A

Linear Approximation of Continuous Systems with Trapezoid Step Functions

GIULIA COSTANTINI, University Ca' Foscari of Venice
PIETRO FERRARA, ETH Zurich, Switzerland
and AGOSTINO CORTESI, University Ca' Foscari of Venice

The Interval domain, which approximates the values of a numerical variable with upper and lower bounds, is too rough when dealing with continuous values (e.g., the continuous physical inputs of embedded systems). In this scenario, a static analyzer has to take into account how values evolve during the time. The Interval domain has been already extended to this context through the Interval Valued Step Functions (IVSF) domain, which is a sound approximation of continuous functions. IVSF abstracts the values of a variable in a slot of time with constant upper and lower bounds.

In this paper, we introduce the Trapezoid Step Functions (TSF) domain. Our main insight is to approximate $\mathcal{C}^2_+$ functions by a finite sequence of trapezoids, one for each slot of time, adopting linear functions (instead of constant intervals) to abstract the upper and the lower bounds of a continuous variable in each slot of time. We formalize the lattice structure of TSF, and how to build and compute a sound abstraction of a given continuous function. Finally, we apply TSF to some case studies comparing our results with IVSF. The experimental results underline the effectiveness of the approach in terms of both precision and efficiency.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; G.1.2 [**Numerical Analysis**]: Approximation

General Terms: Languages, Reliability, Theory, Verification

Additional Key Words and Phrases: Abstract interpretation, static analysis, embedded software, continuous functions

1. INTRODUCTION

Embedded software is made up of discrete (that is, the program) and continuous (that is, the physical environment) components. The program receives inputs from the physical environment through sensors that are usually modeled by volatile variables. Embedded software often deals with safety critical systems, like flight controllers. The reliability of these systems is crucial: even a single bug can result in a disaster, and this is a relevant challenge for formal verification methods. Static analysis is aimed at over-approximating the concrete semantics of a program in a computable way to prove properties that are satisfied by all possible executions. In the context of embedded software, safety properties are particularly important, and it is necessary to precisely approximate both the discrete and the continuous system.

Work partially supported by RAS project "TESLA: Techniques for Enforcing Security in Languages and Applications".
Authors' addresses: G. Costantini and A. Cortesi, Ca' Foscari University, Dipartimento di Informatica, via Torino 172, Mestre - Venice, Italy; P. Ferrara, ETH Zurich, Chair of Programming Methodology, Universitatstr. 6, 8092 Zurich, Switzerland.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

On the one hand, there is a large literature on the static analysis of discrete programs. On the other hand, these approaches do not perform well when they are applied to continuous environments. For instance, in the context of the abstract interpretation framework [Cousot and Cousot 1977; 1979], the Interval domain [Cousot and Cousot 1977] abstracts continuous systems with the minimal and the maximal values a sensor can return at any time. This approach is definitely too rough for the static analysis of embedded programs. Bouissou and Martel [Bouissou and Martel 2008] proposed the Interval Valued Step Functions (IVSF) domain for approximating the behavior of a function in a given interval of time (i.e, a step) with the minimal and the maximal values the function could achieve during that period of time.

Contribution: In this paper, we go one step further by introducing the Trapezoid Step Functions (TSF) domain. TSF abstracts the values of a function in a given slot of time with two linear functions, tracking in this way linear relationships between the time and the output value. Our work is inspired by the evolution of numerical domains. Since the Interval domain is not precise enough in many contexts, various relational domains (e.g., Polyhedra [Cousot and Halbwachs 1978] and Octagons [Miné 2006]) have been introduced. In a similar way, our domain, instead of tracking the minimal and the maximal values of a function in a given slot of time, tracks two linear functions that approximate values of the function as a linear multivalued function [Aubin and Cellina 1984; Bressan and Cortesi 1989]. The two linear functions, together with the two vertical lines that delimit the time slot, form a trapezoid. We approximate the function with a finite number of trapezoids, one for each step.

Consider, for instance, Figure 1. It compares the 4-steps abstraction of $f = sin(x)$ by TSF (on the left) and by IVSF (on the right) in the interval $[0, 2\pi]$. On the one hand, these plots make clear that in general TSF better approximates the shape of the function using trapezoids rather than the rectangles obtained with IVSF. On the other hand, IVSF gives more precise bounds on the maximum and minimum values assumed by the function. Therefore, TSF could be used in combination with IVSF to improve the precision of the overall analysis, and in particular to precisely bound the minimal and maximal values of the function. For instance, the application of the Cartesian product [Cousot and Cousot 1977] TSF $\times$ IVSF to the example of Figure 1 discovers that, at $\frac{\pi}{2}$, the abstracted function has exactly value 1, since (i) TSF tracks that its minimal value is 1, and (ii) IVSF tracks that its maximal value is 1.

In Figure 2 we can see the abstraction of the same function ($\sin(x)$) with the same number of steps (4) on a restricted domain ($[0, \frac{\pi}{2}]$). This scenario is quite different from Figure 1, but also in this case the plots show that TSF is quite more precise than IVSF.

The main contributions of this paper are (i) the formal definition of TSF and its lattice operators, (ii) the introduction of a sound abstraction function that, given a con-

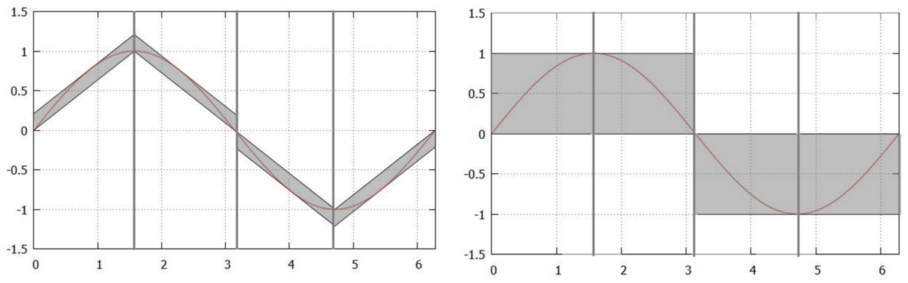


Fig. 1. TSF (left) and IVSF (right) abstractions of $\sin(x)$, with 4 steps, on the domain $[0, 2\pi]$

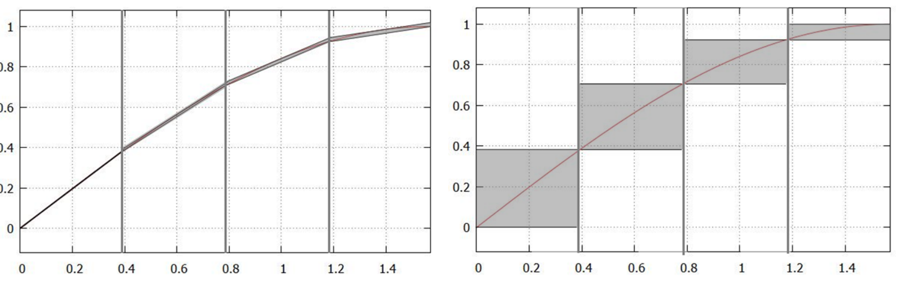


Fig. 2. TSF (left) and IVSF (right) abstractions of $\sin(x)$, with 4 steps, on the domain $[0, \frac{\pi}{2}]$

```
volatile float x;
static float intgrx=0.0, h=1.0/8;
void main() {
  while(true) { //assume frequency=8KHz
    float xi=x;
    intgrx+=xi*h;
  }
}
```

Fig. 3. Simple integrator

tinuous and derivable function, builds up its abstraction in TSF, and (iii) the discussion of some experimental results and the comparison with the ones obtained by IVSF.

The paper is structured as follows. The rest of this Section introduces a motivating example and recalls some basic concepts of abstract interpretation. Sections 2 and 3 formalize the domain and the abstraction function. In Section 4 we present some experimental results when applying TSF to the abstraction of different functions, and how our results compare with IVSF. Section 5 discusses the related work and Section 6 concludes.

1.1. Motivating Example

Our motivating example regards a special case of hybrid system, where we have a discrete system (an embedded program) which takes a continuous environment as input.

Consider the program in Figure 3. This is the code of an integrator, a quite common component of embedded programs. It has been inspired by [Goubault et al. 2006], and it is the example used in [Bouissou and Martel 2008] in order to show the main features of IVSF. This code integrates a function (whose values are provided through the `volatile` variable `x`) using the rectangle method on a sampling step `h`. We assume that the function we integrate is $\sin(2\pi t)$, and that the input data are given by a sensor (hence the `volatile` variable `x`) at a frequency of 8KHz. This scenario is particularly interesting for the analysis of numerical precision, since the sensor will produce the sequence of values $[0, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 0, -\frac{\sqrt{2}}{2}, -1, -\frac{\sqrt{2}}{2}]$ on `x`. Therefore, in a perfect arithmetic computation the summation of these values multiplied by `h` will be equal to zero after $8 \times i$ iterations $\forall i \in \mathbb{N}$. Nevertheless, in a real system this summation would produce some approximate values because of floating point approximation. This code is particularly interesting to test the precision of abstract domains since it propagates the approximation error of our abstract domain at each iteration of the `while` loop, and therefore it is a good candidate to test the precision of TSF.

1.2. Abstract Interpretation

Abstract interpretation [Cousot and Cousot 1977; 1979] is a framework to define and prove the soundness of abstractions. The concrete domain formalizes the information we want to approximate, while the abstract domain which approximated information we track. Usually, concrete states are composed by sets of elements (e.g., all the possible computational states, or a set of continuous functions), that are approximated by a unique element (also referred to as an *abstract state*) in the abstract domain. Formally, the concrete domain $\wp(\mathsf{D})$ forms a complete lattice $\langle \wp(\mathsf{D}), \subseteq, \emptyset, \mathsf{D}, \cup, \cap \rangle$. Similarly, the abstract domain $\overline{\mathsf{A}}$ has to form a complete lattice $\langle \overline{\mathsf{A}}, \leq_{\overline{\mathsf{A}}}, \perp_{\overline{\mathsf{A}}}, \top_{\overline{\mathsf{A}}}, \sqcup_{\overline{\mathsf{A}}}, \sqcap_{\overline{\mathsf{A}}} \rangle$ as well. The concrete and abstract domains are related by a concretization $\gamma_{\overline{\mathsf{A}}}$ and an abstraction $\alpha_{\overline{\mathsf{A}}}$ functions. The abstract domain is a sound approximation of the concrete domain if $\gamma_{\overline{\mathsf{A}}}$ and $\alpha_{\overline{\mathsf{A}}}$ form a Galois connection. When abstract domains do not satisfy the ascending chain condition, a widening operator $\nabla_{\overline{\mathsf{A}}}$ is required in order to guarantee the convergence of the fixed point computation.

2. THE TRAPEZOID STEP FUNCTIONS DOMAIN (TSF)

In this Section, we first introduce the concrete domain. Then we formalize the lattice structure of our abstract domain. Finally, we introduce a widening operator that is necessary for ensuring the convergence of the analysis on this domain.

In this way, we provide a complete definition of an abstract domain that can be used not only to abstract single functions (as we did in the experimental results), but also to abstract set of functions (e.g., to take into account some rounding approximations).

2.1. Concrete Domain

The concrete domain $\mathcal{D}$ is defined as the powerset of continuous functions in $\mathbb{R}^+ \rightarrow \mathbb{R}$ which have two continuous derivatives (i.e., the set $\mathcal{C}^2_+$):

$$\mathcal{D} = \wp(\mathcal{C}^2_+)$$

Since in the scenario of embedded programs the input variable represents the time, the functions' domain is $\mathbb{R}^+$ instead of $\mathbb{R}$.

2.2. Abstract domain elements

Let us first introduce the key-idea behind our domain. Given a function f and a set of ordered indices $\{t_i\}_{0 \leq i \leq N}$, we approximate the values of f in a step $[t_i, t_{i+1}]$ by a trapezoid whose (i) two parallel sides are vertical, in correspondence of t_i and t_{i+1}, and (ii) the other two sides are in the form $f^-(x) = m^- x + q^-$ and $f^+(x) = m^+ x + q^+$ and approximate lower and upper values of f inside $[t_i, t_{i+1}]$. For instance, Figure 4 depicts a trapezoid defined on the step $[0, 3]$ with $f^-(t) = 0.33t + 1$ and $f^+(t) = -0.17t + 3.5$ as lower and upper sides, respectively.

Formally, given a step $[t_i, t_{i+1}]$, a single trapezoid is defined by two linear functions, and each of these two functions is defined by two real numbers representing the slope and intercept. Therefore, the pair of sides of each trapezoid are defined by a tuple $\mathrm{v} = (m^-, q^-, m^+, q^+)$, where $m^-, q^-, m^+, q^+ \in \mathbb{R}$ represents the two lines $f^-(x) = m^- x + q^-$ and $f^+(x) = m^+ x + q^+$. We denote by f^- and f^+ the lower and the upper side, respectively. TSF can be seen as a generalization of IVSF, whose two horizontal sides are parallel, i.e., with $f^+(x) = q^+$ and $f^-(x) = q^-$.

Following the standard notation on step functions [Bouissou and Martel 2008], consider a generic set V of values. These values could be simple numbers (integer, reals, ...) but also more complicated structures like intervals or tuples. In our domain, a value will be composed by two lines, the upper (f^+) and the lower (f^-) side of trapezoids. Then, we represent a step function from time to V as a conjunction of constraints

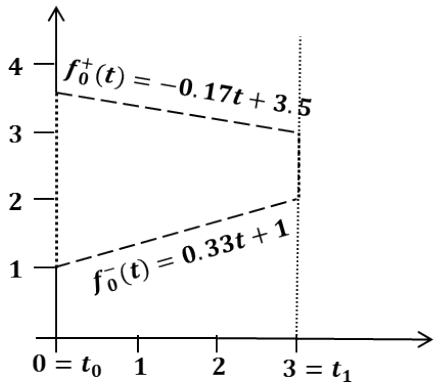


Fig. 4. Example of a trapezoid defined on $[0, 3]$

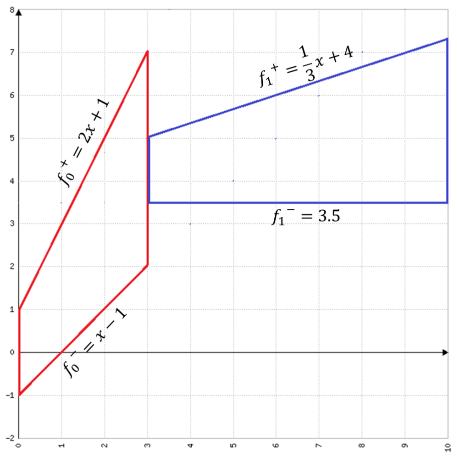


Fig. 5. A TSF abstract element on the domain $[0, 10]$

of the form "$t_i : \mathbf{v_i}$" such that $t_i \in \mathbb{R}^+ \wedge \mathbf{v_i} \in V$. This means that the step function switches to $\mathbf{v_i}$ at time t_i. For $t \in [t_i, t_{i+1}]$ the function respects the constraints represented by value $\mathbf{v_i}$. When $t = t_{i+1}$, the abstract value of the function switches "abruptly" to $\mathbf{v_{i+1}}$. We consider only finite conjunctions of constraints, otherwise the abstract operations of our domain would not be computable. A finite sequence of constraints $f = t_0 : \mathbf{v_0} \wedge t_1 : \mathbf{v_1} \wedge \cdots \wedge t_N : \mathbf{v_N}$ represents the step function f such that $\forall t \in \mathbb{R}^+ : f(t) = \mathbf{v_i}$ with $i = \max(\{j \in [0, N] : t_j \leq t\})$. We use the compact notation $f = \bigwedge_{0 \leq i \leq N} t_i : \mathbf{v_i}$, with $N \in \mathbb{N} \wedge \forall i \in [0, N] : (t_i \in \mathbb{R}^+ \wedge \mathbf{v_i} \in V)$. V is the set of tuples $\{(m^-, q^-, m^+, q^+) : m^-, q^-, m^+, q^+ \in \mathbb{R}\}$. To lighten up the notation, a quadruple $\mathbf{v_i} = (m_i^-, q_i^-, m_i^+, q_i^+)$ denotes $f_i^-(t) = m_i^- t + q_i^-$ and $f_i^+(t) = m_i^+ t + q_i^+$. We will alternatively denote a step value as $\mathbf{v_i} = (m_i^-, q_i^-, m_i^+, q_i^+)$ or $\mathbf{v_i} = (f_i^-, f_i^+)$.

For example, the step function h with two steps $t_0 = 0, t_1 = 3$ with values, respectively, $v_0 = (1, -1, 2, 1), v_1 = (0, 3.5, \frac{1}{3}, 4)$ will be written as $h = (0 : [1, -1, 2, 1]) \wedge (3 : [0, 3.5, \frac{1}{3}, 4])$. The graphical representation of h is depicted in Figure 5, where we can see the two trapezoids composing the step function. On the upper and lower sides of the trapezoids we reported the equations of the corresponding lines.

Normal form and equivalence relation. However, different abstract values may represent exactly the same step function. For example, the conjunctions $(0 : [0, 0, 1, 1]) \wedge (4 : [0, 0, 1, 1])$ and $(0 : [0, 0, 1, 1]) \wedge (7 : [0, 0, 1, 1])$ define the same step function which, for every input $t \in [0, +\infty)$, returns as output value the interval $[0, t+1]$. To avoid that, we use the same notion of normal form defined in [Bouissou and Martel 2008]:

(1) the switching times t_i of a conjunction are sorted and different (if $f = \bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{v_i}\}$ then $0 = t_0 < t_1 < \cdots < t_n < \cdots < t_N$);
(2) two consecutive constraints cannot have equal values (each $\mathbf{v_i}$ must be different from $\mathbf{v_{i+1}}$, that is $\forall i \in [0, N-1], \mathbf{v_i} \neq \mathbf{v_{i+1}}$)

With these conditions, the representation is unique. We will denote by $Norm$ the normalization procedure. The algorithm to compute the normalized form $Norm(f)$ of a given conjunction of constraints f is defined as follows. First we sort the constraints by ascending switching times, with the convention that if two constraints have the same time, then we only keep the one with the highest index. In this way, we fulfill the first normalization condition. Then we remove any constraint $t_i : \mathbf{v_i}$ such that $\mathbf{v_{i-1}} = \mathbf{v_i}$. In this way. we fulfill the second normalization condition. Note that the normalization process does not change the meaning of the representation: for a conjunction f, it holds that $\forall t \in \mathbb{R}^+ f(t) = Norm(f)(t)$. In our previous example, we would obtain a representation with a single constraint $0 : [0, 0, 1, 1]$.

Given two normalized conjunctions, we define the same equality test as in [Bouissou and Martel 2008]:

$$\bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{v_i}\} = \bigwedge_{0 \leq j \leq M} \{u_i : \mathbf{w_i}\} \Leftrightarrow N = M \wedge \forall i \in [0, N], t_i = u_i \wedge \mathbf{v_i} = \mathbf{w_i}$$

The normalization process induces an equivalence relation $\equiv$ defined by $f \equiv g \Leftrightarrow Norm(f) = Norm(g)$.

Constraints. We impose two constraints on abstract elements in order to be valid:

$$\forall i \in [0, N], f_i^-(t_i) \leq f_i^+(t_i) \wedge f_i^-(t_{i+1}) \leq f_i^+(t_{i+1}) \quad (1)$$
$$\forall i \in [0, N-1], [f_i^-(t_{i+1}), f_i^+(t_{i+1})] \cap [f_{i+1}^-(t_{i+1}), f_{i+1}^+(t_{i+1})] \neq \emptyset \quad (2)$$

(1) states that, at each step $[t_i, t_{i+1}]$ of the function, the two lines f_i^+ and f_i^- do not intersect. This assumption is not restrictive, since we can always split a step with intersecting sides into two smaller steps with not intersecting sides through the *refine* operator defined in Section 2.6.

(2) states that two consecutive steps $[t_i, t_{i+1}], [t_{i+1}, t_{i+2}]$ of a step function must have at least one point in common at t_{i+1}. This means that the interval of values at $t = t_{i+1}$ individuated by $\mathbf{v_i}$ and the one individuated by $\mathbf{v_{i+1}}$ at the same input t must have a non-empty intersection. This constraint is needed because otherwise the concrete functions represented by the abstract element would not be continuous, and it would be abstracted by a bottom value in our domain since we abstract functions in $\mathcal{C}_+^2$. In Figure 6 we can see an example of an abstract state which violates this constraint. In particular, at $t = 3$ we can see that the upper side of the left trapezoid passes for $y = 3$, while the lower side passes for $y = 2$, so the interval of values individuated by the left trapezoid is $[2, 3]$. The interval individuated by the right trapezoid, on the other hand, is $[3.5, 5]$. Since $[2, 3] \cap [3.5, 5] = \emptyset$, this abstract state is not valid and it collapses to bottom. This is a sound approximation since there is no continuous function that may satisfy this constraint.

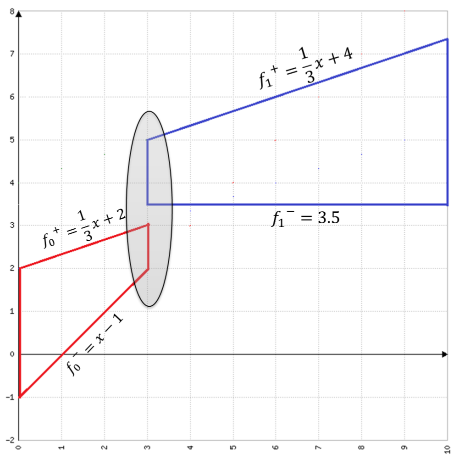


Fig. 6. A TSF abstract element on the domain $[0, 10]$ which violates the second validity constraint at $t = 3$

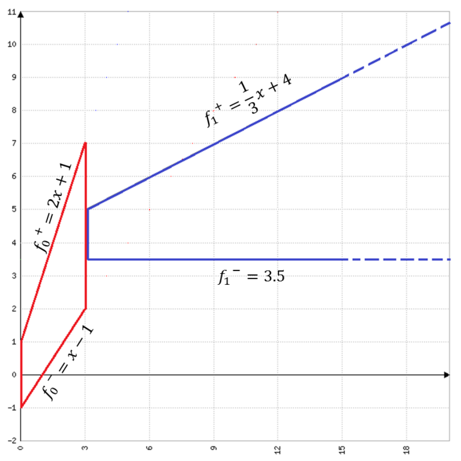


Fig. 7. A TSF abstract element on the entire domain $\mathbb{R}^+$

2.2.1. Abstract domain elements. The elements (that is, states) of our abstract domain, denoted by $D^\sharp$, are normalized finite conjunctions of constraints $f = \bigwedge_{0 \le i \le N} t_i : \mathbf{v_i}$ (with $N \in \mathbb{N} \wedge \forall i \in [0, N] : (t_i \in \mathbb{R}^+ \wedge \mathbf{v_i} \in V)$) which satisfy the equations (1) and (2).

Note that, if it is not specified otherwise, the abstract states of TSF refer to the entire domain $\mathbb{R}^+$. Since the conjunctions we consider are finite, this means that the last trapezoid lacks its right side (it is not closed): the upper and lower sides extend (on the right) to infinite. We can see an example in Figure 7, where we depict the same abstract state of Figure 5 but with an unrestricted domain. In Figure 5 instead, the domain was restricted to $[0, 10]$ and so the trapezoid on the right was closed.

2.3. Concretization function

The abstract step function f defined by $\bigwedge_{0 \le i \le N} \{t_i : \mathbf{v_i}\}$, where $\mathbf{v_i} = (f_i^-, f_i^+) = (m_i^-, q_i^-, m_i^+, q_i^+)$, represents the set of continuous, differentiable functions that are bounded by the lines $f_i^-(t) = m_i^- t + q_i^-$ and $f_i^+(t) = m_i^+ t + q_i^+$ for any time $t \in [t_i, t_{i+1}]$.

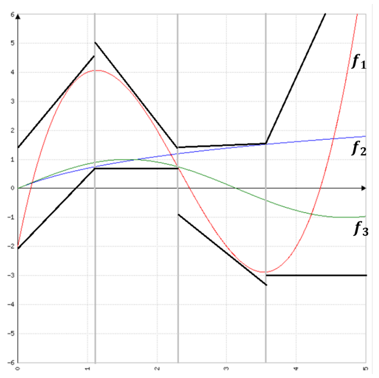


Fig. 8. Concretization function

The concretization function γ is thus defined by:

$$\gamma(\bigwedge_{0\leq i\leq N}\{t_i : \mathbf{v_i}\}) = \{g \in C^2_+ | \forall i \in [0,N], \forall t \in [t_i, t_{i+1}], g(t) \in [f_i^-(t), f_i^+(t)]\}$$

where t_{N+1} is either $+\infty$ if $dom(f) = \mathbb{R}^+$, or k if $dom(f) = [0,k]$, with k constant.

Figure 8 depicts an example of an abstract state defined on the domain $[0,5]$ with 4 steps (note that here $t_{N+1} = 5$). In this Figure we can see three possible concrete functions ($f_1 = x^3 - 7x^2 + 12x - 2$, $f_2 = ln(x+1)$ and $f_3 = sin(x)$) that are all approximated by such abstract state.

2.4. Abstraction function

The definition of an abstraction is not as direct as the concretization. As in the case of the polyhedral domain [Cousot and Halbwachs 1978], we cannot define the best one: it is always possible to increase the quality of the abstraction by increasing the number of steps. Thus, for now, we only give a criterion (the same as [Bouissou and Martel 2008]) for a function α to be a safe abstraction (later, in Section 3, we will also give the definition of a sound abstraction function). Let us first define the lower- and upper-functions for a given set of continuous real functions. Let $\mathcal{Y}$ be a continuous function ($\mathcal{Y} \in \mathcal{D}$). We define two functions $\underline{\mathcal{Y}}$ and $\overline{\mathcal{Y}}$ to be the inf- and sup-functions of $\mathcal{Y}$: $\underline{\mathcal{Y}} = \lambda t.\mathit{inf}\{y(t) : y \in \mathcal{Y}\}$ and $\overline{\mathcal{Y}} = \lambda t.\mathit{sup}\{y(t) : y \in \mathcal{Y}\}$. Equivalently we define the lower- and upper-functions of a Trapezoid Step Function. Let f be $\in \mathcal{D}^\sharp$, the real-valued step functions $\underline{f}$ and $\overline{f}$ are $\underline{f} = \lambda t.f_i^-(t)$ and $\overline{f} = \lambda t.f_i^+(t)$ where, in both cases, $i = \max(\{j \in \mathbb{N} : t_j \leq t\})$. These four functions are used to define the *Validity Condition*.

LEMMA 2.1 (VALIDITY CONDITION). *A function $\alpha : \mathcal{D} \rightarrow \mathcal{D}^\sharp$ satisfies the Validity Condition (V.C.) if and only if for all $\mathcal{Y} \in \mathcal{D}$, it holds that:*

$$\forall t \in \mathbb{R}^+, \underline{\alpha(\mathcal{Y})}(t) \leq \underline{\mathcal{Y}}(t) \leq \overline{\mathcal{Y}}(t) \leq \overline{\alpha(\mathcal{Y})}(t)$$

This property states that the computed Trapezoid Step Function indeed encloses the set $\{y(t) : y \in \mathcal{Y}\}$ for all $t \in \mathbb{R}^+$. The V.C. is a necessary and sufficient condition for the abstraction α to be sound.

We now formulate the theorem that guarantees the soundness of the abstraction.

THEOREM 2.2. *If α satisfies the V.C., then for every $\mathcal{Y} \in \mathcal{D}$, $\mathcal{Y} \subseteq \gamma(\alpha(\mathcal{Y}))$.*

PROOF. Let $\mathcal{Y}$ be $\in \mathcal{D}$ and $f = \alpha(\mathcal{Y})$ be $\in \mathcal{D}^\sharp$. We have to prove that $\mathcal{Y} \subseteq \gamma(f)$. Since α satisfies the V.C., we know that $\forall t \in \mathbb{R}^+, \forall y \in \mathcal{Y}, y(t) \in f(t)$. Let us take a $y \in \mathcal{Y}$. y

is a continuous function that verifies $\forall t \in \mathbb{R}^+, y(t) \in f(t)$, thus $y \in \gamma(f)$. Therefore, we have that $\mathcal{Y} \subseteq \gamma(f)$. $\square$

2.5. Partial order

The partial order $\subseteq^\sharp$ on two functions f, g in $D^\sharp$ is defined pointwisely, that is, for all possible inputs t we check that the set of values assumed by f in that point is a subset of the set of values assumed by g at the same point. Formally, $f \subseteq^\sharp g \Leftrightarrow \forall t \in \mathbb{R}_+ : f(t) \subseteq g(t)$, where $f(t) = \{v : f_j^-(t) \leq v \leq f_j^+(t) \wedge t \in [t_j, t_{j+1}]\}$ and the same holds for $g(t)$.

To define the partial order on step functions, we first define a partial order on single steps. Let $\mathbf{v_i} = (f_i^-, f_i^+)$ and $\mathbf{w_j} = (g_j^-, g_j^+)$ be the values of two steps on the same domain $[a, b]$. Then:

$$\begin{aligned}\mathbf{v_i} \sqsubseteq_{[a,b]} \mathbf{w_j} &\Leftrightarrow \forall t \in [a, b] : f_i^-(t) \geq g_j^-(t) \wedge f_i^+(t) \leq g_j^+(t)\\ &\Leftrightarrow \forall t \in [a, b] : [f_i^-(t), f_i^+(t)] \subseteq [g_j^-(t), g_j^+(t)]\\ &\Leftrightarrow [f_i^-(a), f_i^+(a)] \subseteq [g_j^-(a), g_j^+(a)] \wedge [f_i^-(b), f_i^+(b)] \subseteq [g_j^-(b), g_j^+(b)]\end{aligned} \tag{3}$$

In other words, $\mathbf{v_i}$ is smaller than $\mathbf{w_j}$ if the area of the trapezoid identified by $\mathbf{v_i}$ (in the domain $[a, b]$) is contained in the area of the trapezoid identified by $\mathbf{w_j}$ (in $[a, b]$ as well). To do this, we have to compare only the upper and lower sides of the trapezoids. Formally, we should check that $\forall t \in [a, b] : (f_i^- \geq g_j^- \wedge f_i^+ \leq g_j^+)$. Since the sides are defined by straight lines and they do not intersect, this results into checking only the values of such lines at the left and right extremes of the trapezoid.

Now we can give an effective condition for testing whether $f \subseteq^\sharp g$. Let $f = \bigwedge_{0 \leq i \leq N}\{t_i : \mathbf{v_i}\}$ and $g = \bigwedge_{0 \leq j \leq M}\{u_j : \mathbf{w_j}\}$ be two abstract states, then:

$$\begin{gathered} f \subseteq^\sharp g \\ \Updownarrow \\ \forall (i, j) \in [0, N] \times [0, M] : [a, b] = [t_i, t_{i+1}] \cap [u_j, u_{j+1}] \neq \emptyset \Rightarrow \mathbf{v_i} \sqsubseteq_{[a,b]} \mathbf{w_j} \end{gathered} \tag{4}$$

The condition 4 considers all intersections between steps from the two abstract states, that is, all pairs of steps (t_i, u_j) from f and g that have a non-empty intersection ($[t_i, t_{i+1}] \cap [u_j, u_{j+1}] \neq \emptyset$). To check if two steps have an intersection ($[t_i, t_{i+1}] \cap [u_j, u_{j+1}] \neq \emptyset$), we can use the condition $u_j \leq t_{i+1} \wedge u_{j+1} \geq t_i$. Moreover, if $u_j \leq t_i$ we have that $[a, b] = [t_i, u_{j+1}]$. Otherwise, we have that $[a, b] = [u_j, t_{i+1}]$. For each intersection, we then compare the two values ($\mathbf{v_i}, \mathbf{w_j}$) in $[a, b]$.

If each step value of f is smaller than the value of every intersected step of g (with respect to their intersection on the domain), then $f \subseteq^\sharp g$.

LEMMA 2.3 (SOUNDNESS OF THE PARTIAL ORDER). *If $f, g \in D^\sharp$ are normalized, then*

$$f \subseteq^\sharp g \Leftrightarrow \forall t \in \mathbb{R}_+, f(t) \subseteq g(t)$$

PROOF. We distinguish the two directions:

— ($\Rightarrow$) Let $f = \bigwedge_{0 \leq i \leq N}\{t_i : \mathbf{v_i}\}$ and $g = \bigwedge_{0 \leq j \leq M}\{u_j : \mathbf{w_j}\}$, be such that $f \subseteq^\sharp g$, and let t be $\in \mathbb{R}_+$. Then there exists $i \in [0, N]$ and $j \in [0, M]$ such that $t \in [t_i, t_{i+1}] \wedge t \in [u_j, u_{j+1}]$. Thus, $[t_i, t_{i+1}] \cap [u_j, u_{j+1}] \neq \emptyset$, so $f(t) = [f_i^-(t), f_i^+(t)] \subseteq [g_j^-(t), g_j^+(t)] = g(t)$ by definition of $\sqsubseteq_{[a,b]}$.

— ($\Leftarrow$) Let $f, g \in D^\sharp$ be such that $\forall t \in \mathbb{R}_+, f(t) \subseteq g(t)$. Let $i, j \in [0, N] \times [0, M]$ be such that $[a, b] = [t_i, t_{i+1}] \cap [u_j, u_{j+1}] \neq \emptyset$, and let x be $\in [a, b]$. Since $\forall t \in \mathbb{R}_+, f(t) \subseteq g(t)$, then we also have $\forall x \in [a, b] : f(x) \subseteq g(x)$, that is $\forall x \in [a, b] : ([f_i^-(x), f_i^+(x)] \subseteq$

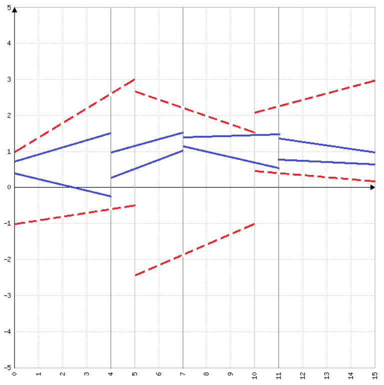


Fig. 9. Partial order

$[g_j^-(x), g_j^+(x)])$. Then 3 we that that $\mathbf{v_i} \sqsubseteq_{[a,b]} \mathbf{w_j}$ for each possible pair (i, j) by the second line of Equation, and $f \subseteq^\sharp g$ by Equation 4.

□

We can see an example of the partial order relationship between two abstract states f and g on the domain $[0, 15]$ in Figure 9. The abstract state f is represented with straight lines, the abstract state g with dashed lines. f is defined on the set of steps $\{0, 5, 10\}$ while g is defined on the the set of steps $\{0, 4, 7, 11\}$. Then the comparison is made on the following intervals: $[0, 4]$, $[4, 5]$, $[5, 7]$, $[7, 10]$, $[10, 11]$, $[11, 15]$. Since in each of these intervals we have that the straight lines lie within the area defined by the dashed lines, we obtain that $f \leq g$.

Top and bottom. The top element of the domain is defined by $\top^\sharp = 0 : [0, -\infty, 0, \infty]$ (that is, the step function with only one step with value $\mathbb{R}$), while $\bot^\sharp$ is a special element such that $\gamma(\bot^\sharp) = \emptyset \wedge \forall f \in D^\sharp, \bot^\sharp \subseteq^\sharp f$.

2.6. *Refine* operator

We define a *refine* operator, which, given an abstract state of TSF and a set of indices, adds these indices to the step list of the state. The concretization of the abstract state remains the same after the application of a *refine* operator, since the values $\mathbf{v_i}$ are not modified. This operator will be useful to make two abstract states directly comparable, by making them defined on the same set of steps.

Consider an abstract state $f = \bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{v_i}\}$ where $T = \{t_i : 0 \leq i \leq N\}$ and a set of indices $U = \{u_j : 0 \leq j \leq M\}$. Let $S = \{s_k : \forall k \in [0, P], (s_k \in (T \cup U) \wedge s_k < s_{k+1})\}$ be the set of all the indices contained in T and U, ordered and without repetitions (therefore $P = N + M - |T \cap U|$). The *refine* operator on this state is defined by:

$$Refine(f, U) = \bigwedge_{0 \leq k \leq P} \{s_k : \widehat{\mathbf{v_k}}\}$$

where $\widehat{\mathbf{v_k}} = \mathbf{v}_{\mathbf{max\{i : t_i \leq s_k\}}}$. Intuitively, you can see that this operator does not change the abstract information of the state. In fact we can enunciate the following lemma:

LEMMA 2.4 (REFINEMENT). *Given a normalized abstract state f, for any set of indices S, it holds that $f \equiv Refine(f, S)$.*

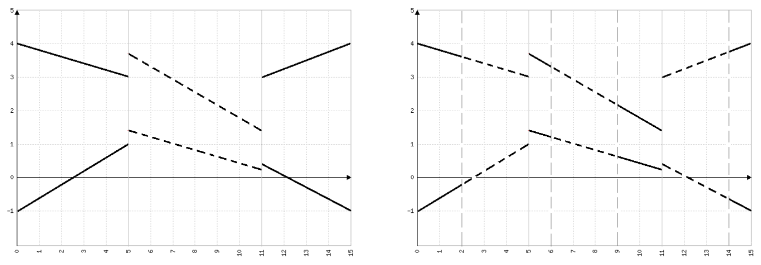


Fig. 10. Refine operator

PROOF. The equivalence relation $\equiv$ considers the normalized version of the two elements being compared. We know by hypothesis that f is already in a normalized form, so we have to compare f with $Norm(Refine(f, S))$.

Suppose that f is defined on the indices set T, that is, $\forall t_i \in T$, the step function f has a step at t_i. To underline this relationship, we write f_T instead of f. Let $Refine(f_T, S) = f'_R$ be the refined version of f_T, where $R = T \cup S$. This means that the result of the *refine* operator is a step function f' defined on the indices set $R = T \cup S$. So we can write $f_T = \bigwedge_{i\in[0,N-1]}\{t_i : \mathbf{v_i}\}$ (supposing $|T| = N$) and $f'_R = \bigwedge_{j\in[0,M-1]}\{r_j : \widehat{\mathbf{v_j}}\}$ (supposing $|R| = M$).

By definition of *refine*, we know that $\forall r_j \in (S \setminus T) \subseteq R$ the value of the step r_j in f'_R is $\widehat{\mathbf{v_j}} = \mathbf{v_{\max\{i:t_i \le r_j\}}}$. So, the value of the step r_j is the same as the one of a consecutive step already present in f_T. The normalization process removes any step which has the same value of the previous one. Then each step $r_j \in (S \setminus T)$ is removed and the result of the normalization of f'_R is a step function f''_T with step indices only from the set T (since $(T \cup S) \setminus (S \setminus T) = T$). Moreover, the value associated to each step t_i of f''_T is $\mathbf{v_i}$, the same as in f_T, since neither the *refine* operator nor the normalization process change any step value. Then f_T and f''_T are defined on the same indices set, and they have the same step values. We can conclude that $f_T = f''_T = Norm(f'_R) = Norm(Refine(f_T, S))$, thus $f = Norm(Refine(f, S))$ and finally, since $f = Norm(f)$, we have $f \equiv Refine(f, S)$ by definition of $\equiv$. $\square$

We can see an example of the *refine* operator in Figure 10. The set of steps of the original abstract state is $[0, 5, 11]$, while the set of indices I is $[2, 6, 9, 14]$. On the left we depict the original abstract state, on the right its refined version with respect to I (the vertical lines are dashed in correspondence with the indices of I).

The *refine* operator will be used when dealing with two abstract states together (for example, in the least upper bound operator) to refine both states through the set of indices of the other state. For example, if f and g are the two abstract states being considered, the preliminary step is to refine f with the steps of g, and to refine g with the steps of f. In this way, we obtain two abstract states defined on the same set of indices (that is, the union of the two original indices sets).

Note that the abstract state resulting from a refinement operation is, in general, not normalized, because it violates the second condition (i.e., two consecutive constraints cannot have equal values). This is not a problem, since the refinement is used as a *preliminary* step of various operations (glb, lub, widening, ...) but each of these operations normalizes its final result, so we are sure that we will never produce (at the end of a computation) an abstract state which is not normalized, even when we used the refinement throughout the computation.

2.7. Greatest Lower Bound

Given two elements x and y of the abstract domain, the greatest lower bound operator defines the greatest element z that under-approximates both x and y. In TSF, this means that we have to create a sequence of trapezoids that are (i) as vast as possible and (ii) contained in both the given sequences of trapezoids.

Let $f' = \bigwedge_{0 \leq k \leq N} \{x_k : \mathbf{v_k}\}$ and $g' = \bigwedge_{0 \leq j \leq M} \{u_j : \mathbf{w_j}\}$ be two abstract states defined on the step$|X| \leq M$ must hold, sets $X = \{x_k : k \in [0, N]\}$ and $U = \{u_j : j \in [0, M]\}$, respectively (note that $N + 1 = |X|$ and $M + 1 = |U|$, that is, the number of steps of an abstract state corresponds to the cardinality of its step set). Let $min(f(t), g(t), [a, b])$ be an operator which compares the lines $f(t)$ and $g(t)$ in the domain range $[a, b]$ and returns the one which is always below the other one, in the assumption that the two lines do not intersect each other in such range. Since $f(t)$ and $g(t)$ are straight lines, it suffices to compare their values in $t = a$ and $t = b$. Let $max(f(t), g(t), [a, b])$ be the opposite operator, which returns the line always above. Let $StepIndexes(f)$ be a function which, given an abstract state f, extracts its step set. Then, we can write the formal definition of the greatest lower bound of f' and g' as the following algorithm:

```
$f = Refine(f', X \cup U)$
$g = Refine(g', X \cup U)$
$T = StepIndexes(f) = \{t_i\}$ //note that $StepIndexes(f) = StepIndexes(g)$
$P = |T|$
result=$\emptyset$
for($i$ from 0 to P−1) {
  if ($f_i^-$ intersects $g_i^+$ $\vee$ $f_i^+$ intersects $g_i^-$ in $]t_i..t_{i+1}[$)
    return $\perp^\sharp$
  if ($f_i^-$ intersects $g_i^-$ in $t_A \in ]t_i..t_{i+1}[$) {
    if ($f_i^+$ intersects $g_i^+$ in $t_B \in ]t_i..t_{i+1}[$) {
      if ($t_A$<$t_B$)
        resSteps = $t_i : (\max(f_i^-, g_i^-, [t_i..t_A]), \min(f_i^+, g_i^+, [t_i..t_A])) \wedge$
          $\wedge t_A : (\max(f_i^-, g_i^-, [t_A..t_B]), \min(f_i^+, g_i^+, [t_A..t_B])) \wedge$
          $\wedge t_B : (\max(f_i^-, g_i^-, [t_B..t_{i+1}]), \min(f_i^+, g_i^+, [t_B..t_{i+1}]))$
      else //the same but reversing $t_A$ and $t_B$
    } else
      resSteps = $t_i : (\max(f_i^-, g_i^-, [t_i..t_A]), \min(f_i^+, g_i^+, [t_i..t_A])) \wedge$
          $\wedge t_A : (\max(f_i^-, g_i^-, [t_A..t_{i+1}]), \min(f_i^+, g_i^+, [t_A..t_{i+1}]))$
  } else if ($f_i^+$ intersects $g_i^+$ in $t_B \in ]t_i..t_{i+1}[$) {
    resSteps = $t_i : (\max(f_i^-, g_i^-, [t_i..t_B]), \min(f_i^+, g_i^+, [t_i..t_B])) \wedge$
      $\wedge t_B : (\max(f_i^-, g_i^-, [t_B..t_{i+1}]), \min(f_i^+, g_i^+, [t_B..t_{i+1}]))$
  } else if ($[f_i^-(t), f_i^+(t)] \cap [g_i^-(t), g_i^+(t)] = \emptyset$ in $t = t_i \vee t = t_{i+1}$) {
    return $\perp^\sharp$
  } else
    resSteps = $t_i : (\max(f_i^-, g_i^-, [t_i..t_{i+1}]), \min(f_i^+, g_i^+, [t_i..t_{i+1}]))$
  result = result $\wedge$ resSteps
}
if ($\exists s_i \in StepIndexes(result) : [result_i^-(s_{i+1}), result_i^+(s_{i+1})]$
    $\cap [result_{i+1}^-(s_{i+1}), result_{i+1}^+(s_{i+1})] = \emptyset$)
    //i.e., result does not satisfy the second validity condition
  return $\perp^\sharp$
else return Norm(result)
```

An informal description of this algorithm is as follows:

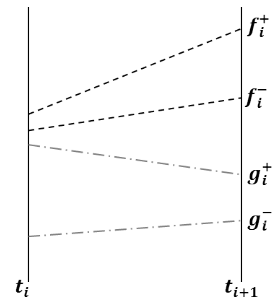
Step 1 (lines 1-3 of the algorithm) In order to make the two abstract states f', g' directly comparable, we refine them on the same set of steps, creating two new abstract states f, g. Let $f = Refine(f', X \cup U)$ and $g = Refine(g', X \cup U)$ be these two new states. By Lemma 2.4, these refined states are equivalent to f' and g', so the greatest lower bound of f, g is also the greatest lower bound of f', g'.

Step 2 Let $T = \{t_i : i \in [0, P-1]\}$ (where $P = |X \cup U| = |T|$) be the set of steps indices of f and g. Let $(f_i^-, f_i^+), (g_i^-, g_i^+)$ be the values (i.e., lower and upper sides) of f, g in the generic step $[t_i, t_{i+1}]$. We will split each step $[t_i, t_{i+1}]$ into sub-steps, with respect to intersections of the sides of the two trapezoids (one from f, one from g, lines 9-21). If there are no intersections, we leave the step unchanged (line 26). Each step $[t_i, t_{i+1}]$ will then generate one (or more) steps in $f \cap^\sharp g$, depending on the intersections of the sides in such step. The goal is to obtain sub-steps in which the sides of the two trapezoids do not intersect each other in any point *inside* the sub-step range.
By Equation 1 (first validity condition), we know that, *inside* a step, the upper and lower side of a trapezoid do not intersect. In fact, the two sides could have an extreme in common (the value at t_i or at t_{i+1}) or they could be the same line, but they surely do not have an intersection point inside $[t_i, t_{i+1}]$, otherwise they would violate the first constraint (Equation 1). For this reason, we know for sure that f_i^-, f_i^+ do not intersect each other, and the same holds for g_i^-, g_i^+. The possible intersections inside (t_i, t_{i+1}) (extremes excluded) are then: (1) between f_i^- and g_i^-, (2) between f_i^+ and g_i^+, (3) between f_i^- and g_i^+, and (4) between f_i^+ and g_i^-.
In case 1 and 2 (intersection between f_i^-, g_i^- or between f_i^+, g_i^+, lines 9-21), we split the step in sub-steps with respect to the intersection point. Note that we could obtain two or three sub-steps: if only f_i^-, g_i^- intersect each other (lines 17-18) then we have two sub-steps (and the same happens if the only intersection regards f_i^+, g_i^+, lines 20-21 - see Figure 11(d)) but if there are two intersections (one between f_i^-, g_i^- and the other between f_i^+, g_i^+, lines 12-14 - see Figure 11(e)) then we obtain three sub-steps.
In case 3 and 4 (intersection between f_i^-, g_i^+ - Figure 11(b) - or between f_i^+, g_i^- - Figure 11(c), lines 7-8), instead of splitting the step with respect to the intersection point, we immediately return $\perp^\sharp$ as result of the $\cap^\sharp$ operation. In fact, assume that f_i^-, g_i^+ have an intersection (the same reasoning holds if the intersection is between f_i^+, g_i^-). Then, at one extreme of the step (i.e., at t_i or t_{i+1}), the lower side of the trapezoid of f (f_i^-) is greater than the upper side of the trapezoid of g (g_i^+). This means that, at such point, the two states have no values in common since the areas of the trapezoids do not intersect: the lowest value assumed by the continuous functions abstracted by f is greater than the greatest value assumed by the continuous functions abstracted by g. The result of the greatest lower bound must therefore be $\perp^\sharp$.
If the step $[t_i, t_{i+1}]$ does not contain any intersection, two cases apply:

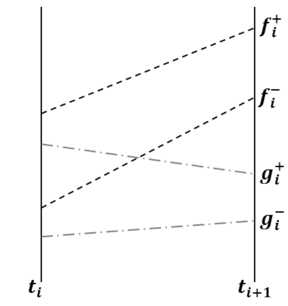
— if $[f_i^-(t_i), f_i^+(t_i)] \cap [g_i^-(t_i), g_i^+(t_i)] = \emptyset \vee [f_i^-(t_{i+1}), f_i^+(t_{i+1})] \cap [g_i^-(t_{i+1}), g_i^+(t_{i+1})] = \emptyset$ (Figure 11(a), lines 22-23) then we return $\perp^\sharp$, because it means that at one extreme of the step (i.e., at t_i or t_{i+1}) the lowest value of one state is greater than the greatest value of the other one.

— otherwise, we do not need to split the step (line 25), since we are in the case sketched by Figure 11(f).

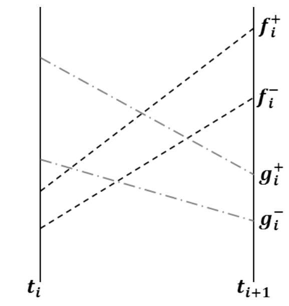
Step 3 We know for sure that, in each of the sub-steps generated by the algorithm: (i) the two states have some values in common (i.e., the areas of the trapezoids have a non-



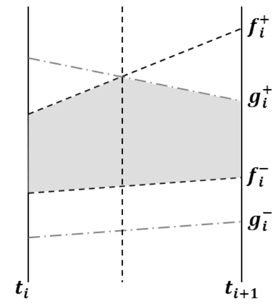
(a) No intersections, since f_i^- is above g_i^+, so the result of the glb operation is $\perp^\sharp$



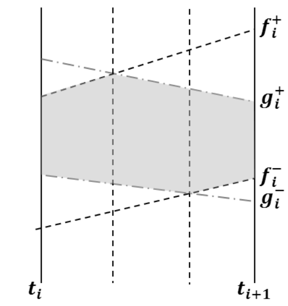
(b) One intersection between f_i^- and g_i^+, so the result of the glb operation is $\perp^\sharp$



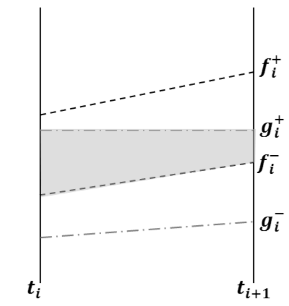
(c) Four intersections; the result of the glb operation is $\perp^\sharp$ because f_i^+ intersects g_i^-



(d) One intersection between f_i^+ and g_i^+, the step is split into two sub-steps. The gray area represents the two resulting trapezoids.



(e) Two intersections, one between f_i^-, g_i^- and one between f_i^+, g_i^+. The step is split into three sub-steps. The gray area represents the three resulting trapezoids.



(f) No intersections, the step is not split into sub-steps. The gray area represents the resulting trapezoid.

Fig. 11. Examples of the GLB sub-step splitting

empty intersection) at each point of the sub-step; (ii) inside the sub-step the four sides of the two trapezoids derived from f and g do not have any intersection. Then, for each sub-step it is easy to compute the value of the corresponding trapezoid of the result ($f \sqcap^\sharp g$): the lower side will correspond to the greatest of the two lower sides of f, g in the sub-step, while the upper side will correspond to the lowest of the two upper sides of f, g in the sub-step. In Figures 11(d), 11(e) and 11(f) we can see the result of this procedure in three different cases (step split into two substeps, step split into three substeps, step not split, respectively). The gray area represents the resulting trapezoids of $f \sqcap^\sharp g$. We add the newly generated sub-steps to the result at line 26 (at the end of each iteration of the loop).

Step 4 At the end of the computation (lines 28-32) we normalize the resulting abstract state using $Norm$ (line 31). In the abstract state $f \sqcap^\sharp g$, it could happen that the intervals individuated by the two sides of two consecutive steps do not intersect at the border between the two steps (for an example, see Figure 12, where the straight lines represent the trapezoids of f, the dashed lines represent the trapezoids of g and the darkened areas represent the trapezoids of $f \sqcap^\sharp g$). For this reason, after computing the greatest lower bound $h = f \sqcap^\sharp g$, we perform a check that h respects the second validity condition (that is, the situation of Figure 12 never happens). If the check fails, we return $\perp^\sharp$ as result of the glb operation (lines 28-31). This happens

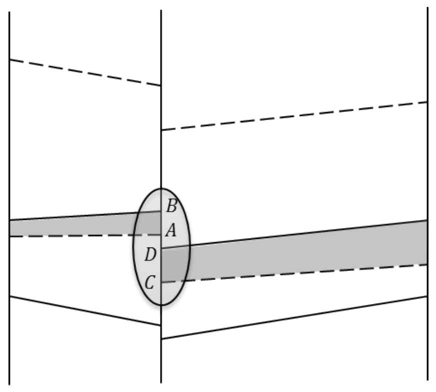


Fig. 12. The glb does not always preserve the second validity condition

because, if one of such border intersections is empty, then the two functions f and g do not have any possible value in common at that border and thus their lower bound in the TSF domain does not exist.

In conclusion, the intersection $f \cap^\sharp g$ creates a new step function whose value is at every time t the intersection $f(t) \cap g(t)$. If this intersection is empty in at least one point, we cannot return a continuous abstract function and then we define $f \cap^\sharp g$ as $\bot^\sharp$, the bottom element of $D^\sharp$.

LEMMA 2.5 ($\cap^\sharp$ MAINTAINS THE VALIDITY CONDITIONS). *Let f and g be two abstract states which both respect the validity conditions enunciated in Section 2.2 and which are defined on the same set of indices $T = \{t_i : i \in [0, N]\}$ (if that is not the case, we can use the refine operator). Let $f \cap^\sharp g = h$ be their greatest lower bound. Then, h respects the two validity conditions.*

PROOF. If $h = \bot^\sharp$, then it automatically respects the validity conditions ($\bot^\sharp$ is a valid element of the domain). Now assume $h \neq \bot^\sharp$ and $h = \bigwedge_{j\in[0,M]}\{x_j : \mathbf{v_j} = (h_j^-, h_j^+)\}$. By construction of h, we know that $T \subseteq X = \{x_j : j \in [0, M]\}$, that is, the steps indices of f and g are a subset of those of h. In fact, h has a number of steps which is greater or equal to the one of f and g, since each step of f and g generates from one to three steps in h, depending on the sides intersections (lines 12-14, 17-18, 20-21 and 25).

The validity conditions applied to h are the following ones:

(1) $\forall j \in [0, M], h_j^-(t_j) \leq h_j^+(t_j) \wedge h_j^-(t_{j+1}) \leq h_j^+(t_{j+1})$
(2) $\forall j \in [0, M-1], [h_j^-(t_{j+1}), h_j^+(t_{j+1})] \cap [h_{j+1}^-(t_{j+1}), h_{j+1}^+(t_{j+1})] \neq \emptyset$

(1) Consider the generic step $[x_j, x_{j+1}]$, where $j \in [0, M]$ and (h_j^-, h_j^+) is the value of h in such step. By construction of h, we know that each step range $[x_j, x_{j+1}]$ of h is a subset of a step range from f, g, that is, $\exists i : [x_j, x_{j+1}] \subseteq [t_i, t_{i+1}]$ (lines 12-14, 17-18, 20-21 and 25). Let $(f_i^-, f_i^+), (g_i^-, g_i^+)$ be the values of f and g in such step, respectively. By construction, we also know that the lines $f_i^-, f_i^+, g_i^-, g_i^+$ do not intersect each other inside $[x_j, x_{j+1}]$ (lines 7,9,10,19). Regarding (h_j^-, h_j^+), we know that h_j^- corresponds to f_i^- or g_i^-, and that h_j^+ corresponds to f_i^+ or g_i^+ (since in the algorithm we always use the functions $max(\cdot,\cdot,\cdot)$ and $min(\cdot,\cdot,\cdot)$ which return one of the first two inputs - lines 12-14, 17-18, 20-21 and 25). If both sides h_j^-, h_j^+ correspond to sides from the same state (i.e., $h_j^- = f_i^- \wedge h_j^+ = f_i^+$ or $h_j^- = g_i^- \wedge$

$h_j^+ = g_i^+$), then the first validity condition is satisfied, because both f and g satisfy it. Otherwise, either $h_j^- = f_i^- \wedge h_j^+ = g_i^+$ or $h_j^- = g_i^- \wedge h_j^+ = f_i^+$. Suppose that $h_j^- = g_i^- \wedge h_j^+ = f_i^+$ (the other case is symmetric). Then, to prove that h satisfies the first validity condition, we must prove that $f_i^+(x_j) \geq g_i^-(x_j) \wedge f_i^+(x_{j+1}) \geq g_i^-(x_{j+1})$:
— To prove $f_i^+(x_j) \geq g_i^-(x_j)$, suppose by absurd that $f_i^+(x_j) < g_i^-(x_j)$. Then, the result of the glb operation would have been $\bot^\sharp$ (because $[f_i^-(x_j), f_i^+(x_j)] \cap [g_i^-(x_j), g_i^+(x_j)] = \emptyset$), which is in contradiction with the hypothesis that $h \neq \bot^\sharp$.
— To prove $f_i^+(x_{j+1}) \geq g_i^-(x_{j+1})$, the reasoning is symmetrical.

(2) The second validity condition is automatically respected by each output of the glb computation, since we return $\bot^\sharp$ if the property is not satisfied (lines 28-31).

□

LEMMA 2.6 ($\cap^\sharp$ IS THE GREATEST LOWER BOUND OPERATOR). *Let f, g be two abstract states and let $h = f \cap^\sharp g$ be their greatest lower bound. Then:*

(*1*) $h \subseteq^\sharp f \wedge h \subseteq^\sharp g$
(*2*) $k \subseteq^\sharp h$ $\forall k$ *lower bound of f and g*

PROOF. Let us first assume that $h \neq \bot^\sharp$ (afterwards we will consider also the case where $h = \bot^\sharp$). Also, let f, g, h be defined on the same set of steps $T = \{t_i : i \in [0, N]\}$. We can obtain this situation by refining the three abstract states on the union of their step sets through *refine*. By Lemma 2.4 we know that an abstract state and its refined version are equivalent, so the refinement does not change order relationship between states. For this reason, in all our proofs we can use indifferently either the original abstract state or its refined version.

(1) Since f, g, h are defined on the same set of indices T, we can write $f = \bigwedge_{i \in [0,N]} \{t_i : \mathbf{f_i}\}$, $g = \bigwedge_{i \in [0,N]} \{t_i : \mathbf{g_i}\}$ and $h = \bigwedge_{i \in [0,N]} \{t_i : \mathbf{h_i}\}$. By definition of $\subseteq^\sharp$, we have that $h \subseteq^\sharp f \wedge h \subseteq^\sharp g \Leftrightarrow \forall i \in [0, N] : \mathbf{h_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{f_i} \wedge \mathbf{h_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{g_i}$. Let $[t_i, t_{i+1}]$ be a generic step, where $i \in [0, N]$. By construction of h, we know that f_i^+ and g_i^+ do not intersect each other in $[t_i, t_{i+1}]$, as well as f_i^- and g_i^- do not intersect (lines 9,10,19). Then:
— For the upper line, we know, by construction, that h_i^+ is the lowest between the two non-intersecting sides f_i^+ and g_i^+ (lines 12-14, 17-18, 20-21 and 25, where we use $min(f_i^+, g_i^+, \cdot)$), so $\forall t \in [t_i, t_{i+1}], h_i^+(t) \leq f_i^+(t) \wedge h_i^+(t) \leq g_i^+(t)$.
— For the lower line, we know, by construction, that h_i^- is the greatest between the two not-intersecting sides f_i^- and g_i^- (lines 12-14, 17-18, 20-21 and 25, where we use $max(f_i^-, g_i^-, \cdot)$), so $\forall t \in [t_i, t_{i+1}], f_i^-(t) \leq h_i^-(t) \wedge g_i^-(t) \leq h_i^-(t)$.
— From $\forall t \in [t_i, t_{i+1}], h_i^+(t) \leq f_i^+(t)$ and from $\forall t \in [t_i, t_{i+1}], f_i^-(t) \leq h_i^-(t)$, by definition of $\mathbf{u} \sqsubseteq_{[a,b]} \mathbf{v}$ (Equation 3), it follows that $\mathbf{h_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{f_i}$. From $\forall t \in [t_i, t_{i+1}], h_i^+(t) \leq g_i^+(t) \wedge g_i^-(t) \leq h_i^-(t)$, it follows that $\mathbf{h_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{g_i}$.
— Then, $\forall i \in I, \mathbf{h_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{f_i} \wedge \mathbf{h_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{g_i}$ and so we conclude that $h \subseteq^\sharp f \wedge h \subseteq^\sharp g$ by definition of $\subseteq^\sharp$.

(2) Let f, g, h be defined on the same set of indices T. By contradiction, assume that $\exists k : k$ is a lower bound of f and g, and $k \not\subseteq^\sharp h$. Let us suppose that k is defined on T (we can always obtain it by applying *refine*). This means that $\exists i \in I : \mathbf{k_i} \not\sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i}$. Since $h_i^-, h_i^+, k_i^-, k_i^+$ are straight lines, and by the definition of $\sqsubseteq_{[a,b]}$ (Equation 3), at least one of the following equations must be satisfied (otherwise it would hold $\mathbf{k_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i}$):

— $k_i^+(t_i) > h_i^+(t_i)$
— $k_i^+(t_{i+1}) > h_i^+(t_{i+1})$
— $k_i^-(t_i) < h_i^-(t_i)$
— $k_i^-(t_{i+1}) < h_i^-(t_{i+1})$

Suppose that $k_i^+(t_i) > h_i^+(t_i)$ is true (the reasoning is symmetrical for the other three equations). By construction of h, we know that $h_i^+ = f_i^+ \vee h_i^+ = g_i^+$ (lines 12-14, 17-18, 20-21 and 25, where $h_i^+ = min(f_i^+, g_i^+, \cdot)$). So we can rewrite $k_i^+(t_i) > h_i^+(t_i)$ as $k_i^+(t_i) > f_i^+(t_i) \vee k_i^+(t_i) > g_i^+(t_i)$. If $k_i^+(t_i) > f_i^+(t_i)$ holds, then $\mathbf{k_i} \not\sqsubseteq_{[t_i,t_{i+1}]} \mathbf{f_i}$, and this implies that $k \not\sqsubseteq^\sharp f$. We have reached a contradiction, because k is a lower bound of f. On the other hand, if $k_i^+(t_i) > g_i^+(t_i)$ holds, then $\mathbf{k_i} \not\sqsubseteq_{[t_i,t_{i+1}]} \mathbf{g_i}$, and this implies that $k \not\sqsubseteq^\sharp g$. We have reached a contradiction in this case, too, because k is also a lower bound of g.

Assume now that $h = \bot^\sharp$.

(1) This part of the proof is trivial, since $\bot^\sharp \sqsubseteq^\sharp f \wedge \bot^\sharp \sqsubseteq^\sharp g$ by definition of $\bot^\sharp$.
(2) If $h = \bot^\sharp$, then one between the following two facts must be true:
 (a) in some point t of the domain, the two trapezoids of f, g did not have any point in common: $[f_i^-(t), f_i^+(t)] \cap [g_i^-(t), g_i^+(t)] = \emptyset$ (supposing that $t \in [t_i, t_{i+1}]$); this means that $f_i^-(t) > g_i^+(t) \vee g_i^-(t) > f_i^+(t)$ (lines 8 and 23)
 (b) the result of the glb computation did not respect the second validity condition, because at the border between two steps $[t_i, t_{i+1}]$ and $[t_{i+1}, t_{i+2}]$ the two trapezoids of h did not have any point in common: $[h_i^-(t_{i+1}), h_i^+(t_{i+1})] \cap [h_{i+1}^-(t_{i+1}), h_{i+1}^+(t_{i+1})] = \emptyset$ (lines 28-31)

Consider case (a), and suppose $k \neq \bot^\sharp$. Since k is a lower bound of f, g, it must respect the following condition: $[k_i^-(t), k_i^+(t)] \subseteq [f_i^-(t), f_i^+(t)] \wedge [k_i^-(t), k_i^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$ throughout all the domain and in particular in the point t where, by hypothesis, $f_i^-(t) > g_i^+(t) \vee g_i^-(t) > f_i^+(t)$. Such condition implies that: $k_i^-(t) \geq f_i^-(t) \wedge k_i^+(t) \leq f_i^+(t) \wedge k_i^-(t) \geq g_i^-(t) \wedge k_i^+(t) \leq g_i^+(t)$. Now, if $f_i^-(t) > g_i^+(t)$, considering that $k_i^-(t) \geq f_i^-(t)$, we obtain $k_i^-(t) > g_i^+(t)$. This fact, combined with $k_i^+(t) \geq k_i^-(t)$ (for the first validity condition of abstract states), results in $k_i^+(t) > g_i^+(t)$, which cannot hold, because $k_i^+(t) \leq g_i^+(t)$. We obtained a contradiction. A similar reasoning can be done when $g_i^-(t) > f_i^+(t)$ (instead of $f_i^-(t) > g_i^+(t)$). Since we reached a contradiction, k must be $\bot^\sharp$ and then $\bot^\sharp = k \sqsubseteq^\sharp h = \bot^\sharp$.

Now consider case (b) (the result of the glb did not respect the second validity condition - lines 28-29). We will prove that any other lower bound k of f and g does not respect the second validity condition, too. Suppose by absurd that $\exists k$: k is a lower bound of f and g and $k \neq \bot^\sharp$. The hypothesis $[h_i^-(t_{i+1}), h_i^+(t_{i+1})] \cap [h_{i+1}^-(t_{i+1}), h_{i+1}^+(t_{i+1})] = \emptyset$ (violation of the second validity condition by h) can be rewritten, by construction of h (lines 12-14, 17-18, 20-21 and 25, where we assign $h_i^- = max(f_i^-, g_i^-, \cdot)$ and $h_i^+ = min(f_i^+, g_i^+, \cdot)$), as $[A, B] \cap [C, D] = \emptyset$ where $A = \max(f_i^-(t_{i+1}), g_i^-(t_{i+1})), B = \min(f_i^+(t_{i+1}), g_i^+(t_{i+1})), C = \max(f_{i+1}^-(t_{i+1}), g_{i+1}^-(t_{i+1})), D = \min(f_{i+1}^+(t_{i+1}), g_{i+1}^+(t_{i+1}))$ (see also Figure 12 for the notation of A, B, C, D). This means that $B < C \vee A > D$. Suppose that $A > D$, exactly like it happens in Figure 12 (the reasoning is symmetrical if $B < C$). The abstract state k, being a lower bound, must be less or equal than f and g. So, for example, $k_i^-(t_{i+1})$ must be greater or equal than $f_i^-(t_{i+1})$ and $g_i^-(t_{i+1})$. Thus, $k_i^-(t_{i+1}) \geq A$. For the same reason (k is a lower bound of f, g), $k_{i+1}^+(t_{i+1})$ must be less or equal than both $f_{i+1}^+(t_{i+1})$ and $g_{i+1}^+(t_{i+1})$: thus, $k_{i+1}^+(t_{i+1}) \leq D$. Then, since we know that $A > D$, by transitivity we have $k^+_{i+1}(t_{i+1}) < k^-_i(t_{i+1})$, and this violates the second validity condition. The TSF domain does not include abstract states which do not respect such condition, so k becomes $\bot^\sharp$. Trivially, it follows $\bot^\sharp = k \subseteq^\sharp h = \bot^\sharp$.

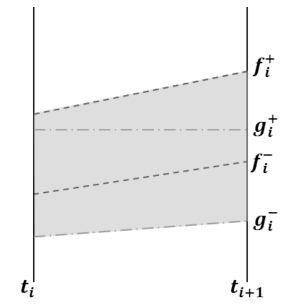
□

2.8. Least upper bound

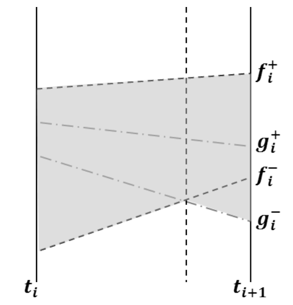
Given two elements x and y of the abstract domain, the least upper bound operator defines the least element z that over-approximates both x and y. In TSF, this means that we have to create a sequence of trapezoids that are as narrow as possible and that, at the same time, contain the two given sequences of trapezoids.

Let $f' = \bigwedge_{0\leq k\leq N}\{x_k : \mathbf{v_k}\}$ and $g' = \bigwedge_{0\leq j\leq M}\{u_j : \mathbf{w_j}\}$ be two abstract states. In order to define the least upper bound of f' and g', we use an algorithm very similar to the one presented for the glb in Section 2.7. First, we refine f' and g' on the same set of steps, obtaining $f = Refine(f', X \cup U)$ and $g = Refine(g', X \cup U)$ where X and U are the step set of f' and g', respectively. Then, for each step of f and g we look at the two trapezoids and check if there are intersections either between the two lower sides (f^-_i, g^-_i) or between the two upper sides (f^+_i, g^+_i). We split the step with respect to such intersections; if there are none, the step remains unsplit. In each of these new steps, we are sure that nor the upper sides nor the lower sides intersect each other. So, the resulting trapezoid for each new step is made by the greatest of the upper sides and the lowest of the lower sides. For some examples, see Figure 13. The algorithm is as follows:

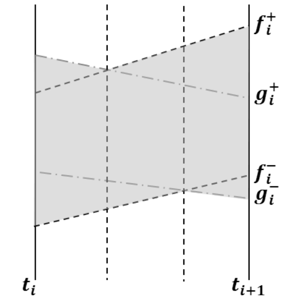
```
$f = Refine(f', X \cup U)$
$g = Refine(g', X \cup U)$
$T = StepIndexes(f) = \{t_i\}$ // StepIndexes(f) = StepIndexes(g)
$P = |T|$
result=$\emptyset$
for($i$ from 0 to P−1) {
  if ($f^-_i$ intersects $g^-_i$ in $t_A \in ]t_i..t_{i+1}[$) {
    if ($f^+_i$ intersects $g^+_i$ in $t_B \in ]t_i..t_{i+1}[$) {
      if ($t_A < t_B$)
        resSteps = $t_i : (\min(f^-_i, g^-_i, [t_i..t_A]), \max(f^+_i, g^+_i, [t_i..t_A]))\wedge$
          $\wedge t_A : (\min(f^-_i, g^-_i, [t_A..t_B]), \max(f^+_i, g^+_i, [t_A..t_B]))\wedge$
          $\wedge t_B : (\min(f^-_i, g^-_i, [t_B..t_{i+1}]), \max(f^+_i, g^+_i, [t_B..t_{i+1}]))$
      else //the same but reversing $t_A$ and $t_B$
    } else
      resSteps = $t_i : (\min(f^-_i, g^-_i, [t_i..t_A]), \max(f^+_i, g^+_i, [t_i..t_A]))\wedge$
          $\wedge t_A : (\min(f^-_i, g^-_i, [t_A..t_{i+1}]), \max(f^+_i, g^+_i, [t_A..t_{i+1}]))$
  } else if ($f^+_i$ intersects $g^+_i$ in $t_B \in ]t_i..t_{i+1}[$) {
    resSteps = $t_i : (\min(f^-_i, g^-_i, [t_i..t_B]), \max(f^+_i, g^+_i, [t_i..t_B]))\wedge$
      $\wedge t_B : (\min(f^-_i, g^-_i, [t_B..t_{i+1}]), \max(f^+_i, g^+_i, [t_B..t_{i+1}]))$
  } else
    resSteps = $t_i : (\min(f^-_i, g^-_i, [t_i..t_{i+1}]), \max(f^+_i, g^+_i, [t_i..t_{i+1}]))$
  result = result ∧ resSteps
}
return Norm(result)
```



(a) No intersections, the step remains unsplit, resulting in one trapezoid (colored in gray)



(b) One intersection between f_i^- and g_i^-, resulting in two sub-steps and, thus, two trapezoids (colored in gray)



(c) Two intersections, one between f_i^-, g_i^- and one between f_i^-, g_i^-, resulting in three sub-steps and, thus, three trapezoids (colored in gray)

Fig. 13. Examples of the LUB sub-step splitting

The algorithm is very similar to the one introduced for the glb, with the following differences:

— $\min$ and $\max$ are reversed, since here we keep the lowest line between f_i^-, g_i^- and the greatest line between f_i^+, g_i^+ in order to over-approximate f_i and g_i.
— we never return $\bot^\sharp$ because the lub between two values is always possible and it also satisfies, by construction, the validity conditions (see also Lemma 2.8 for the formal proof).

LEMMA 2.7 ($\cup^\sharp$ IS THE LEAST UPPER BOUND OPERATOR). *$\cup^\sharp$ is a least upper bound operator. Let $h = f \cup^\sharp g$ be the least upper bound of f and g. Then:*

(1) $f \subseteq^\sharp h \wedge g \subseteq^\sharp h$
(2) $h \subseteq^\sharp k$ $\forall k$ upper bound of f and g

PROOF. Let f, g, h be defined on the same set of steps $T = \{t_i : i \in [0, N]\}$: $f = \bigwedge_{i \in I}\{t_i : \mathbf{f_i}\}, g = \bigwedge_{i \in I}\{t_i : \mathbf{g_i}\}$ and $h = \bigwedge_{i \in I}\{t_i : \mathbf{h_i}\}$. We can obtain this situation by refining the three abstract states on the union of their step sets.

(1) By definition of $\subseteq^\sharp$, we have that $f \subseteq^\sharp h \wedge g \subseteq^\sharp h \Leftrightarrow \forall i \in [0, N] : \mathbf{f_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i} \wedge \mathbf{g_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i}$. Let $[t_i, t_{i+1}]$ be a generic step, where $i \in [0, N]$. By construction of h, we know that f_i^+ and g_i^+ do not intersect each other in $[t_i, t_{i+1}]$, as well as f_i^- and g_i^- do not intersect (lines 7, 8, 17). Then:
 — For the upper line, we know by construction that h_i^+ is the greatest between the two non-intersecting sides f_i^+ and g_i^+ (lines 10-12, 15-16, 18-19, 21), so $\forall t \in [t_i, t_{i+1}], h_i^+(t) \geq f_i^+(t) \wedge h_i^+(t) \geq g_i^+(t)$.
 — For the lower line, we know by construction that h_i^- is the lowest between the two non-intersecting sides f_i^- and g_i^- (lines 10-12, 15-16, 18-19, 21), so $\forall t \in [t_i, t_{i+1}], h_i^-(t) \leq f_i^-(t) \wedge h_i^-(t) \leq g_i^-(t)$.
 — From $\forall t \in [t_i, t_{i+1}], h_i^+(t) \geq f_i^+(t)$ and from $\forall t \in [t_i, t_{i+1}], h_i^-(t) \leq f_i^-(t)$, by definition of $\mathbf{u} \sqsubseteq_{[a,b]} \mathbf{v}$ (Equation 3), it follows that $\mathbf{f_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i}$. From $\forall t \in [t_i, t_{i+1}], h_i^+(t) \geq g_i^+(t) \wedge h_i^-(t) \leq g_i^-(t)$, it follows that $\mathbf{g_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i}$.
 — Then, $\forall i \in I, \mathbf{f_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i} \wedge \mathbf{g_i} \sqsubseteq_{[t_i, t_{i+1}]} \mathbf{h_i}$ and so we conclude that $f \subseteq^\sharp h \wedge g \subseteq^\sharp h$ by definition of $\subseteq^\sharp$.

(2) Assume that k is defined on the same set of steps $T = \{t_i : i \in [0, N]\}$ as that of f, g, h (as said before, if this is not true, we can make it true by refining each abstract state on the union of the four step sets, maintaining all the relationships among such states). By contradiction, assume that $h \not\sqsubseteq^\sharp k$, that is $\exists i \in [0, N] : \mathbf{h_i} \not\sqsubseteq_{[t_i,t_{i+1}]} \mathbf{k_i}$. Since $h_i^-, h_i^+, k_i^-, k_i^+$ are straight lines, and by the definition of $\sqsubseteq_{[a,b]}$ (Equation 3), at least one of the following equations must be true (otherwise it would hold $\mathbf{h_i} \sqsubseteq_{[t_i,t_{i+1}]} \mathbf{k_i}$):
— $k_i^+(t_i) < h_i^+(t_i)$
— $k_i^+(t_{i+1}) < h_i^+(t_{i+1})$
— $k_i^-(t_i) > h_i^-(t_i)$
— $k_i^-(t_{i+1}) > h_i^-(t_{i+1})$
Suppose that $k_i^+(t_i) < h_i^+(t_i)$ is true (the reasoning is symmetrical for the other three equations). By construction of h, we know that $h_i^+ = f_i^+ \vee h_i^+ = g_i^+$ (lines 10-12, 15-16, 18-19, 21, where $h_i^+ = max(f_i^+, g_i^+, \cdot)$). So we can rewrite $k_i^+(t_i) < h_i^+(t_i)$ as $k_i^+(t_i) < f_i^+(t_i) \vee k_i^+(t_i) < g_i^+(t_i)$. If $k_i^+(t_i) < f_i^+(t_i)$ holds, then $\mathbf{f_i} \not\sqsubseteq_{[t_i,t_{i+1}]} \mathbf{k_i}$, and this implies that $f \not\sqsubseteq^\sharp k$. We have reached a contradiction, because k is an upper bound of f. On the other hand, if $k_i^+(t_i) < g_i^+(t_i)$ holds, then $\mathbf{g_i} \not\sqsubseteq_{[t_i,t_{i+1}]} \mathbf{k_i}$, and this implies that $g \not\sqsubseteq^\sharp k$. We have reached a contradiction in this case, too, because k is also an upper bound of g.

□

LEMMA 2.8 ($\cup^\sharp$ MAINTAINS THE VALIDITY CONDITIONS). *Let f, g be two abstract states which both respect the validity conditions enunciated in Section 2.2 and which are defined on the same set of indices $T = \{t_i : i \in [0, N]\}$ (if it is not the case, we can use the refine operator). Let $f \cup^\sharp g = h$ be their least upper bound. Then, the abstract state h satisfies the two validity conditions.*

PROOF. Let $h = \bigwedge_{j\in[0,M]}\{x_j : \mathbf{v_j} = (h_j^-, h_j^+)\}$ be the lub of f, g. By construction of h, we know that $T \subseteq X = \{x_j : j \in [0, M]\}$, that is, the steps indices of f and g are a subset of those of h. In fact, h has a number of steps which is greater or equal to the one of f, g, since each step of f, g generates from one to three steps in h, depending on the sides intersections (lines 10-12, 15-16, 18-19, 21). The validity conditions applied to h are the following ones:

(1) $\forall j \in [0, M], h_j^-(x_j) \leq h_j^+(x_j) \wedge h_j^-(x_{j+1}) \leq h_j^+(x_{j+1})$
(2) $\forall j \in [0, M-1], [h_j^-(x_{j+1}), h_j^+(x_{j+1})] \cap [h_{j+1}^-(x_{j+1}), h_{j+1}^+(x_{j+1})] \neq \emptyset$

(1) Consider the generic step $[x_j, x_{j+1}]$, where $j \in [0, M]$ and (h_j^-, h_j^+) is the value of h in such step. By construction of h, we know that each step range $[x_j, x_{j+1}]$ of h is a subset of a step range from f, g, that is, $\exists i : [x_j, x_{j+1}] \subseteq [t_i, t_{i+1}]$ (lines 10-12, 15-16, 18-19, 21). Let $(f_i^-, f_i^+), (g_i^-, g_i^+)$ be the values of f and g in such step, respectively. Regarding (h_j^-, h_j^+), we know that h_j^- corresponds to the lowest line between f_i^- and g_i^-, and that h_j^+ corresponds to the greatest line between f_i^+ or g_i^+ (lines 10-12, 15-16, 18-19, 21, where we assign $h_j^+ = max(f_i^+, g_i^+, \cdot)$ and $h_j^- = min(f_i^-, g_i^-, \cdot)$). For this reason, it holds that $h_j^+(x_j) \geq f_i^+(x_j) \wedge h_j^+(x_j) \geq g_i^+(x_j)$ (for the upper side) and that $h_j^-(x_j) \leq f_i^-(x_j) \wedge h_j^-(x_j) \leq g_i^-(x_j)$ (for the lower side). By hypothesis, f satisfies the first validity condition, so it holds also $f_i^+(x_j) \geq f_i^-(x_j)$. Combining $h_j^+(x_j) \geq f_i^+(x_j) \wedge f_i^+(x_j) \geq f_i^-(x_j) \wedge h_j^-(x_j) \leq f_i^-(x_j)$ we obtain, by transitivity,

that $h_j^+(x_j) \geq h_j^-(x_j)$. The same reasoning can be done about the value of h in x_{j+1}, obtaining $h_j^+(x_{j+1}) \geq h_j^-(x_{j+1})$. Then, h satisfies the first validity condition.

(2) As before, consider the generic step $[x_j, x_{j+1}]$, where $j \in [0, M-1]$ and (h_j^-, h_j^+) is the value of h in such step. By construction of h, $\exists i : [x_j, x_{j+1}] \subseteq [t_i, t_{i+1}]$ (lines 10-12, 15-16, 18-19, 21). Let $(f_i^-, f_i^+), (g_i^-, g_i^+)$ be the values of f and g in such step respectively. Moreover, let $(h_{j+1}^-, h_{j+1}^+), (f_{i+1}^-, f_{i+1}^+), (g_{i+1}^-, g_{i+1}^+)$ be the values of f, g and h in $[x_{j+1}, x_{j+2}]$ respectively. Using a similar notation to the one introduced in the second part of Lemma 2.6, let $A = \min(f_i^-(x_{j+1}), g_i^-(x_{j+1})), B = \max(f_i^+(x_{j+1}), g_i^+(x_{j+1})), C = \min(f_{i+1}^-(x_{j+1}), g_{i+1}^-(x_{j+1})), D = \max(f_{i+1}^+(x_{j+1}), g_{i+1}^+(x_{j+1}))$ be the lower and upper values assumed by h in x_{j+1} with respect to the trapezoids of steps $[x_j, x_{j+1}]$ and $[x_{j+1}, x_{j+2}]$. The second validity condition ($[h_j^-(x_{j+1}), h_j^+(x_{j+1})] \cap [h_{j+1}^-(x_{j+1}), h_{j+1}^+(x_{j+1})] \neq \emptyset$) can be then rewritten, by construction of h (lines 10-12, 15-16, 18-19, 21, where we assign $h_j^+ = max(f_i^+, g_i^+, \cdot)$ and $h_j^- = min(f_i^-, g_i^-, \cdot)$), as $[A, B] \cap [C, D] \neq \emptyset$. By contradiction, suppose that $[A, B] \cap [C, D] = \emptyset$. Then it must be that $A > D$ or $B < C$. Suppose that $A > D$ is true (the same reasoning applies to $B < C$ symmetrically). Since $A = \min(f_i^-(x_{j+1}), g_i^-(x_{j+1}))$ and $D = \max(f_{i+1}^+(x_{j+1}), g_{i+1}^+(x_{j+1}))$, this means that $f_i^-(x_{j+1}) > f_{i+1}^+(x_{j+1})$. This is a contradiction, because either f has a step starting at x_{j+1} (and in this case f would not respect the second validity condition at the border x_{j+1}) or not. In this second case, $[x_{j+1}, x_{j+2}] \subseteq [t_i, t_{i+1}]$ and thus $f_i^- = f_{i+1}^- \wedge f_i^+ = f_{i+1}^+$. Then, $f_i^-(x_{j+1}) > f_{i+1}^+(x_{j+1})$ can be rewritten as $f_{i+1}^-(x_{j+1}) > f_{i+1}^+(x_{j+1})$ which violates the first validity condition inside step $[t_i, t_{i+1}]$. We reached a contradiction, so we must discard the absurd hypothesis that the second validity condition is not respected by $\cup^\sharp$.

□

2.9. *Compact* operator

The dual operator of *refine* is *compact*. This operator reduces the number of steps contained in an abstract state, and it is useful in order to keep it below a given threshold and make the analysis convergent. *Compact* works by merging a pair of steps into a single one, and repeating the same procedure until the threshold is reached. While *refine* leaves the precision of an abstraction unchanged, the *compact* operator induces some loss of precision, since it merges together some steps.

Let $f = \bigwedge_{0 \leq i \leq N} \{t_i : \mathrm{v_i}\}$ be an abstract state, composed by $N+1$ steps, and let M be the threshold to reach, with $M < N+1$. The algorithm (i) chooses the step with the minimum width ($w_i = t_{i+1} - t_i$), (ii) merges it with the successive one, and (iii) repeats (i) and (ii) iteratively until the threshold M is reached. We choose the step to be merged as the smallest one (i.e., the one with the smallest width), but this choice is arbitrary: alternative solutions are possible (for example considering the similarity of values of successive steps) and can be supported by our approach as well. Another possibility would be to minimize the difference between the slopes and intercepts of corresponding lines, that is, the following quantity:

$$(|m_i^+ - m_{i+1}^+|) + (|q_i^+ - q_{i+1}^+|) + (|m_i^- - m_{i+1}^-|) + (|q_i^- - q_{i+1}^-|)$$

As for the creation of the merged step, let A_i, B_i be the two extremes (the left and right ones, respectively) of f_i^+ in $[t_i, t_{i+1}]$, and let A_{i+1}, B_{i+1} be the two extremes of f_{i+1}^+ in $[t_{i+1}, t_{i+2}]$. Then the upper side f'^+ of the merged step will have the slope of the

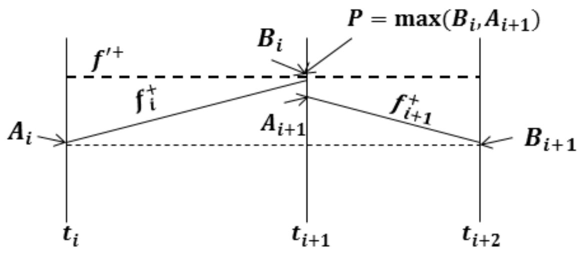


Fig. 14. Merging of two steps within the *compact* operation

side linking A_i and B_{i+1}. If the point $P = \max(B_i, A_{i+1})$ is greater than such side, the intercept will be such that the side covers exactly P, otherwise it is kept the original intercept of the side linking A_i and B_{i+1}. Figure 14 depicts this situation. The same applies symmetrically for the lower side: we consider the minimum between the two points at t_{i+1} and we check if such point is lower than the side linking the two extremes at t_i and t_{i+2}.

A slightly different process is required if the selected step is next to last (that is, $i = N - 1$), since in such case we cannot rely on t_{i+2}. For the upper side, we consider f_N^+ and we increase its intercept if one of the extremes of f_{N-1}^+ in $[t_{N-1}, t_N]$ is greater than such side. The same procedure applies symmetrically for the lower side.

In addition, we can specify a list of steps which we do *not* want to remove from the state. Let T be the set of steps of the abstract state f, and let $X \subseteq T$ be the set of steps of f that have to be preserved. Obviously, if M is the number of steps we want in the resulting abstract state, $|X| \leq M$ holds,. Then, $g = Compact_X(f, M)$ is an abstract state obtained by compacting f to M steps, and discarding only steps coming from $T \setminus X$. The algorithm presented above can be applied in this case as well by considering only the steps in $T \setminus X$ when selecting the steps to remove.

LEMMA 2.9 (SOUNDNESS OF *compact*). *Let f be $\in D^\sharp$ and M be $\in \mathbb{N}$. If $g = Compact(f, M)$, then $f \subseteq^\sharp g$.*

PROOF. Let N be the number of steps of the abstract state f, where $N > M$. In order to prove the proposition above, we have to prove that $f \subseteq^\sharp Reduce(f, N - 1)$, that is, the execution of one step of the reduction, going from N steps to $N - 1$. If the reduction of one step decreases the size of the abstract function, then our proposition is proved to be valid, since the reduction function works by reducing repeatedly one step at a time (being equivalent to an iterative process) and the ordering is transitive. In fact, from $f = Compact(f, N) \subseteq^\sharp Compact(f, N-1) \subseteq^\sharp \ldots \subseteq^\sharp Compact(f, M+1) \subseteq^\sharp Compact(f, M) = g$, we derive (by transitivity of the partial ordering) $f \subseteq^\sharp g$.

Let i be the index of the step to be removed, and suppose that $i < N - 2$ (that is, the last step is not involved in the reduction; that case can be trivially proved with slight modifications to this proof). The steps we are joining together are then f_i and f_{i+1}; the result will be the step g_i of the new abstract function. Since the steps before and after i remain unchanged, in order to prove that $f \subseteq^\sharp g$ we just need to prove that $\forall t \in [t_i, t_{i+1}] : [f_i^-(t), f_i^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$ and $\forall t \in [t_{i+1}, t_{i+2}] : [f_{i+1}^-(t), f_{i+1}^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$. We consider the two cases separately, first $[f_i^-(t), f_i^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$ and then $[f_{i+1}^-(t), f_{i+1}^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$.

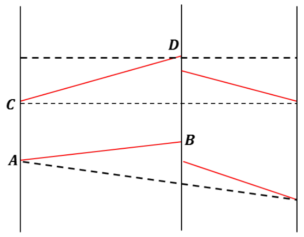


Fig. 15. Notation

We use the following notation, referring to a generic step $[t_i, t_{i+1}]$: $A = f_i^-(t_i)$, $B = f_i^-(t_{i+1})$, $C = f_i^+(t_i)$, $D = f_i^+(t_{i+1})$. See Figure 15 for a graphical representation of this notation.

— ($[f_i^-(t), f_i^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$) For the upper side, we have two distinct cases:
  — In the first one, the upper side passes for C, and D is below such side; then trivially $\forall t \in [t_i, t_{i+1}] : f_i^+(t) \leq g_i^+(t)$.
  — Otherwise, the upper side passes for D, and C is below such side (since the upper side is produced by lifting up the side passing for C, so C necessarily remains below; this case is depicted in Figure 15); in this case we have $\forall t \in [t_i, t_{i+1}] : f_i^+(t) \leq g_i^+(t)$ as well.

  The same can be said for the lower side: or the side that passes through A and B is above, or the side passes for B and A is above (since the side passing for A has been lowered). In both cases, we have that $\forall t \in [t_i, t_{i+1}] : f_i^-(t) \geq g_i^-(t)$. Combining the two equations we obtain that $\forall t \in [t_i, t_{i+1}] : (f_i^-(t) \geq g_i^-(t) \wedge f_i^+(t) \leq g_i^+(t))$ and this, combined with the constraint of Equation 1 (f_i^- is always below f_i^+ in step i-th), implies that $\forall t \in [t_i, t_{i+1}] : [f_i^-(t), f_i^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$.

— ($[f_{i+1}^-(t), f_{i+1}^+(t)] \subseteq [g_i^-(t), g_i^+(t)]$) this case is symmetrical to the previous one.

□

2.10. Widening

The widening operator is parameterized on (i) k_S, corresponding to the maximum number of steps allowed in an abstract state, (ii) k_M, k_Q, corresponding to the maximum value allowed for the slope and intercept of trapezoid sides respectively, and (iii) k_I, k_L, corresponding to the increment constants for the slope and intercept respectively. All these parameters have to be positive. Thanks to these parameters, we can tune the widening operators at different levels of precision and efficiency.

The widening operator $\nabla_{D^\sharp}$ is then defined as follows.

$$\nabla_{D^\sharp} : (D^\sharp, D^\sharp) \to D^\sharp$$
$$f \nabla_{D^\sharp} g = \begin{cases} \top^\sharp & \text{if } |U| > k_S \\ f & \text{if } g \subseteq^\sharp f \\ Norm(Compact_U(h_{MQ}, k_S)) & \text{otherwise} \end{cases}$$

where U is the step set of the abstract state f.

We distinguish three cases:

$|U| > k_S$: f exceeds the maximum number of steps allowed in an abstract state, k_S, and we return $\top^\sharp$.

$g \subseteq^\sharp f$: We do not have an ascending chain and we simply return f, which is already normalized, being an element of $D^\sharp$.

Otherwise: We return the normalized and compacted version of h_{MQ}, keeping all the steps U of f (we know that $|U| \leq k_S$, otherwise we would have returned $\top^\sharp$). In this way, we are sure that U will be a subset of the step set of the result $(f \nabla_{D^\sharp} g)$. The abstract state h_{MQ} is built as follows. Let f be defined on the indices set U, and g be defined on the indices set V. Let $f' = Refine(f, V)$ be the refined version of f with the addition of the indices of g, and $g' = Refine(g, U)$ the refined version of g with the addition of the indices of f. Then f' and g' are defined on the same set of steps $T = U \cup V$. So we have that $f' = \bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{v_i} = (f_i^-, f_i^+)\}$ and $g' = \bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{w_i} = (g_i^-, g_i^+)\}$. We define h_{MQ} as:

$$h_{MQ} = \bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{z_i} = (h_i^-, h_i^+)\}$$

where (h_i^-, h_i^+) are defined as follows:

$$h_i^-(x) = \begin{cases} g_i^-(x) & \text{if } f_i^- = g_i^- \\ -\infty & \text{if } (m_{g_i^-} \leq -k_M) \vee (q_{g_i^-} \leq -k_Q) \vee (m_{f_i^-} \leq -k_M) \vee (q_{f_i^-} \leq -k_Q) \\ (g_i^-)^\bullet(x) & \text{otherwise} \end{cases}$$
$$h_i^+(x) = \begin{cases} g_i^+(x) & \text{if } f_i^+ = g_i^+ \\ +\infty & \text{if } (m_{g_i^+} \geq k_M) \vee (q_{g_i^+} \geq k_Q) \vee (m_{f_i^+} \geq k_M) \vee (q_{f_i^+} \geq k_Q) \\ (g_i^+)^\circ(x) & \text{otherwise} \end{cases}$$

and

$$\begin{aligned} (g_i^-)^\bullet(t) &= (m_{MIN_i^-} - k_I) \times t + (q_{MIN_i^-} - k_L) \\ (g_I^+)^\circ(t) &= (m_{MAX_i^+} + k_I) \times t + (q_{MAX_i^+} + k_L) \\ m_{MIN_i^-} &= \min(m_{f_i^-}, m_{g_i^-}) \\ q_{MIN_i^-} &= \min(q_{f_i^-}, q_{g_i^-}) \\ m_{MAX_i^+} &= \max(m_{f_i^+}, m_{g_i^+}) \\ q_{MAX_i^+} &= \max(q_{f_i^+}, q_{g_i^+}) \end{aligned}$$

The computation is symmetric for the lower and upper side, so let us focus on $h_i^+(x)$. For each step t_i of f' and g' we consider three distinct cases:

— $f_i^+ = g_i^+$: the side is the same in f' and g', so we keep it unchanged.

— $(m_{g_i^+} \geq k_M) \vee (q_{g_i^+} \geq k_Q) \vee (m_{f_i^+} \geq k_M) \vee (q_{f_i^+} \geq k_Q)$: the slope (or the intercept) of the side of one abstract state (g' or f') exceeds the threshold k_M (k_Q), so we move the side to $+\infty$.

— Otherwise: we keep the maximum slope and intercept between their values in f_i^+ and g_i^+ and then we increase them both by a predefined constant quantity (k_I for the slope, k_L for the intercept).

Intuitively, the convergence is guaranteed by the combination of:

— the application of *compact* with the parameter k_S;

— the parameters k_M and k_Q that limit the maximal values allowed for the slope and the intercept of a line, respectively. If a certain line exceeds one of these two values, then it goes to $\pm\infty$, stopping its possible growth in that direction;

— the $\bullet$ and $\circ$ operators, that shift a side down ($\bullet$) and up ($\circ$) for a predefined amount respectively. In particular, referring to $\circ$, the maximum slope $m_{MAX_i^+}$ between the ones of f_i^+ and g_i^+ is increased by k_I, while the maximum intercept $q_{MAX_i^+}$ between the one of f_i^+ and g_i^+ is increased by k_L. k_I and k_L are positive values and ensure that we will reach the convergence in a finite number of steps.

For the soundness of this operator we refer to the definition of [Cortesi 2008; Cortesi and Zanioli 2011], through the two properties of covering and termination.

LEMMA 2.10 (CORRECTNESS OF $\nabla_{D^\sharp}$). *The widening operator $\nabla_{D^\sharp}$ is correct, that is, it respects the properties of covering and termination:*

(1) *Covering: $\forall f, g \in D^\sharp : f \subseteq^\sharp (f \nabla_{D^\sharp} g) \wedge g \subseteq^\sharp (f \nabla_{D^\sharp} g)$*
(2) *Termination: for every ascending chain $\{f_j\}_{j\geq 0}$, the ascending chain defined as $g_0 = f_0; g_{j+1} = g_j \nabla_{D^\sharp} f_{j+1}$ stabilizes after a finite number of terms.*

PROOF.

Covering By definition of $\nabla_{D^\sharp}$, we have three possible cases:
(1) If $|U| > k_S$ we return $\top^\#$, and trivially $f \subseteq^\sharp \top^\# \wedge g \subseteq^\sharp \top^\#$
(2) If $g \subseteq^\sharp f$ then we return f and trivially $f \subseteq^\sharp f$ by the reflexivity property of the partial order and $g \subseteq^\sharp f$ by hypothesis.
(3) In the last case, suppose that the two functions are defined on the same step set (we can obtain that with the *refine* operator, without changing their ordering by Lemma 2.4), $f = \bigwedge_{0\leq i\leq N}\{t_i : \mathbf{v_i}\} \wedge g = \bigwedge_{0\leq i\leq N}\{t_i : \mathbf{w_i}\}$. Let $h = \bigwedge_{0\leq i\leq N}\{t_i : \mathbf{z_i}\}$ be the result of the widening of f and g. We must prove that $\forall i, \mathbf{w_i} \sqsubseteq_{[t_i,t_{i+1}]} \mathbf{z_i} \wedge \mathbf{v_i} \sqsubseteq_{[t_i,t_{i+1}]} \mathbf{z_i}$. Consider first $\mathbf{w_i} \sqsubseteq_{[t_i,t_{i+1}]} \mathbf{z_i}$. By definition of $\sqsubseteq_{[t_i,t_{i+1}]}$, we have that $\mathbf{w_i} \sqsubseteq_{[t_i,t_{i+1}]} \mathbf{z_i} \Leftrightarrow [g_i^-(t_i), g_i^+(t_i)] \subseteq [h_i^-(t_i), h_i^+(t_i)] \;\wedge\; [g_i^-(t_{i+1}), g_i^+(t_{i+1})] \subseteq [h_i^-(t_{i+1}), h_i^+(t_{i+1})] \Leftrightarrow g_i^-(t_i) \geq h_i^-(t_i) \wedge g_i^+(t_i) \leq h_i^+(t_i) \;\wedge\; g_i^-(t_{i+1}) \geq h_i^-(t_{i+1}) \;\wedge\; g_i^+(t_{i+1}) \leq h_i^+(t_{i+1})$. Regarding $h_i^-(t)$, we know that it could be:
— $h_i^-(t) = g_i^-(t)$. Then $\forall t \in [t_i, t_{i+1}] : g_i^-(t) \geq g_i^-(t)$
— $h_i^-(t) = -\infty$. Then $\forall t \in [t_i, t_{i+1}] : g_i^-(t) \geq -\infty$
— $h_i^-(t) = (g_i^-)^\bullet(t)$ that is $h_i^-(t) = (m_{MIN_i^-} - k_I) \times t + (q_{MIN_i^-} - k_L) = m_{MIN_i^-} \times t - k_I \times t + q_{MIN_i^-} - k_L$. Since k_I, k_L are both ≥ 0 by definition and $m_{MIN_i^-} \leq m_{g_i^-} \wedge q_{MIN_i^-} \leq q_{g_i^-}$ also by definition, we can say that $h_i^-(t) = m_{MIN_i^-} \times t + q_{MIN_i^-} - k_I \times t - k_L \leq m_{g_i^-} \times t + q_{g_i^-} = g_i^-(t)$ holds $\forall t \geq 0$.

The same happens symmetrically for $h_i^+(x)$. We have then proved that $\forall i : \mathbf{w_i} \sqsubseteq_{[t_i,t_{i+1}]} \mathbf{z_i}$. The same reasoning can be made considering f instead of g, ending in proving that $\forall i : \mathbf{v_i} \sqsubseteq_{[t_i,t_{i+1}]} \mathbf{z_i}$. Thus $g \subseteq^\sharp h \wedge f \subseteq^\sharp h$.

Termination We want to prove that the ascending chain $\{g_j\}_{j\geq 0}$ stabilizes after a finite number of terms. Consider a generic element of the chain, g_j. Since the chain is ascending, $g_{j-1} \subseteq^\sharp g_j$. If $g_{j-1} = g_j$, the chain has stabilized and the convergence is reached. Otherwise, we have that $g_{j-1} \subset^\sharp g_j$. By definition of $\subseteq^\sharp$, this means that the area of (at least) one trapezoid of g_j has increased with respect to the corresponding trapezoid of g_{j-1}. Let (s_i^-, s_i^+) be the lower and upper sides of such trapezoid in g_j. The area of the step is increased if the two lines (s_i^- and s_i^+) drift apart with respect to their values at the preceding iteration, that is, in g_{j-1}. In particular, or s_i^- went down (that is, the slope, the intercept or both decreased), or s_i^+ went up (that is, the slope, the intercept or both increased). By definition of widening, we know that the lines will stop increasing or decreasing when their slope or intercept reach the

threshold values (k_M, k_Q) they will be approximated by $+\infty$ or $-\infty$. We also know that the growth in the number of steps is limited to k_S, and that the steps of the previous value in the chain will be preserved (the widening only adds indices, but it does not remove them).
Suppose that the chain ascends of the minimum possible quantity (with respect to our parameters k_I, k_L and the float numbers representation of the computer).
Let us define

$$maxIterationsToStopGrowth(i) = \min(\frac{|m_i^+ - k_M|}{k_I}, \frac{|q_i^+ - k_Q|}{k_L}) + \min(\frac{|m_i^- - k_M|}{k_I}, \frac{|q_i^- - k_Q|}{k_L})$$

This value is an upper bound of the number of iterations needed for step t_i to reach the threshold after which it stops growing. In particular, consider a generic step t_i and the increase of its upper side. Since the slope is increased of at least k_I and, at the same time, the intercept is increased of at least k_L, the side will stop to grow after, at most, $\min(\frac{|m_i^+ - k_M|}{k_I}, \frac{|q_i^+ - k_Q|}{k_L})$ iterations (after that, the side is set to $+\infty$, its maximum value). The same happens for the lower side, which stops to grow after, at most, $\min(\frac{|m_i^- - k_M|}{k_I}, \frac{|q_i^- - k_Q|}{k_L})$ iterations. Thus, each step t_i is abstracted to top after at most $\min(\frac{|m_i^+ - k_M|}{k_I}, \frac{|q_i^+ - k_Q|}{k_L}) + \min(\frac{|m_i^- - k_M|}{k_I}, \frac{|q_i^- - k_Q|}{k_L})$ iterations, that is, the quantity $maxIterationsToStopGrowth(i)$.
Let $g = \bigwedge_{i \in [0,N]} \{t_i : \mathbf{v_i} = (m_i^-, q_i^-, m_i^+, q_i^+)\}$ be a generic element of the ascending chain $\{g_j\}$. Then, an upper bound of the maximum number of iterations needed to converge is:

$$\begin{aligned} K = & \textstyle\sum_{i=0}^{N} maxIterationsToStopGrowth(i) + \\ & +(k_S - N) \times maxIterationsToStopGrowth(iMAX) \end{aligned} \tag{5}$$

where $iMAX = \{i \in [0, N] : \nexists j \in [0, N] : maxIterationsToStopGrowth(j) > maxIterationsToStopGrowth(i)\}$ is the step index which maximizes the quantity $maxIterationsToStopGrowth$.
Equation 5 is composed by two parts:

— $\sum_{i=0}^{N} maxIterationsToStopGrowth(i)$ represents an upper bound of the number of iterations needed for *all* the steps of g to reach the threshold after which they stop growing. In fact, at each iteration of the widening, at least one side of one step is moved up or down and this means that its slope and intercept move closer to the thresholds k_M, k_Q. Each step t_i of g will stop to grow after, at most, $maxIterationsToStopGrowth(i)$ iterations. Summing this quantity for all the N steps of g, we obtain the first part of Equation 5.

— $(k_S - N) \times maxIterationsToStopGrowth(iMAX)$ represents an upper bound of the number of iterations needed for all the (possibly) added steps to reach the threshold after which they stop growing. In fact, we know that the maximum number of steps allowed in an abstract state by the widening is k_S. The state g is composed by N steps, so at most $k_S - N$ steps could be added to g along the chain. Nevertheless, we know that the chain $\{g_j\}$ is increasing, so these added steps cannot have values that are smaller (i.e., far from the thresholds that stop the growth) than the already existing values in g. In order to have an upper bound of the iterations needed by these added step to stop growing, we then suppose that all these steps lie within the step of g which is farthest from the growth stop. More precisely, let $iMAX$ be the index of the step which maximizes the quantity $maxIterationsToStopGrowth(\cdot)$. Then, each added step stops to grow after, at most, $maxIterationsToStopGrowth(iMAX)$ iterations. Multiplying this value by

$k_S - N$ (the maximum number of steps we can add to g) we obtain the second part of Equation 5.

Since K is the result of the summation of two finite quantities, we can say that the ascending chain $\{g_j\}_{j\geq 0}$ stabilizes after a finite number of terms, which is at most K.

□

2.11. The lattice $D^\sharp$

THEOREM 2.11. *$(D^\sharp \cup \{\bot^\sharp\}, \subseteq^\sharp, \bot^\sharp, \top^\sharp, \cup^\sharp, \cap^\sharp)$ is a lattice.*

PROOF. By Lemma 2.3 we get that $\subseteq^\sharp$ is a partial order, by Lemma 2.7 that $\cup^\sharp$ is the least upper bound operator, by Lemma 2.6 that $\cap^\sharp$ is the greatest lower bound operator. □

We also proved the correctness of the widening operator $\nabla_{D^\sharp}$ in Lemma 2.10. Therefore, we have defined and proved the correctness of all the operators required to define an abstract domain in the abstract intepretation framework [Cousot and Cousot 1977; 1979].

3. ABSTRACTION OF A CONTINUOUS FUNCTION

In this section, we show how to compute the approximation of $\mathcal{C}^2_+$ functions in IVSF and TSF. We consider IVSF as well since [Bouissou and Martel 2008] did not define its abstraction function, because they relied on a particular type of ODE solver [Bouissou and Martel 2007]. For both domains, we consider two different approaches: when the step width is constant and fixed, and when we automatically determine the step distribution. Note that we abstract only one concrete function; this approach can be generalized to the abstraction of a countable set of concrete functions C by computing the abstraction of each function in the set and then returning the least upper bound of all the resulting abstract states.

In the following subsections, we will denote by:

— $f \in \mathcal{C}^2_+$ the continuous function we want to abstract;
— f', f'' its first and second derivatives;
— F'_0 the set containing the points of the domain where $f'(x) = 0$ (the minimum and maximum points of the function), that is, $F'_0 = \{t : f'(t) = 0\}$;
— F''_0 the set containing the points of the domain where $f''(x) = 0$ (the inflection points), that is, $F''_0 = \{t : f''(t) = 0\}$;
— $G'^{[a,b]}_0$ the set containing the maximum and minimum points of f restricted to the domain interval $[a, b]$, that is, $G'^{[a,b]}_0 = \{f(t) : t \in ([a, b] \cap F'_0)\}$;
— $G''^{[a,b]}_0$ the same set but for the inflection points, that is, $G''^{[a,b]}_0 = \{f(t) : t \in ([a, b] \cap F''_0)\}$.

3.1. IVSF abstraction function, fixed step width

Given a fixed step width w, suppose that $[a, b]$ is a generic interval ($b - a = w \wedge a = k \times w \wedge b = (k+1) \times w \wedge k \geq 0, w > 0$). $M = \max(\{f(a), f(b)\} \cup G'^{[a,b]}_0)$ is the maximum point of the function in the interval $[a, b]$, extremes included, and $m = \min(\{f(a), f(b)\} \cup G'^{[a,b]}_0)$ is the minimum point of the function in the interval $[a, b]$, extremes included. The best abstraction in IVSF of this step is the interval $[m, M]$. To build the abstraction of the function f, we repeat this procedure for each step of the abstract state.

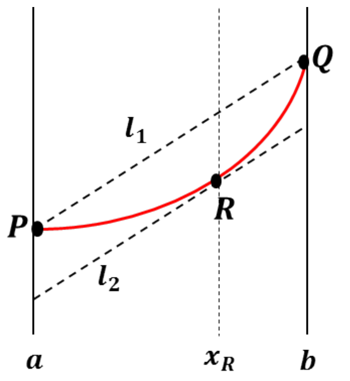


Fig. 16. The abstraction on the step $[a, b]$

3.2. IVSF abstraction function, arbitrary step width

For the IVSF abstract domain, we cannot find *a priori* the best way to split the domain of f in sub-intervals. We could use different techniques, for example:

— splitting in correspondence of the inflection points of the function, and
— splitting in correspondence of the intermediate point between each pair of minimum/maximum points.

There is no best choice that works in any case, since this choice depends on the shape of the function. In some cases it is more precise to use the inflection points for splitting, in other cases the intermediate points yield more precision. In addition we could split also in correspondence of the minimum/maximum points; sometimes (more often than not) this increases the precision of the representation.

Once chosen a technique for splitting the domain in steps, we can apply the procedure introduced in Section 3.1 to compute the abstraction of each single step.

3.3. TSF basic abstraction function, arbitrary step width

In TSF a very good trade-off between complexity and precision of the abstraction can be achieved by splitting the domain in correspondence of (i) the maximum and minimum points F'_0, and (ii) the inflection points F''_0.

Assume that $[a, b]$ is a generic sub-interval obtained using this schema. Then the two sides which compose the value of such step are the following ones (see Figure 16):

(1) the side l_1 linking the points $P = (a, f(a))$ and $Q = (b, f(b))$ (the points of f in correspondence of the extremes of the interval $[a, b]$).
(2) the side l_2 which has the same slope as l_1 and is tangent to f inside $[a, b]$. Since we already know the slope of this side, we just need to compute its intercept. The procedure is the following one:
   (a) find the point $x_R \in [a, b]$ where the first derivative of f is equal to the slope of l_1: $f'(x_R) = m_{l_1}$. This point can be computed by bisection in $[a, b]$.
   (b) let R be the point with coordinates $(x_R, f(x_R))$. Then l_2 is the side that goes through the point R and with slope equal to the one of l_1 ($m_{l_2} = m_{l_1}$). The intercept is computed as follows: $q_{l_2} = f(x_R) - m_{l_2} \times x_R$.

Note that the resulting sides l_1 and l_2 are parallel, as they have the same slope. Moreover, $l2$ is a tangent of f.

3.4. TSF basic abstraction function, fixed step width

Also in the case of TSF we can define the abstraction on a fixed step width. Suppose that $[a, b]$ is a generic interval determined by a fixed width w. First of all, we split the interval in sub-intervals, following the schema introduced in Section 3.3. Then for each sub-interval, we compute the upper and lower sides as specified in Section 3.3. Finally, we have to "join" these sub-intervals into a single one (with range $[a, b]$) through the *compact* operator (see Section 2.9).

Observe that the abstraction function of IVSF is less restrictive than the one of TSF, since TSF's abstraction needs the second derivative of the function to know the inflection points.

3.5. Dealing with floating point precision issues in TSF

Unfortunately, the abstraction technique presented in Section 3.3 is theoretically sound but it is not computable on a finite precision machine, due to the rounding issues of floating point representation. The abstraction function depends on various values: the stationary points (F'_0), the inflection points (F''_0), the point $x_r \in [a, b]$ such that $f'(x_R) = m_{l1}$. Even knowing exactly all the points in F'_0 and F''_0 by mathematical analysis, we could not be able to precisely represent them in a machine (e.g., $\sqrt{2}$). Therefore, we can only compute an approximation of such points and not their exact value. In this section, we introduce some restrictions on the functions we can manipulate and a refinement of the basic abstraction function proposed in Section 3.3 to enforce the soundness of the resulting abstraction function in a floating point computation.

We assume that f respects the following property. If x_0 is a point such that $f'(x_0) = 0$, then, for each interval $[\overline{x}, \overline{x} + \epsilon]$ such that $x_0 \in [\overline{x}, \overline{x} + \epsilon]$, we have:

$$\forall x \in [\overline{x}, \overline{x} + \epsilon] : f(x) \in [f(\overline{x}) - \tau, f(\overline{x}) + \tau]$$

where τ is a parameter of the analysis and ϵ is a constant depending on the machine in use. Intuitively, we ask that function's values change at most of τ around stationary points. We impose the same constraint on the inflection points (x s.t. $f''(x) = 0$). Note that the value of ϵ can be set based on the standard in use on the machine (for instance, IEEE 754 for floating points), while τ has to be set by the user: the smaller the value, the more precise the abstraction.

Like in Section 3.3, we split the domain in steps with respect to the stationary points (F'_0) and the inflection points (F''_0). If we cannot pinpoint those points exactly, we introduce an additional step of width ϵ in correspondence of them. The exact location of the step $[t_i, t_{i+1}] = [\overline{x}, \overline{x} + \epsilon]$ depends on the numerical representation of the machine and it obviously must contain the exact value of the considered stationary or inflection point. Intuitively, $\overline{x}$ will be the greatest representable under-approximation of the stationary/inflection point. The value of such additional step is $\mathbf{v_i} = (0, f(\overline{x}) - \tau, 0, f(\overline{x}) + \tau)$. For the condition imposed above, we are sure that this trapezoid (which is a rectangle, since the two sides are horizontal) soundly contains the abstracted function in the considered step.

Let us see a simple example to better understand how these kinds of steps are computed. Consider a function f in the restricted domain $[0, 4]$. Assume we know by mathematical analysis that, in such domain, the function has only one stationary point (in correspondence of $x = \pi = 3.14159\ldots$) and no inflections points. More formally, we know that: $G'^{[0,4]}_0 = \{\pi\}$ and $G''^{[0,4]}_0 = \emptyset$. Suppose that the machine in use has a precision such that $\epsilon = 0.0001$. This means that we can represent values like $0, 0.0001, 0.0002, \ldots, 3.1415, 3.1416, \ldots$. So, we are not able to represent any other number between 3.1415 and 3.1416. Since $3.1415 \leq \pi \leq 3.1416$, we choose $\overline{x} = 3.1415$. Then

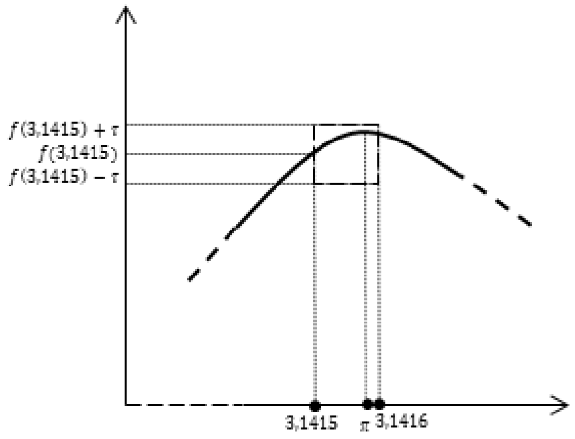


Fig. 17. Creation of the step around stationary point π

we have $[\overline{x}, \overline{x} + \epsilon] = [3.1415, 3.1416]$. The resulting step function constraint around the stationary point π is

$$t_i = 3.1415 : \mathbf{v_i} = (0, f(3.1415) - \tau, 0, f(3.1415) + \tau)$$

where τ is a parameter given by the user and specific with respect to the function which we want to abstract. We have created a step that starts at 3.1415 and whose trapezoid lower and upper sides are horizontal lines at the height of $f(3.1415) - \tau$ and $f(3.1415) + \tau$, respectively. We can see this step depicted in Figure 17. In this case the trapezoid is a rectangle. The next step will start at $t_{i+1} = 3.1416$.

For the steps that do not contain stationary and inflection points, the computational schema of Section 3.3 is refined as follows. The side l_1 (the one which links the extremes of f in the step) is moved up (or down, depending on the concavity of f in the step) of ϵ. This compensates for potential errors in the evaluation of the function values at the extremes. The other side l_2 goes through the point $x_R{}'$ such that $f'(x_R{}')$ is the closest value to m_{l1} that we can reach (given the precision of the machine). The slope of l_2 is $f'(x_R{}')$ so that l_2 is tangent to the function. Since we know that the function is concave (or convex) in the considered subinterval, we are sure that one of its tangents leaves the function always above (or below), resulting in a safe approximation. This refined computational schema is represented in Figure 18.

The round dotted lines l_1 and l_2 represent the final result of the computation. The intermediate process is as follows. First, we create the dashed line joining $P = (a, f(a))$ and $Q = (b, f(b))$, then we move it up of ϵ, resulting in l_1. Then, since we cannot find the dashed line going through the point R (a line tangent to the function and having the same slope as l_1) because of rounding approximation, we use instead the round dotted line going through R', that is, the closest point to R with respect to the machine precision. The slope of the line l_2 is $f'(x_{R'})$, that is, l_2 is tangent in R' to the function f being abstracted.

3.6. Dealing with floating point precision issues in IVSF

The same issues about floating point rounding issues arise for the abstraction function presented in Section 3.1. In this case, though, it is very easy to make the abstraction sound and we do not need to impose any requirement on the function to abstract. After having determined the minimum and maximum values (m and M) assumed by

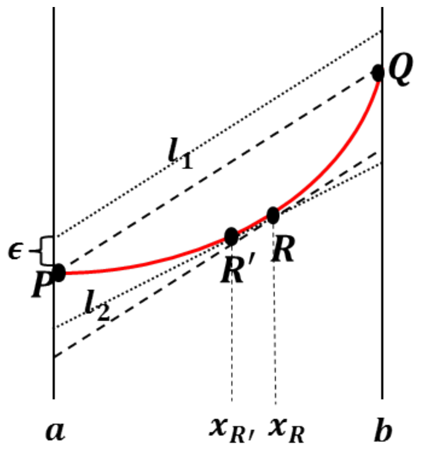


Fig. 18. Creation of steps without stationary/inflection points

the function in the considered step, it is sufficient to return $[m - \epsilon, M + \epsilon]$ (instead of $[m, M]$) as step value. This compensates for possible approximation errors due to the finite precision of the machine.

4. EXPERIMENTAL RESULTS

In this section [1] we present some experimental results about the use of TSF, and we compare them with the ones obtained by IVSF. First of all, we explore how the precision of TSF varies with the number of steps of the representation when analyzing some representative functions. Then we consider a standard example of embedded software (introduced in Section 1.1) as a test-bench.

4.1. Varying the number of steps

Let us first compare the precision of TSF and IVSF. We apply the abstraction function to a set of representative functions (namely, $\sin(x)$, x^3, e^x, and $\ln(x+1)$[2]) in the interval $[0, 10]$ varying the number of steps. We measure the precision of a representation by computing the area covered by the abstract states in the Cartesian plan: the wider the area, the rougher the abstraction. Table 4.1 reports the results of this computation. The first column reports the number of steps. Then, for each analyzed function, we report the area of the TSF and IVSF abstractions, and the ratio between the two areas. For instance, if the ratio is 50%, it means that the TSF area is half of the IVSF one (i.e., it is twice more precise).

[1]**TODO GIULIA: Non ho riletto la sezione, sia perche' l'avevo in parte scritta io, sia perche' prima penso sia necessario rivedere un attimo i risultati sperimentali. A questo punto, sarebbe il caso di vedere se ci sono librerie Java in giro che data una funzione ti ritornano derivata prima e seconda e i punti in cui e' zero. Usando tale libreria, potremmo calcolare automaticamente l'astrazione e avere dati sperimentali sound come abbiamo descritto nell'articolo. Volendo si potrebbe anche fare un'interfaccia web per dare l'opportunita' ai reviewer di usare TSF in questo caso. Qui sarebbe bello avere una libreria per fare un jpg/gif/pdf/eps/... con il grafico della funzione originaria e di TSF. Che ne dici? Ci vorrebbe un po' per queste cose ma se ci sono librerie che fanno ste cose non ci porterebbero via troppo tempo e sarebbe un buon contributo (anche per la tua tesi).**

[2]Note that, since $\ln(x)$ is not continuous in $x = 0$, we apply it to $x + 1$ in order to have a continuous function in the domain $[0, 10]$

Table I. Precision of TSF and IVSF varying the steps number

#s	$\sin(x)$			x^3			e^x			$\ln(x+1)$		
	TSF	IVSF	Ratio	TSF	IVSF	Ratio	TSF	IVSF	Ratio	TSF	IVSF	Ratio
4	4.14	15.10	27.4%	235.04	2500.00	9.4%	15894.07	55063.66	28.9%	0.63	5.99	10.5%
8	1.15	7.99	14.4%	58.64	1250.00	4.7%	4211.62	27531.83	15.3%	0.17	3.00	5.7%
16	0.30	4.01	7.6%	14.65	625.00	2.3%	1069.68	13765.92	7.8%	0.04	1.50	2.9%
32	0.08	2.04	3.7%	3.66	312.50	1.2%	268.50	6882.96	3.9%	0.01	0.75	1.5%
64	0.02	1.02	1.8%	0.92	156.25	0.6%	67.19	3441.48	2.0%	2.77E-03	0.37	0.7%
128	4.70E-03	0.51	0.9%	0.23	78.13	0.3%	16.80	1720.74	1.0%	6.93E-04	0.19	0.4%

We implemented the computation of TSF in Java and we ran it on an Intel Core 2 Quad CPU 2.83 GHz with 4 GB of RAM, running Windows 7, and the Java SE Runtime Environment 1.6.0_16-b01. The execution is always extremely fast: in the worst case (function e^x), TSF requires 40 msec to compute the approximation and the area of the function for *all* the different numbers of steps. This result is not particularly surprising since the computation mainly performs arithmetic operators for whom modern processors are quite efficient. Since the times of executions are so low, we could not notice any significant difference between TSF and IVSF even if we would expect that IVSF is faster. In addition, we did not notice any relevant memory consumption by the computation since it does not need to allocate any memory.

In all cases, TSF is more precise than IVSF. In the worst case, TSF is almost 4 times more precise (since the ratio is $\approx 29\%$) and this happens when using a small number of intervals (4). In the best case, it is approximately 330 times more precise (the ratio is 0.3%), and this happens when using a lot of intervals (128).

Why does the precision of TSF get more and more precise with respect to the one of IVSF when we increase the number of steps? IVSF uses rectangles to approximate portions of the curve, so its precision is greater when the curve is "flat" (i.e., similar to a horizontal line), while it is lower when the curve's slope is high. So, the amount of precision depends more on the kind of function than on the steps width. The precision of TSF, instead, does not depend on the curve slope, since the trapezoids are able to well approximate various kinds of slope. The precision of TSF depends only on how much the curve differs from a straight line *within a single step*. If in a single step the curve is similar to a straight line, then the error is near to zero; if in a single step the curve is very concave/convex then there is a lack of precision. When increasing the number of steps in a given domain, each step has a smaller width: for this reason, the bigger the number of steps, the more the function resembles a straight line in each single step (instead that a convex/concave curve) and the more the TSF precision increases. On the other hand, the decrease in a step width does not modify the slope of the curve in each step, and this is why the precision of IVSF does not increase as much as in the TSF domain.

4.2. An integrator

Consider the motivating example presented in Section 1.1. Table II reports the intervals of the values of `intgrx` computed by TSF and IVSF after 104 iterations of the `while` loop. The smaller the interval, the more precise the analysis. The last column reports the ratio between the widths of the two intervals. TSF obtains more precise results in all the cases. Note that augmenting the numbers of steps in the abstraction improves the precision of both domains, and the error ratio of TSF vs. IVSF stabilizes around 60% even if it is slightly better when augmenting the number of steps.

4.3. Combination of TSF with IVSF

In the two case studies, we have seen that the TSF domain is able to approximate more closely the shape of the abstracted function than IVSF. Moreover we noticed that our

Table II. Values computed by TSF and IVSF on `intgrx`

# steps	TSF	IVSF	**Ratio** (%)
4	[-1.0263, 1.7367]	[-4.8750, 4.8750]	28
8	[-0.2772, 0.3778]	[-0.4760, 0.4760]	69
16	[-0.0740, 0.0870]	[-0.1237, 0.1237]	65
32	[-0.0188, 0.0204]	[-0.0312, 0.0312]	63
64	[-0.0047, 0.0049]	[-0.0078, 0.0078]	62
128	[-0.0012, 0.0012]	[-0.0020, 0.0020]	61

abstraction gets more and more precise (with respect to IVSF) every time we increase the number of steps in the representation.

However, IVSF has the advantage to preserve the minimum and maximum values assumed by the function throughout its entire domain. Unfortunately, TSF does not preserve such information, since the trapezoids vertices might exceed these values.

Sometimes the right side of a trapezoid (defined on $[t_i, t_{i+1}]$) and the left side of the next one (defined on $[t_{i+1}, t_{i+2}]$) have an intersection made up by only one point: in these cases we know for sure which is the value assumed by the function in correspondence of t_{i+1} (the border between the two trapezoids). In some cases, this phenomenon happens also for IVSF, and not necessarily in correspondence of the same points as TSF. In Figure 1 you can see this happening at $t = \pi$ for both TSF and IVSF: the information we have is that the function has precisely value 0 in that point.

Finally, to compute the abstraction of IVSF is necessary to know only the first derivative (other than, obviously, the original function). TSF instead requires also the second derivative, in order to locate the inflection points of the function.

Since both domains track interesting information, it could be useful in some applications, especially the ones where the stationary points of the function have some relevancy, to consider the product of these two domains, by using their Cartesian or the reduced product. For instance, in the analysis of the integrator code presented in the previous subsection we can *precisely* abstract the values of $\sin(x)$ when it is at its maximum (or minimum) by taking the intersection of the values approximated by TSF (that computes that the values are greater or equal to 1 in the maximum, and less or equal than -1 in the minimum) and IVSF (that computes that the values are less or equal to 1 in the maximum, and greater or equal than -1 in the minimum).

5. RELATED WORK

The abstraction of continuous functions is not a new topic in static analysis, even if it has been only partially explored so far.

To the best of our knowledge, IVSF has been the first formalism to allow the integration of the continuous environment in an abstract interpretation of embedded software. A static analyzer based on that formalism has been implemented [Bouissou et al. 2009] in order to consider the interactions between the program and the physical environments on which it acts. The analyzer (HybridFluctuat) is based upon Fluctuat, a tool developed by CEA which aims at analyzing the numerical precision and stability of complex algorithms. However, as far as we know, IVSF theoretical framework has not been refined further since then. In Section 4 we compare extensively the precision of our approach with respect to IVSF showing that TSF is quite more precise than IVSF when abstracting a representative set of continuous functions.

A useful domain theoretic characterization of continuous function can be found in [Edalat and Lieutier 2004], but this work only describes the continuous functions at the concrete level, and there is nothing involving their sound abstraction.

[Feret 2004] introduced domain-specific abstract domains for digital filters, in the context of ASTREE [Blanchet et al. 2003], but did not provide a generic treatment of continuous functions and their abstraction.

As for hybrid systems, previous work on abstract interpretation-based strategies for such systems mainly involves the analysis of hybrid automata [Halbwachs et al. 1994; Henzinger and Ho 1995].

Regarding continuity analysis of programs, [Hamlet 2002] was the first to argue for a testing methodology for Lipschitz-continuity of software. [Reed and Pierce 2010] introduced a type system that verifies the Lipschitz-continuity of functional programs. This system does not handle control flow and does not consider any application other than differential privacy. [Chaudhuri et al. 2010] recently proposed a qualitative program analysis to automatically determine if a program implements a continuous function. The practical motivation they adduce is the verification of robustness properties of programs whose inputs can have small amounts of error and uncertainty. This work was further extended by [Chaudhuri et al. 2011] to quantify the robustness of a program to uncertainty in its inputs. Since they decompose the verification of robustness into the independent sub-problems of verifying continuity and piecewise robustness, they do not add new insights to the specific problem of program continuity with respect to [Chaudhuri et al. 2010]. Our treatment of continuous functions should be applicable to this particular setting (continuity of programs) as well.

A Trapezoid Step Function is a sequence of trapezoids, one for each step. But a TSF can be seen as a pair of PieceWise Linear (PWL) functions as well, where one PWL function bounds the approximated continuous functions from above and the other one bounds them from below. There exists an extensive literature about PWL functions, since they played an important role in approximation, regression and classification. One of the biggest problems concerns their explicit representation in a closed form [Chua and Kang 1977; Kahlert and Chua 1990]. Another important issue is to find a PWL approximation of a certain function to minimize or bound the overall area (or the distance in each point) between the original function and the approximation [Chou et al. 1993; Imai and Iri 1987; Tomek 1974]. Instead, our approach provides a *sound* approximation of a function rather than bounding the error of its representation.

6. CONCLUSION AND FUTURE WORK

In this paper we introduced the Trapezoid Step Functions domain, a new abstract domain aimed at approximating the behaviors of continuous environments. Following the abstract interpretation framework, we formally defined the elements of our domain together with the lattice and the widening operators, proving the soundness of our approach. We then introduced an abstraction function that, given a concrete function, builds up its abstract representation in TSF. Finally, we presented some experimental results about the precision of TSF, and we compared them with the ones obtained by the Interval Valued Step Functions domain (IVSF), that, at the best of our knowledge, is the most refined domain in the context of the abstraction of functions in continuous environments. The experimental results underline that TSF obtains abstractions that are more precise than the ones obtained by IVSF.

Future work: Given the encouraging experimental results obtained by TSF, we are planning to apply TSF to other case studies. First of all, we want to apply TSF to the approximation of the solutions of Ordinary Differential Equations (as done by IVSF). Then we want to explore how we could use TSF to approximate the values produced by a program (e.g., a simulator of the results given by sensors in embedded systems). In addition, we plan to develop some semantic operators over TSF (e.g., sum and product or other arithmetic operations) to apply it to the cost analysis of code [Albert et al. 2007]. In the context of cost analysis, it will probably be necessary to extend our approach to multivariate functions (here we only considered univariate ones) in order to allow more complex cost models. We could try to use our domain to verify the continuity of programs as well. In addition, it could be interesting to do a formal complexity analysis on the domain operations. Finally, an ambitious and possibly very interesting extension is to change the definition of the upper and lower sides of the trapezoids from straight lines into more complex functions (e.g., polynomials).

REFERENCES

ALBERT, E., ARENAS, P., GENAIM, S., PUEBLA, G., AND ZANARDINI, D. 2007. Cost analysis of java bytecode. In *Proceedings of ESOP '07*. LNCS. Springer-Verlag.

AUBIN, J. P. AND CELLINA, A. 1984. *Differential Inclusions: Set-Valued Maps and Viability Theory*. Springer-Verlag New York, Inc.

BLANCHET, B., COUSOT, P., COUSOT, R., FERET, J., MAUBORGNE, L., MINÉ, A., MONNIAUX, D., AND RIVAL, X. 2003. A static analyzer for large safety-critical software. In *Proceedings of PLDI '03*. ACM Press.

BOUISSOU, O., GOUBAULT, E., PUTOT, S., TEKKAL, K., AND VÉDRINE, F. 2009. Hybridfluctuat: A static analyzer of numerical programs within a continuous environment. In *Proceedings of CAV '09*. LNCS. Springer.

BOUISSOU, O. AND MARTEL, M. 2007. GRKLib: a guaranteed runge-kutta library. In *Follow-up of International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*. IEEE Press.

BOUISSOU, O. AND MARTEL, M. 2008. Abstract interpretation of the physical inputs of embedded programs. In *Proceedings of VMCAI '08*. LNCS. Springer.

BRESSAN, A. AND CORTESI, A. 1989. Directionally continuous selections in banach spaces. *Nonlinear Analysis, Theory, Methods and Applications 13,* 8, 987–992.

CHAUDHURI, S., GULWANI, S., AND LUBLINERMAN, R. 2010. Continuity analysis of programs. In *Proceedings of POPL '10*. 57–70.

CHAUDHURI, S., GULWANI, S., LUBLINERMAN, R., AND NAVIDPOUR, S. 2011. Proving programs robust. In *Proceedings of SIGSOFT FSE '11*. 102–112.

CHOU, F., WANG, C. M., AND CHENG, G. D. 1993. Optimal bounding of curves by continuous piecewise linear functions. *Engineering Optimization 21,* 4, 307–317.

CHUA, L. AND KANG, S. M. 1977. Section-wise piecewise-linear functions: Canonical representation, properties, and applications. *Proceedings of the IEEE 65,* 6, 915–929.

CORTESI, A. 2008. Widening operators for abstract interpretation. In *Proceedings of SEFM '08*. IEEE.

CORTESI, A. AND ZANIOLI, M. 2011. Widening and narrowing operators for abstract interpretation. *Computer Languages, Systems & Structures 37,* 1, 24–42.

COUSOT, P. AND COUSOT, R. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of POPL '77*. ACM.

COUSOT, P. AND COUSOT, R. 1979. Systematic design of program analysis frameworks. In *Proceedings of POPL '79*. ACM.

COUSOT, P. AND HALBWACHS, N. 1978. Automatic discovery of linear restraints among variables of a program. In *Proceedings of POPL '78*. ACM Press.

EDALAT, A. AND LIEUTIER, A. 2004. Domain theory and differential calculus (functions of one variable). *Mathematical. Structures in Comp. Sci. 14,* 6.

FERET, J. 2004. Static analysis of digital filters. In *Proceedings of ESOP '04*. LNCS. Springer.

GOUBAULT, E., MARTEL, M., AND PUTOT, S. 2006. Some future challenges in the validation of control systems. In *Proceedings of ERTS '06*.

HALBWACHS, N., PROY, Y.-E., AND RAYMOND, P. 1994. Verification of linear hybrid systems by means of convex approximations. In *Proceedings of SAS '94*. LNCS. Springer.

HAMLET, D. 2002. Continuity in sofware systems. In *Proceedings of ISSTA '02*. 196–200.

HENZINGER, T. A. AND HO, P.-H. 1995. A note on abstract interpretation strategies for hybrid automata. In *Proceedings of Hybrid Systems II*. LNCS. Springer.

IMAI, H. AND IRI, M. 1987. An optimal algorithm for approximating a piecewise linear function. *Journal of information processing 9,* 3, 159–162.

KAHLERT, C. AND CHUA, L. 1990. A generalized canonical piecewise-linear representation. *IEEE Transactions on Circuits and Systems 37,* 3, 373–383.

MINÉ, A. 2006. The octagon abstract domain. *Higher-Order and Symbolic Computation*.

REED, J. AND PIERCE, B. C. 2010. Distance makes the types grow stronger: a calculus for differential privacy. In *Proceedings of ICFP '10*. 157–168.

TOMEK, I. 1974. Two algorithms for piecewise-linear continuous approximation of functions of one variable. *IEEE Trans. Comput. 23,* 4, 445–448.

Received October 2012; revised ? ?; accepted ? ?