Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation*

Caterina Urban and Antoine Miné

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

Dedicated to the memory of Radhia Cousot

Abstract. We present new static analysis methods for proving *liveness properties* of programs. In particular, with reference to the hierarchy of temporal properties proposed by Manna and Pnueli, we focus on guarantee (i.e., "something good occurs *at least once*") and recurrence (i.e., "something good occurs *infinitely often*") temporal properties.
We generalize the abstract interpretation framework for termination presented by Cousot and Cousot. Specifically, static analyses of guarantee and recurrence temporal properties are systematically derived by abstraction of the program operational trace semantics.
These methods automatically infer *sufficient preconditions* for the temporal properties by reusing existing numerical abstract domains based on piecewise-defined ranking functions. We augment these abstract domains with new abstract operators, including a *dual widening*.
To illustrate the potential of the proposed methods, we have implemented a research prototype static analyzer, for programs written in a C-like syntax, that yielded interesting preliminary results.

1 Introduction

Temporal properties play a major role in the specification and verification of programs. The hierarchy of temporal properties proposed by Manna and Pnueli [15] distinguishes four basic classes:

- *safety* properties: "something good *always* happens", i.e., the program never reaches an unacceptable state (e.g., partial correctness, mutual exclusion);
- *guarantee* properties: "something good happens *at least once*", i.e., the program *eventually* reaches a desirable state (e.g., termination);
- *recurrence* properties: "something good happens *infinitely often*", i.e., the program reaches a desirable state *infinitely often* (e.g., starvation freedom);
- *persistence* properties: "something good *eventually* happens *continuously*".

* 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).

```
while ¹( x ≥ 0 ) do ²x := x + 1;
while ³( true ) do
    if ⁴( x ≤ 10 ) then ⁵x := x + 1; else ⁶x := −x; fi
```

Fig. 1. Program SIMPLE.

This paper concerns the verification of programs by static analysis. We set our work in the framework of Abstract Interpretation [9], a general theory of semantic approximation that provides a basis for various successful industrial-scale tools (e.g., Astrée [1]). Abstract Interpretation has to a large extent concerned safety properties and has only recently been extended to program termination [11], which is just a particular guarantee property.

In this paper, we generalize the framework proposed by Cousot and Cousot for termination [11] and we propose an abstract interpretation framework for proving *guarantee* and *recurrence* temporal properties of programs. Moreover, we present new static analysis methods for inferring *sufficient preconditions* for these temporal properties. Let us consider the program SIMPLE in Figure 1, where the program variables are interpreted in the set of mathematical integers[1]. The first while loop is an infinite loop for the values of x greater than or equal to zero: at each iteration the value of x is increased by one. The second while loop is an infinite loop: at each iteration, the value of x is increased by one or negated when it becomes greater than ten. Given the guarantee property "$x = 3$ at least once", where $x = 3$ is the desirable state, our approach is able to automatically infer that the property is true if the initial value of x is smaller than or equal to three. Given the recurrence property "$x = 3$ infinitely often", our approach is able to automatically infer that the property is true if the initial value of x is strictly negative (i.e., if the first while loop is never entered).

Our approach follows the traditional methods for proving liveness properties by means of a well-founded argument (i.e., a function from the states of a program to a well-ordered set whose value decreases during program execution). More precisely, we build a well-founded argument for guarantee and recurrence properties in an incremental way: we start from the desirable program states, where the function has value zero (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. Additionally, for recurrence properties, this process is iteratively repeated in order to construct an argument that is also invariant with respect to program execution steps so that even after reaching a desirable state we know that the execution will reach a desirable state again. We formalize these intuitions into *sound* and *complete* guarantee and recurrence semantics that are systematically derived by abstract interpretation of the program operational trace semantics.

In order to achieve effective static analyses, we further abstract these semantics. Specifically, we leverage existing numerical abstract domains based on

[1] For simplicity, this assumption remains valid throughout the rest of the paper.

piecewise-defined ranking functions [21,22,23] by introducing new abstract operators, including a *dual widening*. The piecewise-defined ranking functions are attached to the program control points and represent an *upper bound* on the number of program execution steps before the program reaches a desirable state. They are automatically inferred through backward analysis and yield *sufficient preconditions* for the guarantee and recurrence temporal properties. We prove the soundness of the analysis, meaning that all program executions respecting these preconditions indeed satisfy the temporal properties, while a program execution that does not respect these preconditions might or might not satisfy the temporal properties.

To illustrate the potential of our approach, let us consider again the program SIMPLE in Figure 1. Given the guarantee property "$x = 3$ at least once", the piecewise-defined ranking function inferred at program control point 1 is:

$$f_1^g(x) = \begin{cases} -3x+10 & x < 0 \\ -2x+6 & x \geq 0 \wedge x \leq 3 \\ \text{undefined} & \text{otherwise} \end{cases}$$

which bounds the wait (from the control point 1) for the desirable state $x = 3$ by $-3x+10$ program execution steps if $x<0$, and by $-2x+6$ execution steps if $x \geq 0 \wedge x \leq 3$. In case $x>3$, the analysis is inconclusive. In fact, if $x>3$ the guarantee property is never true so the precondition $x \leq 3$ for the guarantee property is the weakest precondition. Given the recurrence property "$x=3$ infinitely often", the piecewise-defined ranking function at program point 1 bounds the wait for the *next* occurrence of the desirable state $x=3$ by $-3x+10$ program execution steps:

$$f_1^r(x) = \begin{cases} -3x+10 & x < 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that, if $x \geq 0 \wedge x \leq 3$, $x=3$ occurs once but not infinitely often: f_1^g is defined for $x \geq 0 \wedge x \leq 3$ but not f_1^r. Again, the sufficient precondition $x < 0$ is also a necessary precondition. At program point 3 (i.e., at the beginning of the second while loop), we get the following piecewise-defined ranking function:

$$f_3^g(x) = f_3^r(x) = \begin{cases} -3x+9 & x \leq 3 \\ -3x+72 & x > 3 \end{cases}$$

which bounds the wait (from the control point 3) for the next occurrence of $x=3$ by $-3x+9$ execution steps if $x \leq 3$, and by $-3x+72$ execution steps if $x>3$.

Our Contribution. In summary, this paper makes the following contributions. First, we present an abstract interpretation framework for proving *guarantee* and *recurrence* temporal properties of programs. In particular, we generalize the framework proposed by Cousot and Cousot for termination [11]. Moreover, by means of piecewise-defined ranking function abstract domains [21,22,23], we design new static analysis methods to effectively infer *sufficient preconditions* for these temporal properties, and provide *upper bounds* in terms of program execution steps on the wait before a program reaches a desirable state. Finally, we provide a research prototype static analyzer for programs written in a C-like syntax.

Limitations. In general, liveness properties are used to specify the behavior of *concurrent* programs and are satisfied only under *fairness* hypotheses. In this paper, we model concurrent programs as non-deterministic sequential programs and we assume that the fair scheduler is explicitly represented within the program (e.g., see [13] and Example 6 in Section 8). We plan, as part of our future work, to extend our framework in order to explicitly express and handle fairness properties.

Outline of the Paper. Section 2 introduces the preliminary notions used in the paper. In Section 3, we give a brief overview of Cousot and Cousot's abstract interpretation framework for termination. In Section 4, we define a small specification language to describe guarantee and recurrence properties. The next two sections are devoted to the main contribution of the paper: we formalize our framework for guarantee and recurrence properties in Section 5 and in Section 6, respectively. In Section 7, we present decidable guarantee and recurrence abstractions based on piecewise-defined ranking functions. We describe our prototype static analyzer in Section 8. Finally, Section 9 discusses related work and Section 10 concludes.

2 Trace Semantics

Following [8,11], as a model of the operational semantics of a program, we use a *transition system* $\langle\Sigma,\tau\rangle$, where Σ is the (possibly infinite) set of program states and the program transition relation $\tau\subseteq\Sigma\times\Sigma$ describes the possible transitions between states during program execution. Note that this model allows representing programs with (possibly unbounded) non-determinism.

Let Σ^n be the set of all finite program state sequences of length $n\in\mathbb{N}$. We use ε to denote the empty sequence, i.e., $\Sigma^0 \triangleq \{\varepsilon\}$. The set of final states $\Omega \triangleq \{s\in\Sigma \mid \forall s'\in\Sigma\colon \langle s,s'\rangle\notin\tau\}$ can be understood as a set of sequences of length one and the program transition relation τ can be understood as a set of sequences of length two. Let $\Sigma^+ \triangleq \bigcup_{n\in\mathbb{N}^+}\Sigma^n$ be the set of all non-empty finite sequences, $\Sigma^* \triangleq \Sigma^0\cup\Sigma^+$ be the set of all finite sequences, Σ^ω be the set of all infinite sequences, $\Sigma^{+\infty} \triangleq \Sigma^+\cup\Sigma^\omega$ be the set of all non-empty finite or infinite sequences and $\Sigma^{*\infty} \triangleq \Sigma^*\cup\Sigma^\omega$ be the set of all finite or infinite sequences. We write $\sigma\sigma'$ for the concatenation of sequences $\sigma,\sigma'\in\Sigma^{+\infty}$ (with $\sigma\epsilon=\epsilon\sigma=\sigma$ and $\sigma\sigma'=\sigma$ when $\sigma\in\Sigma^\omega$), $T^+ \triangleq T\cap\Sigma^+$ for the selection of the non-empty finite sequences of $T\subseteq\Sigma^{+\infty}$, $T^\omega \triangleq T\cap\Sigma^\omega$ for the selection of the infinite sequences of $T\subseteq\Sigma^{+\infty}$ and $T\,;T' \triangleq \{\sigma s\sigma' \mid s\in\Sigma \wedge \sigma s\in T \wedge s\sigma'\in T'\}$ for the merging of sets of sequences $T,T'\subseteq\Sigma^{+\infty}$.

The *maximal trace semantics* $\tau^{+\infty}\subseteq\Sigma^{+\infty}$ generated by a transition system $\langle\Sigma,\tau\rangle$ is the union of the set of all non-empty finite program execution traces that are terminating with a final state, and the set of all infinite execution traces. It can be expressed as a least fixpoint in the complete lattice $\langle\Sigma^{+\infty},\sqsubseteq,\sqcup,\sqcap,\Sigma^\omega,\Sigma^+\rangle$ [8]:

$$
\begin{aligned}
\tau^{+\infty} &\triangleq \mathrm{lfp}^{\sqsubseteq}\,\phi^{+\infty}\\
\phi^{+\infty}(T) &\triangleq \Omega\sqcup(\tau\,;T)
\end{aligned}
\tag{1}
$$

where $T_1\sqsubseteq T_2 \triangleq T_1^+\subseteq T_2^+ \wedge T_1^\omega\supseteq T_2^\omega$ and $T_1\sqcup T_2 \triangleq (T_1^+\cup T_2^+)\cup(T_1^\omega\cap T_2^\omega)$.

3 Termination Semantics

The Floyd/Turing 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., $\langle \mathbb{O}, < \rangle$, the well-ordered set of ordinals) whose value decreases during program execution.

In [11], Cousot and Cousot prove the existence of a *most precise program ranking function*[2] $\tau^{\mathrm{t}} \in \Sigma \rightharpoonup \mathbb{O}$ that can be expressed in fixpoint form as follows:

$$\begin{aligned} \tau^{\mathrm{t}} &\triangleq \mathrm{lfp}^{\preccurlyeq}_{\dot{\emptyset}}\ \phi^{\mathrm{t}} \\ \phi^{\mathrm{t}}(v) &\triangleq \lambda s. \begin{cases} 0 & s \in \Omega \\ \sup\{v(s')+1 \mid \langle s,s' \rangle \in \tau\} & s \in \widetilde{\mathrm{pre}}(\mathrm{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

where $\dot{\emptyset}$ is the totally undefined function, $v_1 \preccurlyeq v_2 \triangleq \mathrm{dom}(v_1) \subseteq \mathrm{dom}(v_2) \wedge \forall x \in \mathrm{dom}(v_1)\colon v_1(x) \leq v_2(x)$ and $\widetilde{\mathrm{pre}}(X) \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma\colon \langle s,s' \rangle \in \tau \Rightarrow s' \in X\}$.

The most precise ranking function τ^{t} extracts the well-founded part of the transition relation τ: starting from the final states in Ω, where the function has value zero, and retracing the program backwards while 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 are terminating) to an ordinal in $\mathbb{O}$ representing an upper bound on the number of program execution steps remaining to termination. The domain $\mathrm{dom}(\tau^{\mathrm{t}})$ of τ^{t} is the set of states definitely leading to program termination; any trace starting in a state $s \in \mathrm{dom}(\tau^{\mathrm{t}})$ must terminate in at most $\tau^{\mathrm{t}}(s)$ execution steps, while at least one trace starting in a state $s \notin \mathrm{dom}(\tau^{\mathrm{t}})$ does not terminate:

Theorem 1. *A program terminates for all execution traces starting from an initial state $s \in \Sigma$ if and only if $s \in \mathrm{dom}(\tau^{\mathrm{t}})$.*

We would like to emphasize the elegance of the abstract interpretation theory which allows to tie together seemingly unrelated semantics by *different* abstractions of the *same* operational trace semantics, i.e., the maximal trace semantics (1) [8]. The semantics, rather than being first derived by intuition and then proved correct, are systematically derived by abstract interpretation. Specifically, in [11], in order to derive the most precise ranking function (2), Cousot and Cousot define the following abstraction functions:

- The *prefix abstractions* $\mathrm{pf} \in \Sigma^{+\infty} \to \mathcal{P}(\Sigma^{+\infty})$ and $\mathrm{pf} \in \mathcal{P}(\Sigma^{+\infty}) \to \mathcal{P}(\Sigma^{+\infty})$ yield respectively the set of prefixes of a sequence $\sigma \in \Sigma^{+\infty}$ and the set of prefixes of a set of sequences $T \subseteq \Sigma^{+\infty}$:

$$\begin{aligned} \mathrm{pf}(\sigma) &\triangleq \{\sigma' \in \Sigma^{+\infty} \mid \exists \sigma'' \in \Sigma^{*\infty}\colon \sigma = \sigma'\sigma''\} \\ \mathrm{pf}(T) &\triangleq \bigcup\{\mathrm{pf}(\sigma) \mid \sigma \in T\} \end{aligned} \tag{3}$$

  The *neighborhood* of a sequence $\sigma \in \Sigma^{+\infty}$ in a set of sequences $T \subseteq \Sigma^{+\infty}$ is the set of sequences $\sigma' \in T$ with a common prefix with σ: $\{\sigma' \in T \mid \mathrm{pf}(\sigma) \cap \mathrm{pf}(\sigma') \neq \emptyset\}$.

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

- The *termination abstraction* $\alpha^{\mathrm{t}} \in \mathcal{P}(\Sigma^{+\infty}) \rightarrow \mathcal{P}(\Sigma^{+})$ selects from a set of sequences $T \subseteq \Sigma^{+\infty}$ the sequences that are finite and whose neighborhood in T consists only of finite traces:

$$\alpha^{\mathrm{t}}(T) \triangleq \{\sigma \in T^{+} \mid \mathrm{pf}(\sigma) \cap \mathrm{pf}(T^{\omega}) = \emptyset\} \tag{4}$$

  Example 1. Let $T = \{ab, aba, ba, bb, ba^{\omega}\}$, then $\alpha^{\mathrm{t}}(T) = \{ab, aba\}$. In fact, $\mathrm{pf}(ab) \cap \mathrm{pf}(ba^{\omega}) = \emptyset$ and $\mathrm{pf}(aba) \cap \mathrm{pf}(ba^{\omega}) = \emptyset$, while $\mathrm{pf}(ba) \cap \mathrm{pf}(ba^{\omega}) = \{b, ba\}$ and $\mathrm{pf}(bb) \cap \mathrm{pf}(ba^{\omega}) = \{b\}$. □

- The *transition abstraction* $\vec{\alpha} \in \mathcal{P}(\Sigma^{+\infty}) \rightarrow \mathcal{P}(\Sigma \times \Sigma)$ extracts from a set of sequences $T \subseteq \Sigma^{+\infty}$ the smallest transition relation $r \subseteq \Sigma \times \Sigma$ that generates T:

$$\vec{\alpha}(T) \triangleq \{\langle s, s'\rangle \mid \exists \sigma, \sigma' \in \Sigma^{*\infty} \colon \sigma s s' \sigma' \in T\}$$

- The *ranking abstraction* $\alpha^{\mathrm{rk}} \in \mathcal{P}(\Sigma \times \Sigma) \rightarrow (\Sigma \rightharpoonup \mathbb{O})$ provides the rank of the elements in the domain of a relation $r \subseteq \Sigma \times \Sigma$:

$$\alpha^{\mathrm{rk}}(r)s \triangleq \begin{cases} 0 & \forall s' \in \Sigma \colon \langle s, s'\rangle \notin r \\ \sup\left\{ \alpha^{\mathrm{rk}}(r)s' + 1 \;\middle|\; \begin{array}{c} s' \in \mathrm{dom}(\alpha^{\mathrm{rk}}(r)) \\ \wedge \langle s, s'\rangle \in r \end{array} \right\} & \text{otherwise} \end{cases}$$

- The *variant abstraction* $\alpha^{\mathrm{v}} \in \mathcal{P}(\Sigma^{+}) \rightarrow (\Sigma \rightharpoonup \mathbb{O})$ provides the rank of the elements in the domain of the smallest transition relation that generates a set of sequences $T \subseteq \Sigma^{+}$:

$$\alpha^{\mathrm{v}}(T) \triangleq \alpha^{\mathrm{rk}}(\vec{\alpha}(T)) \tag{5}$$

The most precise ranking function (2) can now be explicitly defined as abstract interpretation of the program maximal trace semantics (1) [11]:

$$\tau^{\mathrm{t}} \triangleq \alpha^{\mathrm{v}}(\alpha^{\mathrm{t}}(\tau^{+\infty}))$$

In Section 5 and Section 6, we will follow the same abstract interpretation approach in order to systematically derive sound and complete semantics for proving guarantee and recurrence temporal properties of programs.

4 Specification Language

In general, we define a program *property* as a set of sequences. A program has a certain property if all the program execution traces belong to the property. In this paper, with respect to the hierarchy of temporal properties proposed in [15], we focus on guarantee ("something good happens *at least once*") and recurrence ("something good happens *infinitely often*") properties. In particular, we consider guarantee and recurrence properties that are expressible by temporal logic.

We define a small specification language, which will be used to describe properties of program states. 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 $\mathcal{E} \triangleq \mathcal{X} \rightarrow \mathbb{Z}$,

$$\delta ::= X \mid n \mid -\delta \mid \delta_1 \diamond \delta_2 \qquad X \in \mathcal{X}, n \in \mathbb{Z}, \diamond \in \{+,-,*,/\}$$
$$\beta ::= \text{true} \mid \text{false} \mid !\beta \mid \beta_1 \vee \beta_2 \mid \beta_1 \wedge \beta_2 \mid \delta_1 \bowtie \delta_2 \qquad \bowtie \in \{<,\leq,=,\neq,>,\geq\}$$
$$\varphi ::= \beta \mid l\!:\!\beta \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \qquad l \in \mathcal{L}$$

Fig. 2. Syntax of State Properties.

which map each program variable to an integer value. In Figure 2 we define inductively the syntax of the state properties. The predicate $l\!:\!\beta$ allows specifying a state property at a particular control point $l \in \mathcal{L}$. We write $s \models \varphi$ when the state $s \in \Sigma$ has the property φ, and $\Sigma_\varphi \triangleq \{s \in \Sigma \mid s \models \varphi\}$ for the set of states $\Sigma_\varphi \subseteq \Sigma$ that have the property φ.

In the following, we define the program properties of interest by means of the temporal operators *always* $\Box$ and *eventually* $\Diamond$.

The *guarantee properties* are expressible by a temporal formula of the form $\Diamond\varphi$, for some state property φ. The formula expresses that at least one state in every program execution trace has the property φ, but it does not promise any repetition. In general, the guarantee properties are used to ensure that some event happens once during a program execution, such as program termination or eventual consistency. Indeed, program termination can be expressed as the guarantee property $\Diamond l_e\!:\!\text{true}$, where $l_e \in \mathcal{L}$ is the program final control point.

The *recurrence properties* are expressible by a temporal formula of the form $\Box\Diamond\varphi$, for some state property φ. The formula expresses that infinitely many states in every program execution trace have the property φ. In general, the recurrence properties are used to ensure that some event happens infinitely many times during a program execution (e.g., a request is always eventually answered).

5 Guarantee Temporal Properties

In the following, we generalize Section 3 from termination to guarantee properties. We define a sound and complete semantics for proving guarantee temporal properties by abstract interpretation of the program maximal trace semantics.

Let $S \subseteq \Sigma$ be a set of states and let $S^{+\infty} \subseteq \Sigma^{+\infty}$ be the set of non-empty finite or infinite sequences of states in $S \subseteq \Sigma$. In the following, we write $\bar{S} \triangleq \Sigma \setminus S$ for the set of states that are not in S and $T^S \triangleq T \cap S^{+\infty}$ for the selection of the sequences of T that are non-empty sequences of states in S.

In order to define our semantics we need the following abstraction functions:

– The *subsequence abstraction* $\alpha^{\mathrm{s}} \in \mathcal{P}(\Sigma^{+\infty}) \to \mathcal{P}(\Sigma^{+\infty})$ extracts from a set of sequences $T \subseteq \Sigma^{+\infty}$ the subsequences of sequences in T:

$$\alpha^{\mathrm{s}}(T) \triangleq \{\sigma \in \Sigma^{+\infty} \mid \exists \sigma' \in \Sigma^*, \sigma'' \in \Sigma^{*\infty} : \sigma'\sigma\sigma'' \in T\}$$

– The *guarantee abstraction* $\alpha^{\mathrm{g}} \in \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma^{+\infty}) \to \mathcal{P}(\Sigma^{+})$, given a set of states $S \subseteq \Sigma$ and a set of sequences $T \subseteq \Sigma^{+\infty}$, extracts from $T \subseteq \Sigma^{+\infty}$ the

subsequences of sequences of T whose neighborhood in $\alpha^{\mathrm{s}}(T)$ consists only of sequences of states in $\bar{S}$ that are terminating with a state $s \in S$:

$$\alpha^{\mathrm{g}}(S)T \triangleq \left\{ \sigma s \in \alpha^{\mathrm{s}}(T) \cap \Sigma^{+} \;\middle|\; \begin{array}{c} s \in S \wedge \sigma \in \bar{S}^{*} \wedge \\ \forall \sigma' \in \mathrm{pf}(\sigma)\colon T^{\sigma'} \cap \bar{S}^{+\infty} = \emptyset \end{array} \right\} \tag{6}$$

where $\mathrm{pf} \in \Sigma^{+\infty} \to \mathcal{P}(\Sigma^{+\infty})$ is the prefix abstraction (3) of Section 3 and $T^{\sigma} \triangleq \{\sigma\sigma'' \in \Sigma^{+\infty} \mid \sigma'' \in \Sigma^{*\infty} \wedge \exists \sigma' \in \Sigma^{*\infty}\colon \sigma'\sigma\sigma'' \in T\}$ is the set of suffixes of sequences of $T \subseteq \Sigma^{+\infty}$ with prefix $\sigma \in \Sigma^{+\infty}$.

Example 2. Let $T = \{cd^{\omega}, (cd)^{\omega}\}$, then $T^{d} = \{d^{\omega}, (dc)^{\omega}\}$. □

Example 3. Let $T = \{(abcd)^{\omega}, (cd)^{\omega}, a^{\omega}, cd^{\omega}\}$ and $S = \{c\}$, then $\alpha^{\mathrm{g}}(S)T = \{c, bc\}$. Let us consider the trace $(abcd)^{\omega}$: the subsequences of $(abcd)^{\omega}$ that are terminating with c (and never encounter c before) are $\{c, bc, abc, dabc\}$. Let us consider the subsequence abc: $T^{ab} \cap \bar{S}^{+\infty} = \emptyset$ but $T^{a} \cap \bar{S}^{+\infty} = \{a^{\omega}\}$. Now let us consider $dabc$: $T^{dab} \cap \bar{S}^{+\infty} = \emptyset$ and $T^{da} \cap \bar{S}^{+\infty} = \emptyset$ but $T^{d} \cap \bar{S}^{+\infty} = \{d^{\omega}\}$. □

The *guarantee semantics* $\tau^{\mathrm{g}} \in \mathcal{P}(\Sigma) \to (\Sigma \rightharpoonup \mathbb{O})$ of a program can now be defined by abstract interpretation of the program maximal trace semantics (1):

$$\begin{aligned} \tau^{\mathrm{g}}(S) &\triangleq \alpha^{\mathrm{v}}(\alpha^{\mathrm{g}}(S)\tau^{+\infty}) = \mathrm{lfp}^{\preccurlyeq}_{\dot{\emptyset}}\; \phi^{\mathrm{g}}(S) \\ \phi^{\mathrm{g}}(S)v &\triangleq \lambda s. \begin{cases} 0 & s \in S \\ \sup\{v(s')+1 \mid \langle s, s'\rangle \in \tau\} & s \in \widetilde{\mathrm{pre}}(\mathrm{dom}(v)) \wedge s \notin S \\ \text{undefined} & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

where $\alpha^{\mathrm{v}} \in \mathcal{P}(\Sigma^{+}) \to (\Sigma \rightharpoonup \mathbb{O})$ is the variant abstraction (5) presented in Section 3.

Intuitively, given a set of states $S \subseteq \Sigma$, the guarantee semantics $\tau^{\mathrm{g}}(S)$ is defined starting from the states in S and retracing the program backwards while mapping each program state definitely leading to S (i.e., a program state such that all the traces to which it belongs eventually reach a state in S) to an ordinal in $\mathbb{O}$ representing an upper bound on the number of program execution steps remaining to S. The domain $\mathrm{dom}(\tau^{\mathrm{g}}(S))$ of $\tau^{\mathrm{g}}(S)$ is the set of states definitely leading to a state in S: any trace starting in a state $s \in \mathrm{dom}(\tau^{\mathrm{g}}(S))$ must reach a state in S in at most $\tau^{\mathrm{g}}(S)s$ execution steps, while at least one trace starting in a state $s \notin \mathrm{dom}(\tau^{\mathrm{g}}(S))$ does not reach S.

Note that, when S is the set of final states Ω, $\phi^{\mathrm{g}}(\Omega) = \phi^{\mathrm{t}}$ and we rediscover precisely Cousot and Cousot's termination semantics [11] presented in Section 3.

Let φ be a state property. We define the φ-*guarantee semantics* $\tau^{\mathrm{g}}_{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$:

$$\tau^{\mathrm{g}}_{\varphi} \triangleq \tau^{\mathrm{g}}(\Sigma_{\varphi}) \tag{8}$$

The semantics $\tau^{\mathrm{g}}_{\varphi}$ is sound and complete for proving a guarantee property $\Diamond\varphi$:

Theorem 2. *A program satisfies a guarantee property $\Diamond\varphi$ for all execution traces starting from an initial state $s \in \Sigma$ if and only if $s \in \mathrm{dom}(\tau^{\mathrm{g}}_{\varphi})$.*

6 Recurrence Temporal Properties

In the following, we define a sound and complete semantics for proving recurrence temporal properties by abstract interpretation of the program maximal trace semantics, following the same approach used in Section 5 for guarantee temporal properties. In particular, the recurrence semantics that we are going to define reuses the guarantee semantics of Section 5 as starting point: from the guarantee that some event happens once during a program execution, the recurrence semantics ensures that the event happens infinitely many times.

In order to define our semantics we need the following abstraction function:

– The *recurrence abstraction* $\alpha^{\mathrm{r}} \in \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma^{+\infty}) \to \mathcal{P}(\Sigma^{+})$, given a set of states $S \subseteq \Sigma$ and a set of sequences $T \subseteq \Sigma^{+\infty}$, extracts from T the subsequences of sequences of T whose neighborhood in $\alpha^{\mathrm{s}}(T)$ consists only of sequences of states in $\bar{S}$ that are terminating with a state in S, and that are prefixes of sequences of T that reach infinitely often a state in S:

$$\begin{aligned} \alpha^{\mathrm{r}}(S)T &\triangleq \mathrm{gfp}^{\subseteq}_{\alpha^{\mathrm{g}}(S)T}\ \phi^{\alpha^{\mathrm{r}}}(T,S) \\ \phi^{\alpha^{\mathrm{r}}}(T,S)T' &\triangleq \alpha^{\mathrm{g}}(\widetilde{\mathrm{pre}}(T)T' \cap S)T \end{aligned} \tag{9}$$

where $\widetilde{\mathrm{pre}}(T)T' \triangleq \{s \in \Sigma \mid \forall \sigma \in \Sigma^{*}, \sigma' \in \Sigma^{*\infty} : \sigma s \sigma' \in T \Rightarrow \mathrm{pf}(\sigma') \cap T' \neq \emptyset\}$ and $\alpha^{\mathrm{g}} \in \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma^{+\infty}) \to \mathcal{P}(\Sigma^{+})$ is the guarantee abstraction (6) of Section 5. To explain intuitively (9), we use the Kleene dual fixpoint theorem [8] to rephrase $\alpha^{\mathrm{r}}(S)T$ as follows:

$$\alpha^{\mathrm{r}}(S)T = \bigcap_{i \in \mathbb{N}} T_{i+1} \quad \text{where } T_{i+1} \triangleq \left[\phi^{\alpha^{\mathrm{r}}}(T,S)\right]^{i} (\alpha^{\mathrm{g}}(S)T)$$

Then, for $i=0$, we get the set $T_1 = \alpha^{\mathrm{g}}(S)T$ of subsequences of sequences of T that guarantee S at least *once*. For $i=1$, starting from T_1, we derive the set of states $S_1 = \widetilde{\mathrm{pre}}(T)T_1 \cap S$ (i.e., $S_1 \subseteq S$) whose successors all belong to the subsequences in T_1, and we get the set $T_2 = \alpha^{\mathrm{g}}(S_1)T$ of subsequences of sequences of T that guarantee S_1 at least once and thus guarantee S at least *twice*. Note that all the subsequences in T_2 terminate with a state $s' \in S_1$ and therefore are *prefixes* of subsequence of T that reach S at least twice. More generally, for each $i \in \mathbb{N}$, we get the set T_{i+1} of subsequences which are *prefixes* of subsequences of T that reach S at least $i+1$ times, i.e., the subsequences that guarantee S at least $i+1$ times. The greatest fixpoint thus guarantees S *infinitely often*.

Example 4. Let $T = \{(cd)^{\omega}, ca^{\omega}, d(be)^{\omega}\}$ and let $S = \{b,c,d\}$. For $i=0$, we have $T_1 = \alpha^{\mathrm{g}}(S)T = \{b, eb, c, d\}$. For $i=1$, we derive $S_1 = \{b,d\}$, since $c(dc)^{\omega} \in T$ and $\mathrm{pf}((dc)^{\omega}) \cap T_1 = \{d\} \neq \emptyset$ but $ca^{\omega} \in T$ and $\mathrm{pf}(a^{\omega}) \cap T_1 = \emptyset$. We get $T_2 = \alpha^{\mathrm{g}}(S_1)T = \{b, eb, d\}$. For $i=2$, we derive $S_2 = \{b\}$, since $d(be)^{\omega} \in T$ and $\mathrm{pf}((be)^{\omega}) \cap T_1 = \{b\} \neq \emptyset$ but $d(cd)^{\omega} \in T$ and $\mathrm{pf}((cd)^{\omega}) \cap T_2 = \emptyset$. We get $T_3 = \alpha^{\mathrm{g}}(S_2)T = \{b, eb\}$ which is the greatest fixpoint: the only subsequences of sequences in T that guarantee S infinitely often start with b or eb. □

The *recurrence semantics* $\tau^{\mathrm{r}} \in \mathcal{P}(\Sigma) \rightarrow (\Sigma \rightharpoonup \mathbb{O})$ of a program can now be defined by abstract interpretation of the program maximal trace semantics (1):

$$\begin{aligned} \tau^{\mathrm{r}}(S) &\triangleq \alpha^{\mathrm{v}}(\alpha^{\mathrm{r}}(S)\tau^{+\infty}) = \mathrm{gfp}^{\preccurlyeq}_{\tau^{\mathrm{g}}(S)}\, \phi^{\mathrm{r}}(S) \\ \phi^{\mathrm{r}}(S)v &\triangleq \lambda s. \begin{cases} v(s) & s \in \mathrm{dom}(\tau^{\mathrm{g}}(\widetilde{\mathrm{pre}}(\mathrm{dom}(v)) \cap S)) \\ \text{undefined} & \text{otherwise} \end{cases} \end{aligned} \quad (10)$$

where $\alpha^{\mathrm{v}} \in \mathcal{P}(\Sigma^{+}) \rightarrow (\Sigma \rightharpoonup \mathbb{O})$ is the variant abstraction (5) presented in Section 3 and $\tau^{\mathrm{g}} \in \mathcal{P}(\Sigma) \rightarrow (\Sigma \rightharpoonup \mathbb{O})$ is the guarantee semantics (7) defined in Section 5. Note that, given the definition of (7), (10) contains a nested fixpoint.

Given a set of states $S \subseteq \Sigma$, the recurrence semantics $\tau^{\mathrm{r}}(S)$ maps each program state definitely leading infinitely many times to S to an ordinal in $\mathbb{O}$ representing an upper bound on the number of execution steps remaining to the *next occurrence* of a state in S: any trace starting in a state $s \in \mathrm{dom}(\tau^{\mathrm{r}}(S))$ must reach the next occurrence of a state in S in at most $\tau^{\mathrm{r}}(S)s$ execution steps, while at least one trace starting in a state $s \notin \mathrm{dom}(\tau^{\mathrm{r}}(S))$ reaches a state in S at most finitely many times.

Let φ be a state property. We the define φ*-recurrence semantics* $\tau^{\mathrm{r}}_{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$:

$$\tau^{\mathrm{r}}_{\varphi} \triangleq \tau^{\mathrm{r}}(\Sigma_{\varphi}) \quad (11)$$

The semantics $\tau^{\mathrm{r}}_{\varphi}$ is sound and complete for proving a recurrence property $\Box\Diamond\varphi$:

Theorem 3. *A program satisfies a recurrence property* $\Box\Diamond\varphi$ *for all execution traces starting from an initial state* $s \in \Sigma$ *if and only if* $s \in \mathrm{dom}(\tau^{\mathrm{r}}_{\varphi})$.

7 Piecewise-Defined Ranking Functions

The termination semantics τ^{t} of Section 3, the φ-guarantee semantics $\tau^{\mathrm{g}}_{\varphi}$ of Section 5 and the φ-recurrence semantics $\tau^{\mathrm{r}}_{\varphi}$ of Section 6 are usually *not computable* (i.e., when the program state space is infinite).

In [21,22,23], we present decidable abstractions of τ^{t} by means of piecewise-defined ranking functions over natural numbers [21], over ordinals [22] and with relational partitioning [23]. In the following, we will briefly recall the main characteristics of these abstractions and we will show how to modify the abstract domains in order to obtain decidable abstractions of $\tau^{\mathrm{g}}_{\varphi}$ and $\tau^{\mathrm{r}}_{\varphi}$ as well.

7.1 Abstract Termination Semantics

The formal treatment given in the previous sections is defined over general transition systems. In practice, it is sufficient to provide a transfer function for each atomic instruction of a programming language to define a semantics for all the programs in the language and obtain an effective static analysis after opportune abstraction.

In [21], we provide an isomorphic definition of the termination semantics $\tau^{\mathrm{t}} \in \Sigma \rightharpoonup \mathbb{O}$ for a C-like programming language by partitioning with respect to the set of program control points $\mathcal{L}$: $\tau^{\mathrm{t}} \in \mathcal{L} \rightarrow (\mathcal{E} \rightharpoonup \mathbb{O})$. In this way, to each control

point $l \in \mathcal{L}$ corresponds a function $v \in \mathcal{E} \rightharpoonup \mathbb{O}$ and to each program statement i corresponds a transfer function $[\![i]\!]^{\mathrm{t}} \in (\mathcal{E} \rightharpoonup \mathbb{O}) \to (\mathcal{E} \rightharpoonup \mathbb{O})$. As an example, given an assignment $x := e$ of the expression e to the variable $x \in \mathcal{X}$, the transfer function is defined as follows:

$$[\![x := e]\!]^{\mathrm{t}} v \triangleq \lambda\rho. \begin{cases} \sup\{v(\rho[x \mapsto z]) + 1 \mid z \in [\![e]\!]\rho\} & \forall z \in [\![e]\!]\rho\colon \rho[x \mapsto z] \in \mathrm{dom}(v) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $[\![e]\!] \in \mathcal{E} \to \wp(\mathbb{Z})$ maps an environment $\rho \in \mathcal{E}$ to the set of all possible values for the expression e in the given environment. In case of a loop statement the transfer function involves a least fixpoint. More details can be found in [21].

Subsequently, in [21,22,23] we present an abstract termination semantics $\tau^{\alpha^{\mathrm{t}}} \in \mathcal{L} \to \mathcal{V}$: to each program control point $l \in \mathcal{L}$ corresponds an element $v \in \mathcal{V}$ of an abstract domain $\mathcal{V}$, equipped with a concretization function $\gamma \in \mathcal{V} \to (\mathcal{E} \rightharpoonup \mathbb{O})$ and a sound abstract transfer function $[\![i]\!]^{\alpha^{\mathrm{t}}} \in \mathcal{V} \to \mathcal{V}$ for each program statement i. In particular, the elements of the abstract domain $\mathcal{V}$ are piecewise-defined ranking functions represented by means of two parameter abstract domains: an abstract domain whose elements establish the shape of the pieces of the ranking functions, and an abstract domain whose elements represent the value of the ranking functions within their pieces. As an example, in [21] we consider piecewise-defined ranking functions whose pieces have the shape of intervals and whose value is represented by an affine function.

The abstract transfer functions are combined together to compute an abstract ranking function for a program through backward analysis. The starting point is the constant function equal to zero at the program final control point. This function is then propagated backwards towards the program initial control point taking assignments and tests into account and, in case of loops, solving least fixpoints by iteration with a widening operator. We give an intuition for how the abstract assignment transfer function works by means of the following example:

Example 5. Let us consider the piecewise-defined ranking function with value $2x{+}1$ for $x \in [-\infty,3]$ and undefined elsewhere, and the assignment $x := x + 1$. The abstract assignment transfer function substitutes the variable x with the expression $x + 1$ and increases the value of the function by one (to take into account that one more program step is needed before termination). The result is the piecewise-defined ranking function with value $2(x{+}1){+}1{+}1 = 2x{+}4$ for $x{+}1 \in [-\infty,3]$ (i.e., $x \in [-\infty,2]$) and undefined elsewhere. □

We refer to [21,22,23] for more details.

The abstract transfer functions are *sound* with respect to the approximation order $v_1 \sqsubseteq v_2 \triangleq \mathrm{dom}(v_1) \supseteq \mathrm{dom}(v_2) \wedge \forall x \in \mathrm{dom}(v_2)\colon v_1(x) \leq v_2(x)$ (see [10] for further discussion on approximation and computational order of an abstract domain):

Theorem 4. $[\![i]\!]^{\mathrm{t}} \gamma(v) \sqsubseteq \gamma([\![i]\!]^{\alpha^{\mathrm{t}}} v)$

The backward analysis computes an over-approximation of the value of the most precise ranking function τ^{t} and an *under-approximation* of its domain of definition $\mathrm{dom}(\tau^{\mathrm{t}})$. In this way, an abstraction provides *sufficient preconditions* for program

termination: if the abstraction is defined on a program state, then all the program execution traces branching from that state are terminating.

7.2 Abstract Guarantee Semantics

In the following, we describe how to reuse the piecewise-defined ranking function abstract domains introduced in [21,22,23] and what changes are required in order to obtain decidable abstractions of the φ-guarantee semantics $\tau_\varphi^{\mathrm{g}}$ (8).

First, as before, we partition the φ-guarantee semantics $\tau_\varphi^{\mathrm{g}} \in \Sigma \rightharpoonup \mathbb{O}$ with respect to the set of program control points $\mathcal{L}$: $\tau_\varphi^{\mathrm{g}} \in \mathcal{L} \to (\mathcal{E} \rightharpoonup \mathbb{O})$. The transfer functions $[\![i]\!]_\varphi^{\mathrm{g}} \in (\mathcal{E} \rightharpoonup \mathbb{O}) \to (\mathcal{E} \rightharpoonup \mathbb{O})$ behave as the transfer functions for the termination semantics but in addition they *reset* the value of the ranking function for the environments that have the property φ. As an example, the transfer function for an assignment $x := e$ is now defined as follows:

$$[\![x := e]\!]^{\mathrm{g}} v \triangleq \lambda\rho. \begin{cases} 0 & \rho \models \varphi \\ \sup\{v(\rho[x \mapsto z]) + 1 \mid z \in [\![e]\!]\rho\} & \rho \not\models \varphi \wedge \forall z \in [\![e]\!]\rho\colon \rho[x \mapsto z] \in \mathrm{dom}(v) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $\rho \models \varphi$ means that the environment $\rho \in \mathcal{E}$ has the property φ (cf. Section 4).

Then, we define the abstract φ-guarantee semantics $\tau_\varphi^{\alpha^{\mathrm{g}}} \in \mathcal{L} \to \mathcal{V}$: to each program control point $l \in \mathcal{L}$ corresponds a piecewise-defined ranking function $v \in \mathcal{V}$. To each program statement i corresponds a *sound* (with respect to the approximation order $\sqsubseteq$) abstract transfer function $[\![i]\!]_\varphi^{\alpha^{\mathrm{g}}} \in \mathcal{V} \to \mathcal{V}$:

Theorem 5. $[\![i]\!]_\varphi^{\mathrm{g}} \gamma(v) \sqsubseteq \gamma([\![i]\!]_\varphi^{\alpha^{\mathrm{g}}} v)$

We give an intuition for how the abstract assignment transfer function now works by means of the following example:

Example 6. Let us consider the guarantee property $\Diamond(x = 3)$ and let us consider again, as in Example 5, the piecewise-defined ranking function with value $2x+1$ for $x \in [-\infty, 3]$ and undefined elsewhere, and the assignment $x := x + 1$. As in Example 5, the abstract assignment transfer function substitutes the variable x with the expression $x+1$ and increases the value of the function by one. Unlike Example 5, it also resets the value of the function for $x \in [3,3]$. The result is the piecewise-defined ranking function with value $2x+4$ for $x \in [-\infty, 2]$, 0 for $x \in [3,3]$ and undefined elsewhere. □

As before, the abstract transfer functions are combined together through backward analysis. The starting point is now the constant function equal to zero only for the environments that have the property φ, and undefined elsewhere, at the program final control point. The backward analysis computes an over-approximation of the value of the function $\tau_\varphi^{\mathrm{g}}$ and an *under-approximation* of its domain of definition $\mathrm{dom}(\tau_\varphi^{\mathrm{g}})$. In this way, an abstraction provides *sufficient preconditions* for the guarantee property $\Diamond\varphi$: if the abstraction is defined on a program state, then all the program execution traces branching from that state *eventually* reach a state with the property φ. Note that, when the property φ is $l_e : \text{true}$, where $l_e \in \mathcal{L}$ is the program final control point, we rediscover the backward termination analysis from Section 7.1.

7.3 Abstract Recurrence Semantics

In the following, we describe how to reuse the piecewise-defined ranking function abstract domains introduced in [21,22,23] and what changes are required in order to obtain decidable abstractions of the φ-recurrence semantics $\tau_\varphi^{\mathrm{r}}$ (11).

As before, we associate each program control point $l \in \mathcal{L}$ with a different ranking function $v \in \mathcal{V}$: $\tau_\varphi^{\mathrm{r}} \in \mathcal{L} \to (\mathcal{E} \rightharpoonup \mathbb{O})$. The transfer functions $[\![i]\!]_\varphi^{\mathrm{r}} \in (\mathcal{E} \rightharpoonup \mathbb{O}) \to (\mathcal{E} \rightharpoonup \mathbb{O})$ behave as the transfer functions for the guarantee semantics with the only difference that they *reset* the value of the ranking function for the environments that have the property φ *only if* all successors of the environments (by means of the program statement i) belong to the domain of the ranking function; hence, they ensure that each time φ is satisfied, it will be satisfied again in the future. As an example, the transfer function for an assignment $x := e$ is defined as follows:

$$[\![x := e]\!]^{\mathrm{r}} v \triangleq \lambda \rho. \begin{cases} 0 & \rho \models \varphi \wedge \forall z \in [\![e]\!]\rho \colon \rho[x \mapsto z] \in \mathrm{dom}(v) \\ \sup\{v(\rho[x \mapsto z]) + 1 \mid z \in [\![e]\!]\rho\} & \rho \not\models \varphi \wedge \forall z \in [\![e]\!]\rho \colon \rho[x \mapsto z] \in \mathrm{dom}(v) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Then, we define the abstract φ-recurrence semantics $\tau_\varphi^{\alpha^{\mathrm{r}}} \in \mathcal{L} \to \mathcal{V}$ by means of *sound* (with respect to $\sqsubseteq$) abstract transfer functions $[\![i]\!]_\varphi^{\alpha^{\mathrm{r}}} \in \mathcal{V} \to \mathcal{V}$:

Theorem 6. $[\![i]\!]_\varphi^{\mathrm{r}} \gamma(v) \sqsubseteq \gamma([\![i]\!]_\varphi^{\alpha^{\mathrm{r}}} v)$

We give an intuition for how the abstract assignment transfer function now works by means of the following example:

Example 7. Let us consider the guarantee property $\Box\Diamond(x = 3)$ and let us consider again, as in Example 6, the piecewise-defined ranking function with value $2x+1$ for $x \in [-\infty, 3]$ and undefined elsewhere, and the assignment $x := x + 1$. Unlike Example 6, the abstract assignment transfer function does not reset the value of the function for $x \in [3,3]$ because the ranking function is undefined for $x \in [4,4]$ (i.e., the successor of the environment $x \in [3,3]$ by means of the assignment $x := x + 1$). The result is the piecewise-defined ranking function with value $2x+4$ for $x \in [-\infty, 2]$, and undefined elsewhere.

Let us consider instead the piecewise-defined ranking function with value $2x+1$ for $x \in [-\infty, 4]$ and undefined elsewhere. The result of the assignment $x := x + 1$ is now the piecewise-defined ranking function with value $2x+4$ for $x \in [-\infty, 2]$, 0 for $x \in [3,3]$ and undefined elsewhere. □

Since the program final states cannot satisfy a recurrence property, the starting point of the recurrence backward analysis is now the totally undefined function at the program final control point. This function is then propagated backwards towards the program initial control point.

Note that, in case of a loop statement, according to the definition (10) of $\tau_\varphi^{\mathrm{r}}$ from Section 5, the transfer function involves a least fixpoint *nested* into a greatest fixpoint. Nested fixpoints are solved by iteration with the same widening operator used for termination [23] for the least fixpoint, and a new *dual widening* operator $\bar{\nabla}$

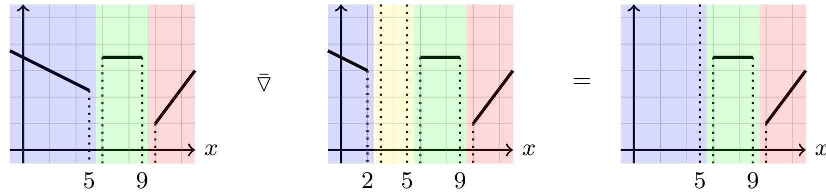

Fig. 3. Example of Dual Widening.

for the greatest fixpoint. The dual widening $\bar{\nabla}$ obeys (i) $\gamma(A) \sqcap \gamma(B) \sqsubseteq \gamma(A \mathbin{\bar{\nabla}} B)$, and (ii) for any sequence $(X_n)_{n \in \mathbb{N}}$, the sequence $Y_0 = X_0$, $Y_{n+1} = Y_n \mathbin{\bar{\nabla}} X_{n+1}$ stabilizes (i.e., $\exists i: \ Y_{i+1} = Y_i$). Dual widenings are rather unknown and, up to our knowledge, only few practical instance has been proposed (e.g., [5,18]). In our case, the dual widening $\bar{\nabla}$ enforces the termination of the analysis by preventing the set of pieces of a piecewise-defined ranking function from growing indefinitely: given two piecewise-defined ranking functions $v_1 \in \mathcal{V}$ and $v_2 \in \mathcal{V}$, it enforces the piecewise-definition of the first function v_1 on the second function v_2. Then, for each piece of the ranking functions, it maintains the value of the function only if both v_1 and v_2 are defined on that piece (cf. Figure 3).

The backward analysis computes an over-approximation of the value of the function $\tau_\varphi^{\mathrm{r}}$ and an *under-approximation* of its domain $\mathrm{dom}(\tau_\varphi^{\mathrm{r}})$. In this way, an abstraction provides *sufficient preconditions* for the recurrence property $\Box\Diamond\varphi$: if the abstraction is defined on a program state, then all the program execution traces branching from that state *always eventually* reach a state with the property φ.

8 Implementation

We have incorporated the static analysis methods for guarantee and recurrence temporal properties that we have presented into our prototype static analyzer FuncTion based on piecewise-defined ranking functions. It is available online[3].

The prototype accepts (non-deterministic) programs written in a C-like syntax and, when the guarantee or recurrence analysis methods are selected, it accepts state properties written as C-like pure expressions. It is written in OCaml and, at the time of writing, the available abstract domains to control the pieces of the ranking functions are based on intervals, octagons and convex polyhedra, and the available abstract domain to represent the value of the ranking functions is based on affine functions. The operators for the intervals, octagons and convex polyhedra abstract domains are provided by the APRON library [14]. It is also possible to activate the extension to ordinal-valued ranking functions [22] and tune the precision of the analysis by adjusting the widening delay.

The analysis proceeds by structural induction on the program syntax, iterating loops with widening (and, for recurrence properties, both widening and dual

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

```
¹c := 1;
while ²( true ) do
    ³x := c;
    while ⁴( x > 0 ) do ⁵x := x − 1; ⁶c := c + 1;
```

Fig. 4. Program COUNT-DOWN.

widening) until stabilization. In case of nested loops, the analysis stabilizes the inner loop for each iteration of the outer loop.

To illustrate the effectiveness of our new static analysis methods, we consider more examples besides the program SIMPLE of Section 1.

Example 8. Let us consider the program COUNT-DOWN in Figure 4 and the recurrence property $\Box\Diamond x{=}0$. At each iteration of the outer loop, the variable x takes the value of some counter c (which initially has value one); then, the inner loop decreases the value of x and increases the value of the counter c until x becomes less than or equal to zero. The recurrence property is clearly satisfied and indeed our prototype (parameterized by intervals and affine functions) is able to prove it: the piecewise-defined ranking function inferred at program control point 1 bounds the wait for the *next* occurrence of the desirable state $x{=}0$ by five program execution steps (i.e., executing the assignment $c := 1$, testing the outer loop condition, executing the assignment $x := c$, testing the inner loop condition and executing the assignment $x := x - 1$). The analysis infers a more interesting raking function associated to program control point 4:

$$f_4^r(x,c) = \begin{cases} 3c{+}2 & x < 0 \wedge c > 0 \\ 3 & x < 0 \wedge c = 0 \\ 1 & x = 0 \wedge c \geq 0 \\ 3x{-}1 & (x = 1 \wedge c \geq -1) \vee (x \geq 2 \wedge c \geq -2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

The function bounds the wait for the next occurrence of $x{=}0$ by $3c{+}2$ execution steps if $x{<}0 \wedge c{>}0$, by 3 execution steps if $x{<}0 \wedge c{=}0$ (i.e., testing the inner loop condition, testing the outer loop condition and executing the assignment $x := c$), by 1 execution step if $x{=}0 \wedge c{\geq}0$ (i.e., testing the inner loop condition) and by $3x{-}1$ execution steps if $(x{=}1 \wedge c{\geq}{-1}) \vee (x{\geq}2 \wedge c{\geq}{-2})$. In the last case there is a precision loss due to a lack of expressiveness of the intervals abstract domain: if x is strictly positive at program control point 4, the weakest precondition ensuring infinitely many occurrences of the desirable state $x = 0$ is $c \geq -x$ (which is not representable in the intervals abstract domain). □

```
while ¹( true ) do
    ²x := ?;
    while ³( x ≠ 0 ) do
        if ⁴( x > 0 ) then ⁵x := x − 1; else ⁶x := x + 1; fi
```

Fig. 5. Program SINK.

$flag_1 := 0;\ flag_2 := 0;$

```
⎡while ¹( true ) do                           ⎤   ⎡while ¹( true ) do                           ⎤
⎢    ²flag₁ := 1                              ⎥   ⎢    ²flag₂ := 1                              ⎥
⎢    ³turn := 2                               ⎥   ⎢    ³turn := 1                               ⎥
⎢    while ⁴( flag₂≠0 ∧ turn≠1 ) do           ⎥ ∥ ⎢    while ⁴( flag₁≠0 ∧ turn≠2 ) do           ⎥
⎢        BUSY_WAIT                            ⎥   ⎢        BUSY_WAIT                            ⎥
⎢    ⁵CRITICAL_SECTION                        ⎥   ⎢    ⁵CRITICAL_SECTION                        ⎥
⎣    ⁶flag₁ := 0                              ⎦   ⎣    ⁶flag₂ := 0                              ⎦
```

Fig. 6. Program PETERSON (Peterson's Algorithm).

Example 9. Let us consider the program SINK in Figure 5 and the recurrence property $\Box\Diamond x = 0$. At each iteration of the outer loop, the value of the variable x is reset by the non-deterministic assignment $x := ?$; then, the inner loop decreases or increases the value of x until it becomes equal to zero. Note that the program features *unbounded non-determinism* due to the assignment $x := ?$. The recurrence property is clearly satisfied, however the number of execution steps between two occurrences of the desirable state $x = 0$ is unbounded. Our prototype (parameterized by intervals and *ordinal-valued* ranking functions) is able to prove it as, at program control point 1, it finds a ranking function defined everywhere; its value is $\omega + 8$, meaning that the number of execution steps between two occurrences of the desirable state $x = 0$ is unbounded but *finite*. □

Example 10. Let us consider the program PETERSON, Peterson's algorithm for mutual exclusion, in Figure 6. Note that *weak fairness* assumptions are required in order to guarantee bounded bypass (i.e., a process cannot be bypassed by any other process in entering the critical section for more than a finite number of times). Since at the moment our prototype is not able to directly analyze concurrent programs, we have modeled the algorithm as a fair non-deterministic sequential program which interleaves execution steps from both processes while enforcing 1-bounded bypass (i.e., a process cannot be bypassed by any other process in entering the critical section for more than once). Our prototype is able to prove that both processes are allowed to enter their critical section infinitely often. □

These and additional examples are available from FuncTion web interface.

9 Related Work

In the recent past, a large body of work has been devoted to proving liveness properties of (concurrent) programs.

A successful approach for proving liveness properties is based on a transformation from model checking of liveness properties to model checking of *safety* properties [3]. The approach looks for and exploits lasso-shaped counterexamples. A similar search for lasso-shaped counterexamples has been used to generalize the model checking algorithm IC3 to deal with liveness properties [4]. However, in general, counterexamples to liveness properties in infinite-state systems are not necessarily lasso-shaped. Our approach is not counterexample-based and is meant for proving liveness properties directly, without reduction to safety properties.

In [20], Podelski and Rybalchenko present a method for the verification of liveness properties based on transition invariants [19]. The approach, as in [24], reduces the proof of a liveness properties to the proof of *fair termination* by means of a program transformation. It is at the basis of the industrial-scale tool Terminator [6]. By contrast, our method is meant for proving liveness properties directly, without reduction to termination. Moreover, it avoids the cost of explicit checking for the well-foundedness of the transition invariants.

A distinguishing aspect of our work is the use of infinite height abstract domains, equipped with (dual) widening. We are aware of only one other such work: in [16], Massé proposes a method for proving arbitrary temporal properties based on abstract domains for lower closure operators. A small analyzer is presented in [17] but the approach remains mainly theoretical. We believe that our framework, albeit less general, is more straightforward and of practical use.

An emerging trend focuses on proving *existential* temporal properties (e.g., proving that there exists a particular execution trace). The most recent approaches [2,7] are based on counterexample-guided abstraction refinement. Our work is designed for proving universal temporal properties (i.e., valid for all program execution traces). We leave proving existential temporal properties as part of our future work.

Finally, to our knowledge, the inference of sufficient preconditions for guarantee and recurrence program properties, and the ability to provide upper bounds on the wait before a program reaches a desirable state, is unique to our work.

10 Conclusion and Future Work

In this paper, we have presented an abstract interpretation framework for proving *guarantee* and *recurrence* temporal properties of programs. We have systematically derived by abstract interpretation new sound static analysis methods to effectively infer *sufficient preconditions* for these temporal properties, and to provide *upper bounds* on the wait before a program reaches a desirable state.

It remains for future work to express and handle fairness properties. We also plan to extend the present framework to the full hierarchy of temporal properties [15] and more generally to *arbitrary* (universal and existential) liveness properties.

References

1. 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.
2. T. A. Beyene, C. Popeea, and A. Rybalchenko. Solving Existentially Quantified Horn Clauses. In *CAV*, pages 869–882, 2013.
3. A. Biere, C. Artho, and V. Schuppan. Liveness Checking as Safety Checking. *Electronic Notes in Theoretical Computer Science*, 66(2):160–177, 2002.
4. A. R. Bradley, F. Somenzi, Z. Hassan, and Y. Zhang. An Incremental Approach to Model Checking Progress Properties. In *FMCAD*, pages 144–153, 2011.
5. A. Chakarov and S. Sankaranarayanan. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *SAS*, pages 85–100, 2014.
6. B. Cook, A. Gotsman, A. Podelski, A. Rybalchenko, and M. Y. Vardi. Proving that Programs Eventually do Something Good. In *POPL*, pages 265–276, 2007.
7. B. Cook and E. Koskinen. Reasoning About Nondeterminism in Programs. In *PLDI*, pages 219–230, 2013.
8. P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *ENTCS*, 6:77–102, 1997.
9. 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.
10. 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.
11. P. Cousot and R. Cousot. An Abstract Interpretation Framework for Termination. In *POPL*, pages 245–258, 2012.
12. R. W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.
13. N. Francez. *Fairness*. Springer, 1986.
14. B. Jeannet and A. Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *CAV*, pages 661–667, 2009.
15. Z. Manna and A. Pnueli. A Hierarchy of Temporal Properties. In *PODC*, pages 377–410, 1990.
16. D. Massé. Property Checking Driven Abstract Interpretation-Based Static Analysis. In *VMCAI*, pages 56–69, 2003.
17. D. Massé. Abstract Domains for Property Checking Driven Analysis of Temporal Properties. In *AMAST*, pages 349–363, 2004.
18. A. Miné. Inferring Sufficient Conditions with Backward Polyhedral Under-Approximations. In *NSAD*, volume 287 of *ENTCS*, pages 89–100, 2012.
19. A. Podelski and A. Rybalchenko. Transition Invariants. In *LICS*, pages 32–41, 2004.
20. A. Podelski and A. Rybalchenko. Transition Predicate Abstraction and Fair Termination. In *POPL*, pages 132–144, 2005.
21. C. Urban. The Abstract Domain of Segmented Ranking Functions. In *SAS*, pages 43–62, 2013.
22. C. Urban and A. Miné. An Abstract Domain to Infer Ordinal-Valued Ranking Functions. In *ESOP*, pages 412–431, 2014.
23. C. Urban and A. Miné. A Decision Tree Abstract Domain for Proving Conditional Termination. In *SAS*, 2014.
24. M. Y. Vardi. Verification of Concurrent Programs: The Automata-Theoretic Framework. *Annals of Pure and Applied Logic*, 51(1-2):79–98, 1991.