. Cook, D. Distefano, and P. W. O'Hearn. Variance Analyses from Invariance Analyses. In *POPL*, pages 211–224, 2007.
3. J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static Analysis and Verification of Aerospace Software by Abstract Interpretation. In *AIAA*, 2010.
4. A. R. Bradley, Z. Manna, and H. B. Sipma. Linear Ranking with Reachability. In *CAV*, pages 491–504, 2005.
5. M. Brockschmidt, B. Cook, and C. Fuhs. Better Termination Proving through Cooperation. In *CAV*, pages 413–429, 2013.
6. H. Y. Chen, S. Flur, and S. Mukhopadhyay. Termination Proofs for Linear Simple Loops. In *SAS*, pages 422–438, 2012.
7. M. Colón and H. Sipma. Practical Methods for Proving Program Termination. In *CAV*, pages 442–454, 2002.
8. B. Cook, S. Gulwani, T. Lev-Ami, A. Rybalchenko, and M. Sagiv. Proving Conditional Termination. In *CAV*, pages 328–340, 2008.
9. B. Cook, A. See, and F. Zuleger. Ramsey vs. Lexicographic Termination Proving. In *TACAS*, pages 47–61, 2013.
10. P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Programs. In *Symposium on Programming*, pages 106–130, 1976.
11. 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.
12. P. Cousot and R. Cousot. An Abstract Interpretation Framework for Termination. In *POPL*, pages 245–258, 2012.
13. P. Cousot, R. Cousot, and L. Mauborgne. A Scalable Segmented Decision Tree Abstract Domain. In *Essays in Memory of Amir Pnueli*, pages 72–95, 2010.
14. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*, pages 84–96, 1978.
15. R. W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.
16. H. Fuchs, Z. M. Kedem, and B. F. Naylor. On Visible Surface Generation by a Priori Tree Structures. *SIGGRAPH Computer Graphics*, 14(3):124–133, 1980.
17. P. Ganty and S. Genaim. Proving Termination Starting from the End. In *CAV*, pages 397–412, 2013.
18. R. Giacobazzi and F. Ranzato. Optimal Domains for Disjunctive Abstract Intepretation. *Sci. Comput. Program.*, 32(1-3):177–210, 1998.
19. J. Giesl, P. Schneider-Kamp, and R. Thiemann. Automatic Termination Proofs in the Dependency Pair Framework. In *IJCAR*, pages 281–286, 2006.
20. A. Gurfinkel and S. Chaki. BOXES: A Symbolic Abstract Domain of Boxes. In *SAS*, pages 287–303, 2010.
21. A. Gurfinkel and S. Chaki. Combining Predicate and Numeric Abstraction for Software Model Checking. *STTT*, 12(6):409–427, 2010.
22. M. Heizmann, J. Hoenicke, J. Leike, and A. Podelski. Linear Ranking for Linear Lasso Programs. In *ATVA*, pages 365–380, 2013.
23. B. Jeannet. Representing and Approximating Transfer Functions in Abstract Interpretation of Hetereogeneous Datatypes. In *SAS*, pages 52–68, 2002.

24. B. Jeannet and A. Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *CAV*, pages 661–667, 2009.
25. D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Proving Termination of Imperative Programs using Max-SMT. In *FMCAD*, pages 218–225, 2013.
26. D. Massé. Policy Iteration-based Conditional Termination and Ranking Functions. In *VMCAI*, pages 453–471, 2014.
27. A. Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
28. A. Podelski and A. Rybalchenko. A Complete Method for the Synthesis of Linear Ranking Functions. In *VMCAI*, pages 239–251, 2004.
29. S. Sankaranarayanan, F. Ivancic, I. Shlyakhter, and A. Gupta. Static Analysis in Disjunctive Numerical Domains. In *SAS*, pages 3–17, 2006.
30. C. Urban. The Abstract Domain of Segmented Ranking Functions. In *SAS*, pages 43–62, 2013.
31. C. Urban and A. Miné. An Abstract Domain to Infer Ordinal-Valued Ranking Functions. In *ESOP*, pages 412–431, 2014.

A Soundness of the Iteration Sequence with Widening

Let $\phi_{\mathsf{T}} \in \mathcal{T} \to \mathcal{T}$ be the abstract counterpart of $\phi \in (\Sigma \rightharpoonup \mathbb{O}) \to (\Sigma \rightharpoonup \mathbb{O})$ (that uses the sound abstract operators of $\mathbb{T}$ for joins $\sqcup_{\mathsf{T}}$, assignments $\mathsf{ASSIGN}_{\mathsf{T}}$ and tests $\mathsf{FILTER}_{\mathsf{T}}$) and let $\preccurlyeq_{\mathsf{T}}$ be the abstract counterpart of $\preccurlyeq$. The limit w_{T} of the iteration sequence with widening:

$$X_0 = \bot_{\mathsf{T}}$$
$$X_{i+1} = \begin{cases} X_i & \text{if } \phi_{\mathsf{T}}(X_i) \preccurlyeq_{\mathsf{T}} X_i \wedge \phi_{\mathsf{T}}(X_i) \sqsubseteq_{\mathsf{T}} X_i \\ X_i \triangledown_{\mathsf{T}} \phi_{\mathsf{T}}(X_i) & \text{otherwise} \end{cases}$$

is a sound over-approximation of the most precise ranking function w:

Theorem 1. $w \sqsubseteq \gamma_{\mathsf{T}}(w_{\mathsf{T}})$

In order to prove Theorem 1, we need some preliminary results.

Lemma 1. $\mathsf{dom}(\gamma_{\mathsf{T}}(X_i)) \not\subseteq \mathsf{dom}(w) \Rightarrow \mathsf{dom}(\gamma_{\mathsf{T}}(X_{i+1})) \subset \mathsf{dom}(\gamma_{\mathsf{T}}(X_i))$

Lemma 1 establishes that the widening operator $\triangledown_{\mathsf{T}}$ always recovers from the inclusion of non-terminating program states into the domain of an abstract ranking function at some iterate X_i (i.e., it always recovers from an over-approximation of the domain $\mathsf{dom}(w)$ of the most precise ranking function w — cases A and C in Section 3.3).

Proof (of Lemma 1). In case the iteration sequence with widening reaches an iterate X_i such that $\mathsf{dom}(\gamma_{\mathsf{T}}(X_i)) \not\subseteq \mathsf{dom}(w)$ it means that there exists a program state $s \in \mathsf{dom}(\gamma_{\mathsf{T}}(X_i))$ belonging to a non-terminating program trace σ (i.e., such that some states are repeated infinitely many times in σ). Without loss of generality we can assume that there is a single program state s' repeated infinitely many times in σ and that s' is the immediate successor of s in σ. Thus, either $s' \in \mathsf{dom}(\gamma_{\mathsf{T}}(X_i))$ or $s' \notin \mathsf{dom}(\gamma_{\mathsf{T}}(X_i))$. In case $s' \notin \mathsf{dom}(\gamma_{\mathsf{T}}(X_i))$, by the definition of ϕ, we have $s \notin \mathsf{dom}(\phi(\gamma_{\mathsf{T}}(X_i)))$ and, by

the soundness of all abstract operators, we have $\mathsf{dom}(\gamma_\mathsf{T}(\phi_\mathsf{T}(X_i))) \subseteq \mathsf{dom}(\phi(\gamma_\mathsf{T}(X_i)))$ which implies $s \notin \mathsf{dom}(\gamma_\mathsf{T}(\phi_\mathsf{T}(X_i)))$. Therefore, by definition of $\triangledown_\mathsf{T}$, we conclude that $\mathsf{dom}(\gamma_\mathsf{T}(X_{i+1})) \subset \mathsf{dom}(\gamma_\mathsf{T}(X_i))$. In case $s' \in \mathsf{dom}(\gamma_\mathsf{T}(X_i))$, without loss of generality we can assume that s and s' coincide. Thus, by the definition of ϕ and the soundness of all abstract operators we have $\gamma_\mathsf{T}(X_i)(s) < \gamma_\mathsf{T}(X_i)(s) + 1 = \phi(\gamma_\mathsf{T}(X_i))(s) \leq \gamma_\mathsf{T}(\phi_\mathsf{T}(X_i))(s)$. Therefore, by the definition of $\triangledown_\mathsf{T}$ we have that $s \notin \mathsf{dom}(\gamma_\mathsf{T}(X_{i+1}))$ and we conclude that $\mathsf{dom}(\gamma_\mathsf{T}(X_{i+1})) \subset \mathsf{dom}(\gamma_\mathsf{T}(X_i))$. □

It follows that the domain of the limit w_T of the iteration sequence with widening is a sound *under-approximation* of the domain of the most precise ranking function w:

Corollary 1. $\mathsf{dom}(\gamma_\mathsf{T}(w_\mathsf{T})) \subseteq \mathsf{dom}(w)$

Proof. Let us assume that $\mathsf{dom}(\gamma_\mathsf{T}(w_\mathsf{T})) \not\subseteq \mathsf{dom}(w)$. Then, by Lemma 1 we have that $\mathsf{dom}(\gamma_\mathsf{T}(w_\mathsf{T} \triangledown_\mathsf{T} \phi_\mathsf{T}(w_\mathsf{T}))) \subset \mathsf{dom}(\gamma_\mathsf{T}(w_\mathsf{T}))$, that is $\phi_\mathsf{T}(w_\mathsf{T}) \not\sqsubseteq_\mathsf{T} w_\mathsf{T}$. Thus w_T would not be the limit of the iteration sequence with widening. □

Lemma 2. $\exists s \in \mathsf{dom}(\gamma_\mathsf{T}(X_i)) \cap \mathsf{dom}(w) : w(s) > \gamma_\mathsf{T}(X_i)(s) \Rightarrow s \notin \mathsf{dom}(\gamma_\mathsf{T}(X_{i+1}))$

Lemma 2 establishes that, if at some iterate X_i the value of the most precise ranking function w is under-approximated (case B in Section 3.3), the iteration sequence with widening $\triangledown_\mathsf{T}$ is not stable.

Proof (of Lemma 2). In case the iteration sequence with widening reaches an iterate X_i such that $\exists s \in \mathsf{dom}(\gamma_\mathsf{T}(X_i)) \cap \mathsf{dom}(w) : w(s) > \gamma_\mathsf{T}(X_i)(s)$ it means that there exists a program trace σ starting at program state s whose length is greater than $\gamma_\mathsf{T}(X_i)(s)$. Let s' be the successor of the state s on the trace σ. Without loss of generality we can assume that σ is the longest program trace starting from s and that $w(s') \leq \gamma_\mathsf{T}(X_i)(s')$. Thus, by the definition of ϕ and the soundness of all abstract operators we have $w(s) = \phi(w)(s) = w(s') + 1 \leq \gamma_\mathsf{T}(X_i)(s') + 1 = \phi(\gamma_\mathsf{T}(X_i))(s) \leq \gamma_\mathsf{T}(\phi_\mathsf{T}(X_i))(s)$, that is $w(s) \leq \gamma_\mathsf{T}(\phi_\mathsf{T}(X_i))(s)$. Now, since $w(s) > \gamma_\mathsf{T}(X_i)(s)$ and $w(s) \leq \gamma_\mathsf{T}(\phi_\mathsf{T}(X_i))(s)$, we have $\gamma_\mathsf{T}(X_i)(s) < \gamma_\mathsf{T}(\phi_\mathsf{T}(X_i))(s)$ and by definition of $\triangledown_\mathsf{T}$ we conclude that $s \notin \mathsf{dom}(\gamma_\mathsf{T}(X_{i+1}))$. □

It follows that the value of the limit w_T of the iteration sequence with widening is a sound *over-approximation* of the value of the most precise ranking function w:

Corollary 2. $\forall s \in \mathsf{dom}(w) \cap \mathsf{dom}(\gamma_\mathsf{T}(w_\mathsf{T})) : w(s) \leq \gamma_\mathsf{T}(w_\mathsf{T})(s)$

Proof. Let us assume that $\exists s \in \mathsf{dom}(w) \cap \mathsf{dom}(\gamma_\mathsf{T}(w_\mathsf{T})) : w(s) > w_\mathsf{T}(s)$. Then, by Lemma 2 we have that $s \notin \mathsf{dom}(\gamma_\mathsf{T}(w_\mathsf{T} \triangledown_\mathsf{T} \phi_\mathsf{T}(w_\mathsf{T})))$, that is $\phi_\mathsf{T}(w_\mathsf{T}) \not\preceq_\mathsf{T} w_\mathsf{T}$. Thus w_T would not be the limit of the iteration sequence with widening. □

Now we are ready to prove Theorem 1:

Proof (of Theorem 1). Follows by definition of $\sqsubseteq$ from Corollary 1 and 2. □

B Simple Example Analysis

Let us consider the simple C program from Example 3:

$$\begin{array}{l}\mathsf{while}\ {}^{1}(\ x > 0\ \wedge\ y > 0\ )\ \{\\ \quad {}^{2}x = x - y;\\ \}^{3}\end{array}$$

At each loop iteration, the value of x is decreased until it becomes less than or equal to zero. The program terminates whatever the initial values for x and y are.

We present the analysis of this toy example with our prototype analyzer. We use our abstract domain of constraint-based decision trees, parameterized with polyhedral [14] constraints at the decision nodes and affine ranking functions [30] at the leaf nodes, and a widening delay of two iterations. The starting point is the constant function equal to zero at the program final control point:

$${}^{3}: \quad \mathsf{LEAF} : 0$$

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:

$${}^{3'}: \quad \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : \bot_{\mathsf{F}}))$$
$$\{\ \mathsf{FILTER}_{\mathsf{T}}({}^{3}, \neg(x > 0 \wedge y > 0))\ \}$$

Note that, the transfer functions for assignments $\mathsf{ASSIGN}_{\mathsf{T}}$ and tests $\mathsf{FILTER}_{\mathsf{T}}$ increase the value of the ranking function at the leaves in order to "count" the program steps.

$${}^{1}: \quad \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : \bot_{\mathsf{F}}))$$
$$\{\ {}^{3'}\ \}$$

After the first iterate the analysis infers that the program terminates in one program step if $x \leq 0$ or $y \leq 0$ (i.e., if the loop condition is not satisfied).

$${}^{2}: \quad \mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 2); (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 2))$$
$$\{\ \mathsf{ASSIGN}_{\mathsf{T}}({}^{1}, x = x - y)\ \}$$

At program control point 2, in order to cope with the assignment $x = x - y$, x is replaced with $x - y$ in the constraint-based decision tree.

$${}^{2'}: \quad \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 3))$$
$$\{\ \mathsf{FILTER}_{\mathsf{T}}({}^{2}, (x > 0 \wedge y > 0))\ \}$$
$${}^{1}: \quad \mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : \bot_{\mathsf{F}})); (\mathsf{LEAF} : 3))$$
$$\{\ {}^{3'}\ \sqcup_{\mathsf{T}}\ {}^{2'}\ \}$$

After the second iterate we know that the program terminates in three step if $x \leq y$ (i.e., if the while loop is executed only once) and in one step if the while loop is not entered.

^{2} : $\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 2); (\mathsf{LEAF} : \bot_{\mathsf{F}})); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 4))); (\mathsf{LEAF} : 2)$

$\{\ \mathsf{ASSIGN}_{\mathsf{T}}(^{1}, x = x - y)\ \}$

$^{2'}$: $\mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 5))); (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 3))))$

$\{\ \mathsf{FILTER}_{\mathsf{T}}(^{2}, (x > 0 \wedge y > 0))\ \}$

$^{1'}$: $\mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : \bot_{\mathsf{F}})); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 5))); (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 3)))$

$\{\ ^{3'} \sqcup_{\mathsf{T}} {}^{2'}\ \}$

After the third iterate we know that the program terminates in five steps if the while loop is executed twice, in three steps if the while loop is executed once, and in one step if the while loop is not entered.

^{1} : $\mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : 2x + 1)); (\mathsf{LEAF} : 3))$

$\{\ ^{1} \triangledown_{\mathsf{T}} {}^{1'}\ \}$

The widening infers that the program terminates in $2x + 1$ program steps if the while loop is entered and in one step otherwise.

^{2} : $\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 2); (\mathsf{LEAF} : 2x - 2y + 2)); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 4))); (\mathsf{LEAF} : 2)$

$\{\ \mathsf{ASSIGN}_{\mathsf{T}}(^{1}, x = x - y)\ \}$

$^{2'}$: $\mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 2x - 2y + 3); (\mathsf{LEAF} : 5))); (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 3))))$

$\{\ \mathsf{FILTER}_{\mathsf{T}}(^{2}, (x > 0 \wedge y > 0))\ \}$

$^{1'}$: $\mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : 2x - 2y + 3)); (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 5))); (\mathsf{NODE}\{x - 2y \leq 0\} : (\mathsf{LEAF} : \bot_{\mathsf{F}}); (\mathsf{LEAF} : 3)))$

$\{\ ^{3'} \sqcup_{\mathsf{T}} {}^{2'}\ \}$

^{1} : $\mathsf{NODE}\{x \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{NODE}\{x - y \leq 0\} : (\mathsf{NODE}\{y \leq 0\} : (\mathsf{LEAF} : 1); (\mathsf{LEAF} : 2x + 1)); (\mathsf{LEAF} : 3))$

$\{\ \text{convergence: } ^{1} \triangledown_{\mathsf{T}} {}^{1'} = {}^{1}\ \}$

At the fourth iteration the analysis with our abstract domain converges: the program terminates in at most $2x + 1$ program steps, whatever the initial values for x and y are.