Linear Approximation of Continuous Systems with Trapezoid Step Functions

Giulia Costantini[1], Pietro Ferrara[2], and Agostino Cortesi[1]

[1] University Ca' Foscari of Venice, Italy
{costantini,cortesi}@dsi.unive.it
[2] ETH Zurich, Switzerland
pietro.ferrara@inf.ethz.ch

Abstract. We introduce a novel abstract domain for the safe approximation of continuous functions in the context of abstract interpretation-based static analysis. The key-idea is to represent $\mathcal{C}^2_+$ functions by a finite sequence of trapezoids. In this way, we get a strictly more precise approximation of the actual values with respect to existing approaches in the literature. Experimental results underline the effectiveness of the approach in terms of both precision and efficiency.

1 Introduction

Embedded software is composed by 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 modelled by volatile variables. The reliability of these systems is crucial: a single bug can produce catastrophic effects, and this is a relevant challenge for formal verification methods. 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 [11,12], the Interval domain [11] abstracts continuous systems with the minimal and maximal values a sensor can return at any time. To refine this approach, Bouissou and Martel [5] 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 maximal values the function could achieve during that period of time.

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 linear relationships between the time and the output value. 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 TSF better approximates the

R. Jhala and A. Igarashi (Eds.): APLAS 2012, LNCS 7705, pp. 98–114, 2012.
© Springer-Verlag Berlin Heidelberg 2012

shape of the function. 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, we could combine TSF and IVSF in a Cartesian product [11]. In the example of Figure 1, this combination discovers that, when $x = \frac{\pi}{2}$, 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.

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 continuous 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 show 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 2. This is the code of an integrator, a quite common component of embedded programs. It has been inspired by [15], and it is the example used in [5] 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

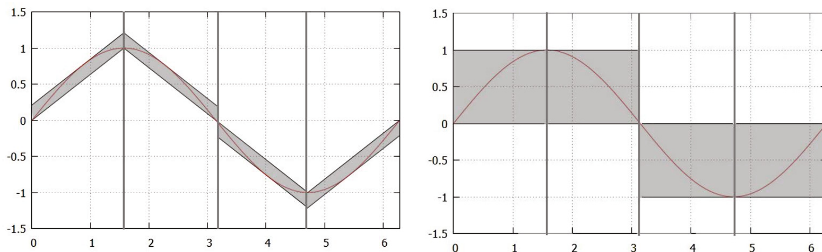


Fig. 1. TSF (left) and IVSF (right) abstractions of $\sin(x)$, with 4 steps, on the domain $[0, 2\pi]$

```
1 volatile float x;
2 static float intgrx=0.0, h=1.0/8;
3 void main() {
4         while(true) { // assume frequency = 8 KHz
5                 float xi=x;
6                 intgrx+=xi*h;
7         }
8 }
```

Fig. 2. Simple integrator

(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 is a framework to define and prove the soundness of approximations. The concrete domain formalizes the information we want to approximate, while the abstract domain specifies which approximated information we track. Usually, concrete states are composed by sets of elements (e.g., all the possible computational states), 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 [11]. 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 present the concrete domain. Then, we introduce TSF, with the partial order and the least upper bound operator, to show its lattice structure. Finally, we introduce a widening operator that is crucial to ensure 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 is defined as the powerset of continuous functions in $\mathbb{R}^+ \to \mathbb{R}$ which have two continuous derivatives (i.e., the set $\mathcal{C}^2_+$). We focus our attention to a scenario in which the input variable represents the time, so the functions domain is $\mathbb{R}^+$ instead of $\mathbb{R}$.

2.2 Abstract Domain Elements

Let us first formalize 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 two other sides are in the form $f^-(t) = m^-t + q^-$ and $f^+(t) = m^+t + q^+$ and approximate lower and upper values of f inside $[t_i, t_{i+1}]$. Figure 3 depicts a trapezoid defined on the step $[0, 3]$ and having $f^-(t) = 0.33t + 1$ and $f^+(t) = -0.17t + 3.5$ as, respectively, lower and upper sides.

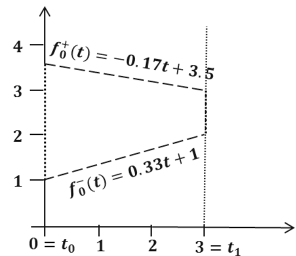


Fig. 3. Example of a trapezoid defined on $[0, 3]$

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 the intercept). Therefore, the pair of sides of each trapezoid is defined by a tuple $\mathbf{v} = (m^-, q^-, m^+, q^+)$, where $m^-, q^-, m^+, q^+ \in \mathbb{R}$ represent the two lines $f^-(t) = m^-t + q^-$ and $f^+(t) = m^+t + q^+$. We denote by f^- and f^+ the lower and the upper side, respectively. TSF can be seen as a generalization of IVSF, whose lower and upper sides are parallel and horizontal, i.e., with $m^- = m^+ = 0$.

Following the standard notation [5], given a set V of values, we represent a step function from time to V as a conjunction of constraints 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. The sequence of constraints can be finite or infinite but we only consider finite ones, 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 $N \in \mathbb{N} \wedge t_i \in \mathbb{R}^+ \wedge \mathbf{v_i} \in V\, \forall i$. V is the set of tuples $\{(m^-, q^-, m^+, q^+) : m^-, q^-, m^+, q^+ \in \mathbb{R}\}$. We will alternatively denote the value in a step as $\mathbf{v_i} = (m_i^-, q_i^-, m_i^+, q_i^+)$ or $\mathbf{v_i} = (f_i^-, f_i^+)$ where $f_i^-(t) = m_i^-t + q_i^-$ and $f_i^+(t) = m_i^+t + q_i^+$.

Normal Form and Equivalence Relation: This notation is not unique. 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 $[0, t + 1]$. For this reason, we use the same notion of normal form defined in [5]: the switching times t_i of a conjunction are sorted and different; moreover two consecutive constraints cannot have equal values (each $\mathbf{v_i}$ must be different from $\mathbf{v_{i+1}}$). With these conditions, the representation is unique. We will denote by *Norm* the normalization procedure. In our previous example, we would obtain a representation with a single constraint $0 : [0, 0, 1, 1]$. The normalization process induces an equivalence relation $(f \equiv g \Leftrightarrow Norm(f) = Norm(g))$.

Constraints: We impose two constraints on abstract elements:

1. inside each step $[t_i, t_{i+1}]$, the two lines f_i^- and f_i^+ do not intersect. This assumption is not restrictive: we can always split the invalid step with intersecting sides into two smaller and valid steps with non-intersecting sides through the *refine* operator that will be defined in Section 2.5;
2. two consecutive steps $[t_i, t_{i+1}]$ and $[t_{i+1}, t_{i+2}]$ must have at least one point in common at t_{i+1}. This constraint is needed because otherwise the concrete functions represented by the abstract element would not be continuous. Observe that an abstract state which violates this constraint is equivalent to bottom.

Formally, these constraints can be stated as follows:

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

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

Note that we do not require that the intervals of two consecutive steps are exactly the same at the border between them (i.e., neither the upper nor the lower sides have to link exactly the upper or the lower sides of the following step). We want our approach to be generic and for this reason we give as much freedom as we can to the abstract element definition.

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

2.3 Concretization Function

The abstract step function $f = \bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{v_i}\}$, 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 $t \in [t_i, t_{i+1}]$. The concretization function γ is thus defined by:

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

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

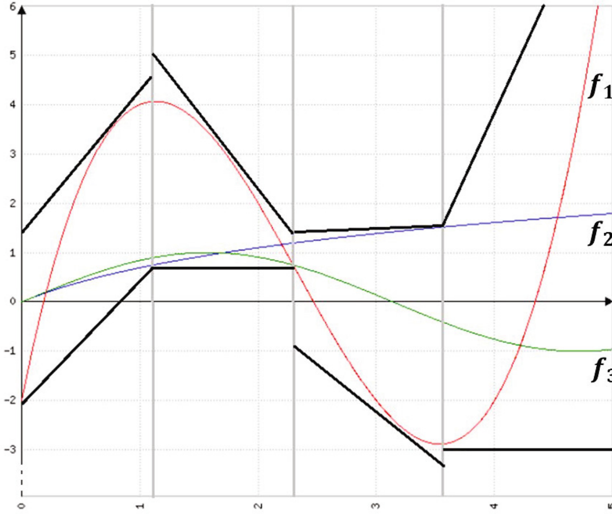


Fig. 4. Concretization function

Figure 4 depicts an example of an abstract state defined on the domain $[0, 5]$ with 4 steps (note that here $t_{N+1} = 5$). In the 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 Partial Order

The partial order $\subseteq^\sharp$ on two functions $f, g \in D^\sharp$ is defined point-wisely, 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 $f(t) = \{v : f_i^-(t) \leq v \leq f_i^+(t) \wedge t \in [t_i, t_{i+1}]\}$ 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:

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

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). We have to compare only the upper and lower sides of the trapezoids. To do this, we check that $f_i^- \geq g_j^- \wedge f_i^+ \leq g_j^+$ for all inputs $t \in [a, b]$.

Since the sides are defined by straight lines, it is sufficient to check only the values of such lines at the left and right extremes of the trapezoid.

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

$$\begin{array}{c} f \subseteq^\sharp g \\ \Updownarrow \\ \forall (i,j) \in [0,N] \times [0,M] : [a,b] = [t_i, t_{i+1}] \cap [u_j, u_{j+1}] \neq \emptyset \Rightarrow \mathbf{v_i} \sqsubseteq_{[a,b]} \mathbf{w_j} \end{array} \tag{3}$$

Observe that in (3) we compare the values of pairs of steps which have a part of their domain in common. 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 $f \subseteq^\sharp g$. 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 $[a,b] = [t_i, u_{j+1}]$ else $[a,b] = [u_j, t_{i+1}]$.

Lemma 1. *If $f, g \in D^\sharp$ are normalized, then $f \subseteq^\sharp g \Leftrightarrow \forall t \in \mathbb{R}_+, f(t) \subseteq g(t)$.*

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 $\gamma(\bot^\sharp) = \emptyset \wedge \forall f \in D^\sharp, \bot^\sharp \subseteq^\sharp f$. $D^\sharp \cup \{\bot^\sharp\}$ is a lattice.

2.5 *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, thus augmenting its number of steps. This operation has no impact on the concretization of the abstract state, 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 $S = \{s_k : s_k \in (T \cup U) \wedge s_k < s_{k+1}\, \forall k \in [0,P]\}$ 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 $\widehat{\mathbf{v_k}} = \mathbf{v_{max\{i : t_i \leq s_k\}}}$.

2.6 *Compact* Operator

The opposite operator with respect to *refine* is *compact*. This operator reduces the number of steps contained in an abstract state, and it will be useful in order to keep such number below a given threshold. The *compact* function works by merging a pair of steps, and repeating this procedure until a given threshold is reached. While *refine* leaves the precision of an abstract state unchanged, the *compact* operator induces some loss of precision.

Let $f = \bigwedge_{0 \leq i \leq N} \{t_i : \mathbf{v_i}\}$ be an abstract state, composed by $N + 1$ steps, and let M be a given threshold, 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 arbitrarily choose the step to be merged as the one with smallest width, but alternative solutions are possible and can be supported by our approach as well.

Let A_i and B_i be the two extremes (the left and right one, respectively) of f_i^+ in $[t_i, t_{i+1}]$, and let A_{i+1} and 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 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 we keep the original intercept of the side linking A_i and B_{i+1}. Figure 5 depicts this situation. The same applies symmetrically for the lower side. A slightly different process is required if the selected step is next to the last one (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 simply increase its intercept if one of the extremes of f_{N-1}^+ in $[t_{N-1}, t_N]$ is higher than such side. The same procedure applies for the lower side.

Note that this is not the only possible way to merge two steps. For example, in Figure 5, it could have been chosen the line passing through P and parallel to f_{i+1}^+ or others as well, but our method is the one which we found (empirically) to work best in most cases (i.e., it does not introduce too much approximation).

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. Then, $g = Compact_X(f, M)$ is an abstract state obtained by compacting f to M steps, while discarding only steps coming from $T \setminus X$.

Lemma 2. *Let $f \in D^\sharp$ and $M \in \mathbb{N}^+$. If $g = Compact(f, M)$, then $f \subseteq^\sharp g$.*

2.7 Least Upper Bound

Given two elements x and y of the abstract domain, the least upper bound operator $\sqcup^\sharp$ defines the least element z that overapproximates 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.

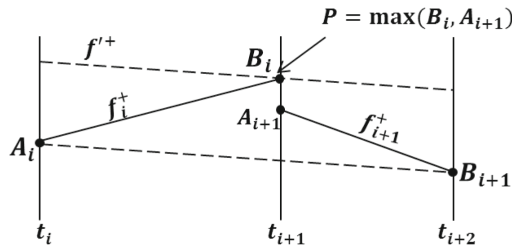
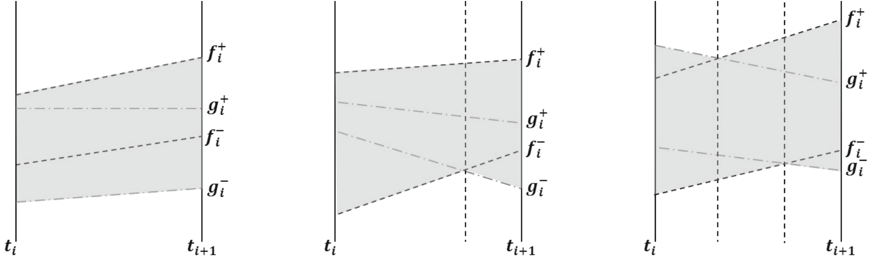


Fig. 5. Merging of two steps within the *compact* operation



(a) No intersections, the step remains unsplit. The grey area represents the resulting trapezoid

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

(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 grey)

Fig. 6. Examples of the LUB computation

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 the following algorithm. First, we refine f' and g' on the same set of steps, obtaining $f = \mathit{Refine}(f', U)$ and $g = \mathit{Refine}(g', X)$, where X and U are the steps sets of f' and g', respectively. Then, for each step t_i 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. In each of these new steps, we are sure that nor the upper sides nor the lower sides of f, g intersect each other. So, the resulting trapezoid for each new step is made by the greatest of the two upper sides and the lowest of the two lower sides. See Figure 6 for some representative examples.

Lemma 3. $\sqcup^\sharp$ *is the least upper bound operator.*

Lemma 4. *Let f and g be two* TSF *elements that satisfy the validity conditions stated in Section 2.2. Then, $f \sqcup^\sharp g$ satisfies the same conditions too.*

Similarly, we can also define the greatest lower bound operator.

2.8 Widening

The widening operator $\nabla_{D^\sharp}$ is parameterized on (i) k_S, the maximum number of steps allowed in an abstract state, (ii) k_M and k_Q, the maximum values allowed for the slope and intercept of trapezoid sides, respectively, (iii) k_I and k_L, the increment constants for the slope and intercept, respectively. All these parameters have to be ≥ 0. $\nabla_{D^\sharp}$ is then defined as follows.

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

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

We distinguish three cases: a) $|U| > k_S$; b) $g \subseteq^\sharp f$; c) otherwise. In case a), f exceeds the maximum number of steps allowed in an abstract state, k_S, and we return $\top^\sharp$. In case b), we do not have an ascending chain and we simply return f, which is already normalized, being an element of $D^\sharp$. In case c), we return the normalized and compacted version of h_{MQ}, keeping all the steps U of f. In this way, we are sure that U will be a subset of the steps set of $f\nabla_{D^\sharp}g$ and this is important for proving the convergence of $\nabla_{D^\sharp}$. The abstract state h_{MQ} is built as follows. Let g be defined on the indices set V. Let $f' = \mathit{Refine}(f, V)$ be the refined version of f with the addition of the indices of g and let $g' = \mathit{Refine}(g, U)$ be 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 $T = U \cup V$. Calling t_i the elements of T, we have: $f' = \bigwedge_{0\leq i\leq N}\{t_i : \mathbf{v_i} = (f_i^-, f_i^+)\}$ and $g' = \bigwedge_{0\leq i\leq N}\{t_i : \mathbf{w_i} = (g_i^-, g_i^+)\}$. We define $h_{MQ} = \bigwedge_{0\leq i\leq N}\{t_i : \mathbf{z_i} = (h_i^-, h_i^+)\}$ where (h_i^-, h_i^+) are as follows:

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

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

and

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

The computation is symmetric for the lower and the upper side, so let us focus on $h_i^+(t)$. For each step t_i of f' and g' we consider three distinct cases: 1) $f_i^+ = g_i^+$, 2) $(m_{g_i^+} \geq k_M) \vee (q_{g_i^+} \geq k_Q) \vee (m_{f_i^+} \geq k_M) \vee (q_{f_i^+} \geq k_Q)$, 3) otherwise. In case 1) the side is the same in f' and g', so we keep it unchanged. In case 2) the slope (or the intercept) of the side of one abstract state (g' or f') exceeds the threshold k_M (or k_Q), so we move the side to $+\infty$. Otherwise (in case 3), 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). This last case is needed to ensure the convergence of the operator. For the soundness of this operator we refer to the definition in [10].

3 Abstraction of a Continuous Function

In this Section we show how to compute the approximation of $\mathcal{C}^2_+$ functions, both in IVSF and in TSF. We consider both domains, as in [5] the abstraction function was not defined, since the authors relied on a particular type of ODE solver [4]. For TSF, we also consider two different approaches: when the step width is constant and fixed, and when we automatically determine the steps distribution. For IVSF, we consider only the case where the step list is fixed. Note that we abstract only one concrete function; this approach can be generalized to the abstraction of a discrete (or 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 (i) $f \in \mathcal{C}^2_+$ the continuous function we want to abstract, (ii) f' and f'' its first and second derivatives, respectively, (iii) ${F'}_0$ the set containing the points of the domain where $f'(t) = 0$ (stationary points), that is ${F'}_0 = \{t : f'(t) = 0\}$ (iv) ${F''}_0$ the same for f'' (inflection points), (v) ${F'}_0^{[a,b]} = \{f(t) : t \in ([a,b] \cap F'_0)\}$, that is, ${F'}_0^{[a,b]}$ is the set containing the stationary points of f restricted to the domain interval $[a,b]$, (vi) ${F''}_0^{[a,b]}$ the same as ${F'}_0^{[a,b]}$ but for the inflection points.

Note that the IVSF abstraction function needs to know only the first derivative of f (other than, obviously, f itself), while TSF requires also the second derivative.

3.1 IVSF Abstraction Function, Fixed Step Width

Given a step width w, suppose that $[a,b]$ is a generic interval ($b - a = w \wedge a = k \times w \wedge b = (k+1) \times w \wedge k, w \geq 0$). $M = \max(\{f(a), f(b)\} \cup {F'}_0^{[a,b]})$ is the maximum point of the function in the interval $[a,b]$, extremes included, and $m = \min(\{f(a), f(b)\} \cup {F'}_0^{[a,b]})$ 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.

3.2 TSF Basic Abstraction Function, Arbitrary Step Width

In TSF we can get a better representation by choosing a step distribution using (i) the inflection points ${F''}_0$, and (ii) the stationary points ${F'}_0$. Assume that $[a,b]$ is a generic interval obtained using this schema. Then, the two sides which compose the value of such step are as follows (see Figure 7):

1. the side l_1 linking the points $P = (a, f(a))$ and $Q = (b, f(b))$;
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 to do this is the following one:

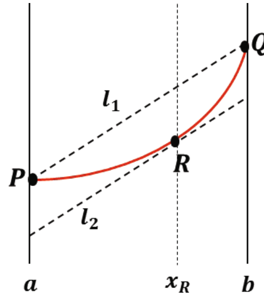


Fig. 7. The abstraction on the step $[a, b]$

(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 $R = (x_R, f(x_R))$. Then, l_2 is the side that goes through the point R and with slope equal to m_{l_1}, i.e., the slope of l_1.

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

3.3 TSF Basic Abstraction Function, Fixed Step Width

Of course, 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 the fixed width w. First of all, we split the interval in sub-intervals, following the schema introduced in Section 3.2. Then, for each sub-interval, we compute the upper and lower sides as specified in Section 3.2. Finally, we have to merge together these sub-intervals through the *compact* operator defined in Section 2.6. Notice that we lose some precision with this method, but we could achieve better results by adopting a more complex schema.

3.4 Dealing with Floating Point Precision Issues

Unfortunately, the abstraction technique presented in Section 3.2 is theoretically sound but it is not computable on a finite precision machine, due to the well-known problems concerning floating point representation. The abstraction function depends on various values: the points x at which $f'(x) = 0$, the points x at which $f''(x) = 0$, the point $x_r \in [a, b]$ such that $f'(x_R) = m_{l_1}$. 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. In this way, we enforce the soundness of the resulting abstraction function, not only theoretically but in a practical setting as well.

We assume that f respects the following property. If x_0 is a stationary or inflection point, 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, ϵ are parameters of the analysis. Intuitively, we ask that the function values change at most of τ around stationary and inflection points. Note that the value of ϵ can be set based on the standard in use on the machine (e.g., the IEEE 754 standard for floating points), while τ has to be set by the user: the smaller the value, the more precise the abstraction.

As in Section 3.2, we split the domain in steps with respect to the stationary and inflection points. 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. 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.

For the steps which do not contain stationary/inflection points, the computational schema of Section 3.2 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_{l_1} 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 sub-interval considered, we are sure that a tangent of it leaves the function always above (or below), resulting in a safe approximation.

4 Experimental Results

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 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

Table 1. Precision of TSF and IVSF varying the number of steps

#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%

$\ln(x+1)$[1]) 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 bigger the area, the rougher the abstraction. 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 execution times are very short, 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 memory. Table 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 the IVSF one (i.e., twice more precise). In all cases, TSF is more precise than IVSF. In the worst case, TSF is almost 3.5 times more precise (ratio $\approx$ 28.9%). In the best case, it is approximately 330 times more precise (ratio $\approx$ 0.3%). 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 slope of the curve 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 or 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 to a straight line in each single step (instead that a convex or concave curve) and the more the TSF precision increases.

[1] 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 $[0, 10]$.

Table 2. Values computed by TSF and IVSF on `intgrx`

Num. 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

4.2 An Integrator

Consider the motivating example presented in Section 1.1. Table 2 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

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 abstraction gets more and more precise (with respect to IVSF) every time we increase the number of steps in the representation. On the other hand, IVSF has the advantage to preserve the minimum and maximum values assumed by the function, while, unfortunately, TSF does not preserve such information, since the trapezoids vertices might exceed these values. Then, it could be useful in some applications, especially the ones where the stationary points of the function are relevant (e.g., $sin(x)$), to consider the product of these two domains, by using the Cartesian or the reduced product of the two [11]. For instance, in the analysis of the integrator code presented in Section 1.1 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

To the best of our knowledge, IVSF is the first formalism that allows the integration of the continuous environment in an abstract interpretation of embedded software. The static analyzer HybridFluctuat, based on IVSF, has been

implemented [3] in order to consider the interactions between the program and the physical environments on which it acts. In Section 4 we compared extensively the precision of our approach with respect to IVSF.

A useful domain theoretical characterization of continuous function can be found in [13], but this work only describes the continuous functions at the concrete level, and there is nothing involving the abstract interpretation theory.

Feret [14] introduced domain-specific abstract domains for digital filters, in the context of ASTREE [2], but did not provide a generic treatment of continuous functions and their abstraction.

As for hybrid systems, previous work in the context of abstract interpretation is mainly related to the analysis of hybrid automata [16,18].

Regarding continuity analysis of programs, Hamlet [17] was the first one to argue for a testing methodology for Lipschitz-continuity of software. Chaudhuri et al. recently proposed a qualitative program analysis to automatically determine if a program implements a continuous function [6]. Their practical motivation is the verification of robustness properties of programs whose inputs are uncertain. This work was further extended [7] to quantify the robustness of a program. 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 element 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 [9,20]. Another important issue is to find a PWL approximation of a certain function in order to minimize or bound the overall area (or the distance in each point) between the original function and the approximation [8,19,21]. Our approach, rather than bounding the error of the representation of a function, provides a *sound* approximation of it.

6 Conclusion and Future Work

Given the encouraging experimental results, 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 aim at exploring the use of 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 necessary for cost analysis [1]. Also, it could be interesting to do a formal complexity analysis on the domain operations.

Acknowledgments. Work partially supported by RAS project "TESLA - Tecniche di enforcement per la sicurezza dei linguaggi e delle applicazioni".

References

1. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: Cost Analysis of Java Bytecode. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 157–172. Springer, Heidelberg (2007)
2. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Proceedings of PLDI 2003. ACM (2003)
3. Bouissou, O., Goubault, E., Putot, S., Tekkal, K., Vedrine, F.: HybridFluctuat: A Static Analyzer of Numerical Programs within a Continuous Environment. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 620–626. Springer, Heidelberg (2009)
4. Bouissou, O., Martel, M.: GRKLib: a guaranteed runge-kutta library. In: Proceedings of SCAN 2007. IEEE Press (2007)
5. Bouissou, O., Martel, M.: Abstract Interpretation of the Physical Inputs of Embedded Programs. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 37–51. Springer, Heidelberg (2008)
6. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity analysis of programs. In: Proceedings of POPL 2010. ACM (2010)
7. Chaudhuri, S., Gulwani, S., Lublinerman, R., NavidPour, S.: Proving programs robust. In: Proceedings of FSE 2011. ACM (2011)
8. Chou, F., Wang, C.M., Cheng, G.D.: Optimal bounding of curves by continuous piecewise linear functions. Engineering Optimization 21(4), 307–317 (1993)
9. Chua, L., Kang, S.M.: Section-wise piecewise-linear functions: Canonical representation, properties, and applications. Proceedings of the IEEE 65(6), 915–929 (1977)
10. Cortesi, A.: Widening operators for abstract interpretation. In: Proceedings of SEFM 2008. IEEE Press (2008)
11. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of POPL 1977. ACM (1977)
12. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proceedings of POPL 1979. ACM (1979)
13. Edalat, A., Lieutier, A.: Domain theory and differential calculus (functions of one variable). Mathematical Structures in Comp. Sci. 14(6) (2004)
14. Feret, J.: Static Analysis of Digital Filters. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 33–48. Springer, Heidelberg (2004)
15. Goubault, E., Martel, M., Putot, S.: Some future challenges in the validation of control systems. In: Proceedings of ERTS 2006 (2006)
16. Halbwachs, N., Proy, Y.-E., Raymond, P.: Verification of Linear Hybrid Systems by Means of Convex Approximations. In: LeCharlier, B. (ed.) SAS 1994. LNCS, vol. 864, pp. 223–237. Springer, Heidelberg (1994)
17. Hamlet, D.: Continuity in software systems. In: Proceedings of ISSTA 2002. ACM (2002)
18. Henzinger, T.A., Ho, P.-H.: A Note on Abstract Interpretation Strategies for Hybrid Automata. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994. LNCS, vol. 999, pp. 252–264. Springer, Heidelberg (1995)
19. Imai, H., Iri, M.: An optimal algorithm for approximating a piecewise linear function. Journal of Information Processing 9(3), 159–162 (1987)
20. Kahlert, C., Chua, L.: A generalized canonical piecewise-linear representation. IEEE Transactions on Circuits and Systems 37(3), 373–383 (1990)
21. Tomek, I.: Two algorithms for piecewise-linear continuous approximation of functions of one variable. IEEE Trans. Comput. 23(4), 445–448 (1974)