This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Formal Feature Interpretation of Hybrid Systems

Antonio Anastasio Bruto da Costa, *Student Member, IEEE,* Goran Frehse, *Member, IEEE,*
Pallab Dasgupta, *Senior Member, IEEE.*

***Abstract*—In current practice a *formal* analysis of hybrid system models is assertion-based. The work presented here is based on *features* that look beyond functional correctness towards a quantitative evaluation of behavioural attributes. A feature defines a real-valued evaluation function over a specific set of traces. This article describes an improved method for the interpretation of features over hybrid automata models. It further demonstrates how Satisfiability Modulo Theory (SMT) solvers can be used for extracting behavioural traces corresponding to corner cases of a feature. Results are demonstrated on examples from the control and circuit domains.**

***Index Terms*—Hybrid Automata, Sequence Expressions, Features, Model Checking**

I. Introduction

The theory of Hybrid Automata (HA) has been extensively studied in the context of designing provably safe designs of embedded hybrid systems [1], [2], [3]. The formal safety analysis of hybrid systems is becoming increasingly significant with wider proliferation of automated control in circuits and systems.

An important component of any formal verification framework is the mechanism for formally specifying the design intent. In the discrete domain, formalisms based on temporal logic have been widely adopted with the use of standard assertion languages, such as SystemVerilog Assertions (SVA) [4] and Property Specification Language (PSL) [5]. Analog Mixed-Signal (AMS) extensions of assertions have been explored as well [6], [7], [8] and provide constructs for assertions over real-valued attributes. Assertion based verification of HA has also been studied [9], while tools such as SpaceEx [10] have been used to analyze timed and hybrid models of embedded control systems using reachability analysis and model checking. However, assertions in these languages, in and of themselves limit the information carried by their Boolean outcome. Our experience is that designers want to understand what the design is doing and how it behaves, not just the success/fail scenarios. This is naturally expressed as a quantitative real-valued measure.

Existing literature on quantitative specifications [7], [11], [12] is assertion based, and uses metrics, with positive values indicating truth, negative values indicating violations, and robustness being described as the distance of the quantity from zero. Some metrics (such as in [13]) are associated with uncertainty. Assertion-based languages are designed to be flexible with respect to the assertions written but their quantitative interpretations are restricted. On the other hand, the language of *features*, Feature Indented Assertions (FIA) [14], is designed to be flexible on the definition of the quantity, with the set of assertions that can be expressed limited to those that are sequences of predicates and events.

Many properties used in practice concerning system attributes can be intuitively expressed as features. Related work exists in learning parameters of Signal Temporal Logic (STL) properties, such as [15], wherein the authors propose learning tight bounds on parameters of STL properties from system traces. The approach can indeed be used to learn those features that can be expressed as parameters in STL properties. However, note that expressing a feature to be learned as an STL property can require the use of additional parameters, thus making the analysis more expensive.

STL property based analysis is implemented in MATLAB toolboxes such as Breach [16] and S-Taliro [17]. These can be used for parameter synthesis, robustness monitoring and parameter sensitivity analysis. The work presented in this manuscript is largely influenced by the semiconductor industry which finds expressing properties tedious in temporal logic based languages. As a result, for digital designs, the IEEE 1800-2012 Standard SVA language is widely used across the industry for expressing assertions for the validation and verification of digital circuits. FIA is developed over the fabric of SVA, which in practice enables the more intuitive expression of real-valued quantities over system traces.

To understand features, consider the *settling time* of a DC-DC Buck Regulator, defined as the time taken for the output voltage `x1` to settle to below $Vr + \epsilon$ for two successive openings of the capacitor switch; Vr is the rated voltage for the regulator. Booleanizing the notion of settling of `x1` within 100 clock cycles in SVA, using propositional variables `x1_GE_Vr` $\equiv$ `(x1>=Vr+E)`, and `swOpen` to mean the capacitor switch is open, yields the following sequence:

```
x1_GT_Vr ##[0:100] first_match(@(posedge swOpen)&&
!x1_GE_Vr ##[0:$] @(posedge swOpen) && !x1_GE_Vr)
```

The expression represents the regulator's behaviour of settling, that is the first time when the regulator's voltage output `x1` is found to be less than `Vr+E` for two consecutive openings of the buck regulator capacitor switch, specified as two successive capacitor switch open events, after having risen above `Vr+E`. The semantics of the assertion depends on a clock and all sequence delays are in terms of this clock. A change of clock requires re-writing the assertion with delays consistent with the revised clock, thereby inviting human error. Additionally, this form of expression requires Booleanization and is non-intuitive.

Bruto da Costa, Antonio A., and Dasgupta, Pallab, are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India. Frehse, Goran is with Verimag, Université Grenoble Alpes, Grenoble, France.

The authors acknowledge the support of Semiconductor Research Corporation (SRC) through the research grants, 2012-TJ-2267 and 2017-CT-2740.

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

The verification of a buck converter model against an expression like the one above would yield a Boolean outcome. However, the feature *settling time* is a real valued artifact. In FIA this is expressed by overlaying the computation of settling time over a sequence expressing the behaviour of settling. This is done using the power of *local variables* to store state variable values as the sequence matches, shown below in Example 1. FIA was introduced in [14], wherein features were used to analyze systems in a simulation environment. The formal expression of features is based on the syntactic fabric of assertions, but the definition of assertions is overlayed with real valued functions that are computed over matches of underlying logical expressions. This enables the formal expression of definitions of standard features like *rise time*, *peak overshoot* and *settling time*, and other design specific features.

Example 1. *Settling Time:* The local variable `st` is assigned in the antecedent and is used to define the feature value `settlingTime` in the consequent.

```
feature settlingTime(Vr,E);
begin
    var st;
    (x1>=Vr+E) ##[0:$]
        @+(state==Open) && (x1<=Vr+E), st=$time
            ##[0:$] @+(state==Open) && (x1<=Vr+E)
    |-> settlingTime = st;
end
```

The feature in Example 1 has two parameters `Vr` and `E` that are used later in the contained sequence expression. `st` is an uninterpreted local variable of the feature that is assigned the real time at the first opening of the switch after `x1>=Vr+E`, but when `x1<=Vr+E`. The variable `settlingTime` has the same name as the feature, and is assigned the value of the local variable when the entire sequence matches. In the sequence expression of the feature, the notable differences with SVA are the following:

1) Predicates over real valued signals (PORVs) [7], such as `(x1>=Vr+E)`, are allowed. PORVs can be over real variables or over the special variable `state` which refers to the name of the mode of operation of the buck regulator automaton.
2) `@+` is used to denote the positive crossing of PORVs. For a predicate involving variable `state`, this indicates that the state is entered. Similarly `@-` may be used for negative crossings of PORVs.
3) All intervals of the form `##[a:b]` are treated as *real time intervals*, as opposed to intervals countable in terms of the number of clock cycles in SVA semantics. This avoids rewriting the property if the clock cycle changes.

The repertoire of work presented in this article is rooted in the use of features for the verification and analysis of hybrid systems and consists of the following:

1) A methodology to compute an over-approximation of the range of feature values for all possible runs of the system.
2) A methodology for finding the extremal values of the feature range through successive refinement using SMT.

Methodology 2 makes its first appearance in this article. Methodology 1 was first reported in [18], where a technique for manually transforming models was outlined, but only for very specific types of features. A more general technique was later reported in [19], [20] with its integration into simulation flows in [21]. Here, at the heart of Methodology 1, we present an improved computation of the *product* automaton of [19]. The product in [19] implements conservative semantics for the feature's sequence expression, yielding a feature range that ignores some matches of the sequence-expression, and also includes matches with broader semantics than intended. The semantics of the keyword `first_match` used in the SVA sequence-expression enforces predictability in the match, by ensuring that only the earliest observation of the contained sub-sequence is matched. In this article, we extend the more general product described in [19] with *first match region semantics* described in Section III-B. In a feature, a combination of first-match and non-first-match semantics may be used (as in SVA). However, in this article, for simplicity we assume that all features are evaluated with first-match region semantics. In the past, Methodology 1 was primarily used with reachset computation tools like SpaceEx [10] to compute feature ranges. However, this is not always the best solution because in our experience the results tend to be conservative. In practice, we find that tighter feature ranges can be computed using SMT tools, at the price of longer runtimes, and possibly choking if the unfolding is too large. In summary, we propose two technologies, one which is faster but coarser, and another which is slower but more precise. We present various case studies on hybrid systems from the circuit and control domains.

II. Preliminaries

Given a system defined as a HA $\mathcal{H}$, and a feature F, the objective is to find the range of valuations of F over all possible runs of $\mathcal{H}$. This section presents the requisite definitions of HA, and feature semantics over runs of a HA.

A. Hybrid Automata

A hybrid automaton is defined as follows:

Definition 1. ***Hybrid Automaton***
A hybrid automaton [1] is a collection $\mathcal{H} = (Q, X, Lab, Init, Dom, Edg, Act)$, where:

- Q is the set of ***discrete states*** *also known as* ***locations****;* X is a finite set of real-valued variables. A *valuation* is a function $\nu : X \to \mathbb{R}$. Let $\mathcal{V}(X)$ denote the set of valuations over X*;* Lab is a *finite set of* ***synchronization labels****;* $Init \subseteq Q \times \mathcal{V}(X)$ is a set of ***initial states****;* $Dom(l) : Q \to 2^{\mathcal{V}(X)}$ is a **domain**. *Dom(l)* $\subseteq \mathbb{R}^n$ is function that assigns a set of continuous states to each discrete state $l \in Q$.
- Edg is a set of **edges**, also called transitions. Each edge $e = (p, a, \mu, r)$ consists of a source location $p \in Q$, a target location $r \in Q$, a synchronization label $a \in Lab$, and a transition relation $\mu \subseteq \mathcal{V}(X) \times \mathcal{V}(X)$. A transition e is enabled in state (p, ν) if for some valuation $\nu' \in \mathcal{V}(X)$, $(\nu, \nu') \in \mu$. We require that for each location $p \in Q$, there be a stutter transition of the form $(p, \kappa, \mu_{ID_X}, p)$, $\mu_{ID_X} = \{(\nu, \nu) | \nu \in \mathcal{V}(X)\}$.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

- Act is a function that assigns to each location a (possibly infinite) set of activities, where each activity $f : \mathbb{R}^{\geq 0} \to \mathcal{V}(X)$ represents an evolution of the variables over time. The set of activities is usually defined implicitly as the set of solutions to a system of differential equations or inclusions. We denote the expression associated with the time derivative of a variable $x \in X$ in location $p \in Q$ as $flow_p^x$.

A state of $\mathcal{H}$ *is given as* $(p, \nu) \in Q \times \mathcal{V}(X)$. □

For valuation ν, we use $\nu_{\downarrow U}$ as the projection of ν on the set of variables $U \subseteq X$. For variable u, $\nu_{\downarrow u}$ is the value of variable $u \in X$ in state ν. Similarly for a set of valuations $\mathcal{R}$, $\mathcal{R}_{\downarrow U}$ and $\mathcal{R}_{\downarrow u}$ are respectively the projection of $\mathcal{R}$ on the variable set U, and variable u respectively. For an edge $e = (p, a, \mu, r)$, $e \in Edg$, $G(\mu) = \{\nu | (\nu, \nu') \in \mu\}$. $G(\mu)$ is commonly known as the *transition guard* and is often represented as a set of predicates over variables in X. Similarly $R(\mu, \nu) = \{\nu' | (\nu, \nu') \in \mu\}$, is known as the *reset relation*, and most often appears as a function, i.e. $R(\mu, \nu) = \nu'$, $(\nu, \nu') \in \mu$. When a system consists of multiple interacting components, we assume that a parallel composition of automata is available prior to applying the algorithms presented in this article. The methods discussed in this article are applied on linear hybrid systems that have monotonically increasing or decreasing variable dynamics in each location. Non-linear systems can be approximated as piece-wise affine models using techniques such as *hybridization* [22]. Furthermore, a location with non-monotonic variable dynamics can be transformed into an equivalent model with location-wise monotonic variable dynamics. The constraint of monotonicity is used in the article to accommodate existing tools (which use *may*, non-urgent, semantics on transitions). With tools that use urgent semantics, this restriction can be lifted.

Definition 2. ***Run of the hybrid system*** $\mathcal{H}$

A run of the hybrid system $\mathcal{H}$*, is a finite or infinite sequence*

$$\rho : \quad \sigma_0 \mapsto_{f_0}^{t_0} \sigma_1 \mapsto_{f_1}^{t_1} \sigma_2 \mapsto_{f_2}^{t_2} \cdots$$

of states $\sigma_i = (l_i, \nu_i)$*, non-negative reals* $t_i \in \mathbb{R}^+$*, and activities* f_i *of location* l_i*, such that for all* $i \geq 0$*,*

1) $f_i(0) = \nu_i$*,*
2) *For all* $0 \leq \theta \leq t_i$*,* $f_i(\theta) \in Dom(l_i)$*,*
3) *There exists an edge* (l_i, a, μ, l_{i+1}) *such that* $(f_i(t_i), \nu_{i+1}) \in \mu$ *and* $\nu_{i+1} \in Dom(l_{i+1})$. □

The HA model for a two location DC-DC Buck Regulator [23], in Figure 1, presents analysis complexities due to its high location switching frequency, yet is simple enough to help explain the notion of feature analysis. It has two variables, the voltage across the load (x_1) and the load current (x_2). The dynamics in each mode of operation may be found in [23]. The notation u' in a reset relation on an edge is the value of u after the transition is taken. This model is used as a running example in this article along with the feature `Settling Time` described in Example 1.

B. Feature Semantics for Hybrid Automata

This section presents the semantics of features which are an improved version of [19]. We introduce stricter first-match semantics for feature matches in Section III-B, while a more general semantic for feature-matches is presented here. The language for expressing features, FIA, uses *Predicates Over Real Variables* (PORVs)[7]. A feature is formally defined using the following syntax,

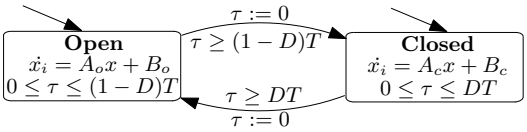

Fig. 1: HA model of a DC-DC Buck Regulator [23].

```
feature F_name (L_p);
begin
      var L;
      S |-> F_name = F ;
end
```

where $\mathcal{F}_{name}$ is the feature name, L_p and $\mathcal{L}$ are respectively the list of parameters and the list of local variables used in the body of the feature. $\mathcal{F}_{name}$ is a special variable representing the value of the feature assigned to it in the expression $\mathcal{F}$. S is a sequence expression of the form,

$$s_1 \texttt{ ## } \tau_1 \quad s_2 \texttt{ ## } \tau_2 \quad \ldots \texttt{ ## } \tau_{n-1} \quad s_n$$

and $\mathcal{F}$ is a linear function over $\mathcal{L}$ which assigns the feature value. τ_i represents a time interval, also referred to as a *delay operator*, and is of the form $[a : b]$, where $a, b \in \mathbb{R}^+, a \leq b$, and additionally b can be the symbol \$, representing infinity.

Sub-expressions s_1, s_2, ..., s_n are each of the form "$D \wedge E$, $\mathcal{A}$", where D is a Boolean expression of PORVs in disjunctive normal form, E is an optional event and $\mathcal{A}$ is an optional list of comma-separated local variable assignments. For sub-expression s_i, we use $Cond(s_i) \equiv D_i \wedge E_i$ to represent a Boolean expression of PORVs and an event, and $\mathcal{A}_i = [A_i^1, A_i^2, ..., A_i^k]$ is a list of k local variable assignments in s_i. The feature expression S `|->` $\mathcal{F}$ is interpreted as the computation of $\mathcal{F}$ whenever there is a match of sequence expression S. We use the notation S_i^j, $1 \leq i \leq j \leq n$ to denote the sub-sequence expression s_i `##` τ_i ... `##` τ_{j-1} s_j.

Given run ρ of $\mathcal{H}$, the following definitions indicate what it means for a feature to match ρ.

Definition 3. ***Event Match:*** *An event* $E \equiv @^+(P)$ *matches in a run* $\rho : \quad \sigma_0 \mapsto_{f_0}^{t_0} \cdots \mapsto_{f_{i-1}}^{t_{i-1}} \sigma_i \mapsto_{f_i}^{t_i} \cdots$ *at index* i *iff* $i > 0, t_{i-1} > 0, \forall_{\theta \in [0:t_{i-1})} (l_{i-1}, f_{i-1}(\theta)) \not\models P \bigwedge \sigma_i \models P$.

We define $@^-(P) \equiv @^+(\neg P)$ *and* $@(P) \equiv @^+(P) \vee @^-(P)$.

We use the notation $\sigma_i \models_\rho E$ *to denote the fact that the event E matches in the run* ρ *at index* i. □

To extend predicates to be evaluated over locations of the HA, for state $\sigma = (l, \nu)$, a predicate P can also take the form, `state == l`, where `state` is a special variable denoting the location label.

Definition 4. *The notation* $\sigma \models_\rho s$*, where* s *is treated as* $Cond(s)$ *and* $\sigma = (l, f(t))$*, is extended to conjunctions and disjunctions of PORVs and events recursively, as defined below. Note that* s *does not have any delay operators.*

- $\sigma \models_\rho P$ *iff* P *is a PORV and* P *is true for signal valuation* $f(t)$*, or* $P \equiv$ `(state == l)`.
- $\sigma \models_\rho C$*, where* $C = P_1 \wedge P_2 \wedge ... \wedge P_n$*,* P_i *is a PORV iff* $\forall_{i=1}^n \sigma \models P_i$.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

- $\sigma \vDash_\rho D$, *where* $D = C_1 \vee C_2 \vee ... \vee C_n$, C_i *is a conjunction of PORVs iff* $\exists_{i=1}^{n}\ \sigma \vDash C_i$.
- $\sigma \vDash_\rho D \wedge E$, *where* $D = C_1 \vee C_2 \vee ... \vee C_n$, C_i *is a conjunction of PORVs and* E *is an optional event, iff* $\sigma \vDash_\rho D \ \wedge \ \sigma \vDash_\rho E$.

For hybrid system $\mathcal{H}$ *and sub-expression* s, *we say that* $l \vDash s$ *if for some* $\sigma = (l, \nu)$, $\sigma \vDash_\rho s$. *For sub-expression* $s_j = D_j \wedge E_j$, C_i^j *is the* i^{th} *conjunct term in* D_j. Υ_i^j *is the* `state` *context for* C_i^j, *i.e.* $\Upsilon_i^j = l$ *iff* (`state` $== l$) *is a PORV in* C_i^j. *Similarly* Υ^j *is the state context of event* E_j *in* s_j.[1] □

Definition 5. ***Match*** $\mathcal{M}$ ***of Sequence Expression*** S: *A run* $\rho : \ \sigma_0 \mapsto_{f_0}^{t_0} \sigma_1 \mapsto_{f_1}^{t_1} \sigma_2 \mapsto_{f_2}^{t_2} \cdots$ *has a match* $\mathcal{M} = \langle i_1, \ldots, i_n \rangle$ *of the sequence expression* $S = s_1$ `##` $\tau_1 \ s_2$ `##` τ_2 ... `##` $\tau_{n-1} \ s_n$, $n \geq 1$ *if* $\forall_{j=1}^{n-1} i_j \leq i_{j+1}$ *and the following conditions hold:*

- $\sigma_{i_1} \vDash_\rho s_1$,
- $\sigma_{i_2} \vDash_\rho s_2$ *and* $t_{i_1} + ... + t_{i_2 - 1} \in \tau_1$,
- *and so on ... until,*
- $\sigma_{i_n} \vDash_\rho s_n$ *and* $t_{i_{n-1}} + ... + t_{i_n - 1} \in \tau_n$,

Local variables associated with s_j *are assigned values from variable valuations in state* σ_{i_j}. *Note that there can be multiple matches of* S *in* ρ, *and multiple runs of* H *that match* S. *Each match* $\mathcal{M}$ *defines a feature value, denoted* $Eval(\mathcal{M}, \rho, \mathcal{F})$, *computed as the value of feature expression* $\mathcal{F}$ *over values of the local variables assigned during match* $\mathcal{M}$ *in run* ρ. □

Definition 6. ***Feature Range of a Hybrid Automaton:*** *Given a feature sequence expression* S *and a feature computation function* $\mathcal{F}$, *that computes the value of* F_{name}, *the feature range* $[\mathcal{F}_{min}, \mathcal{F}_{max}]$ *of a hybrid automaton* H *is computed as follows:*

- $\mathcal{F}_{min}$ = $\min\{\ Eval(\mathcal{M}, \rho, \mathcal{F}) \mid \rho$ *is a run in* H *and* $\mathcal{M}$ *is a match of* S *in* $\rho\ \}$
- $\mathcal{F}_{max}$ = $\max\{\ Eval(\mathcal{M}, \rho, \mathcal{F}) \mid \rho$ *is a run in* H *and* $\mathcal{M}$ *is a match of* S *in* $\rho\ \}$

Note that max and min are computed over all matches in all runs ρ *of* H. □

III. Method-1: Feature Transformations

The problem of feature analysis can be formally defined as follows: Given a HA $\mathcal{H} = (Q, X, Lab, Init, Dom, Edg, Act)$ and a feature F, we wish to compute $\mathcal{F}_{min}$ and $\mathcal{F}_{max}$ over all runs ρ of $\mathcal{H}$.

The methodology consists of the three steps shown in Figure 2. Compared to [19], the Feature Automaton definition is re-written to better capture the feature intent of Section II-B, and in Section III-B we present an improved construction of the product automaton.

A. Feature Automaton Construction

The feature automaton is a monitor automaton which is similar to a HA, but allows guards to be written with predicates over location labels and events. Additionally, unlike the HA in Definition 1, a feature automaton also has an accept location.

[1]We assume for simplicity that a conjunct in a sub-expression of the sequence does not have any contradictory 'state' constraints.

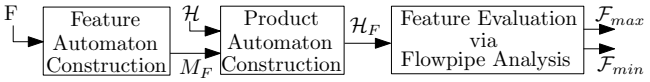

Fig. 2: Methodology-1: Feature Transformations.

Given a HA $\mathcal{H} = (Q_H, X_H, Lab_H, Init_H, Dom_H, Edg_H, Act_H)$, a feature F with local variable set $\mathcal{L} = \{l_0, l_1, ..., l_m\}$, sequence expression $S = s_1$ `##` $\tau_1 \ \ s_2$ `##` $\tau_2 \ \ ...$ `##` $\tau_{n-1} \ \ s_n$ and feature computation expression $\mathcal{F}$ that assigns the feature value to $\mathcal{F}_{name}$, we construct the feature automaton as follows:

Definition 7. ***Feature Automaton****:* A Feature Automaton for HA $\mathcal{H}$ is a collection $M_F = (Q, Z, X, V, C, E, Init, q_\mathcal{F})$, where:

- $Q = \{q_1, q_2, ..., q_{n+1}, q_\mathcal{F}\} \cup Z$ is the set of feature locations. Intuitively, location q_i is reached when the sequence expression has matched upto s_{i-1} and it awaits the match of s_i within the time interval τ_{i-1}. $q_\mathcal{F}$ is the *accept* location;
- Z = $\{q_{i,j} \mid |\mathcal{A}_i| > 1, 1 \leq i \leq n, 1 \leq j \leq |\mathcal{A}_i| - 1\} \cup \{q_{n+1}, q_\mathcal{F}\}$ is a set of **pause locations** where time does not progress. If a subexpression has multiple assignment statements, then a sequence of pause locations are added corresponding to all but the first assignment, to capture the order in which these assignments are made. Pause locations $q_{i,j}$ are added if $|\mathcal{A}_i| > 1$. State $q_{i,j}$ is reached when the j^{th} assignment of s_i has been executed;
- $X = X_H$; V *is the set of feature variables,* $V = \mathcal{L} \cup \{\mathcal{F}_{name}\}$; C = $\{t, lt\}$ is a set of **timers** where t measures cumulative time along a match, and lt is a location timer, measuring time spent in every location. All timers are initially 0;
- We augment the assignment list $\mathcal{A}_i$ for sub-expression s_i with the assignment `lt:=0` for the *location timer* lt.
- An event of the form $@^*(x \sim a)$ is associated with PORVs as follows:
  - $@^+(x \geq a)$ is associated with $(x == a) \bigwedge flow_q^x > 0$
  - $@^-(x \leq a)$ is associated with $(x == a) \bigwedge flow_q^x < 0$
- $E \subseteq Q \times \mathcal{V}(Q_H \cup X \cup V \cup C) \times Q$ is the set of edges defined by the following rules:
  - $\forall_{1 \leq i \leq n}(|\mathcal{A}_i| = 1) \rightarrow (q_i, \mu_i, q_{i+1}) \in E$
  - $\forall_{1 \leq i \leq n}(|\mathcal{A}_i| > 1) \rightarrow (q_i, \mu_i, q_{i,1}) \in E \wedge \forall_{1 \leq j < |\mathcal{A}_i| - 1} (q_{i,j}, \mu_{i,j}, q_{i,j+1}) \in E \wedge (q_{i,|A_{i-1}|}, \mu_{i,|\mathcal{A}_{i-1}|}, q_{i+1}) \in E$.
  - $\forall_{1 \leq i < n}\ \mu_i = (Cond(s_i) \wedge (lt \in \tau_i) \wedge A_i^1 \wedge \{\texttt{lt}' == 0\})$; $\mu_n = (Cond(s_n) \wedge A_n^1)$
  - $\forall_{1 \leq i < n}\ \forall_{1 \leq j \leq |\mathcal{A}_i| - 1}\ \mu_{i,j} = true \wedge A_i^{j+1}$
  - $(q_{n+1}, \mu_F, q_\mathcal{F}) \in E$, where $\mu_F = \{\mathcal{F}'_{name} == \mathcal{F}\}$
  - For relation μ, ω denotes the projection of the relation μ onto $\mathcal{V}(X \cup V \cup C) \times \mathcal{V}(X \cup V \cup C)$.
- $Init = \{q_1\} \times [0]^{|V \cup C}$ is the set of initial states.

A state of M_F *is given as* $(q, \nu) \in Q \times \mathcal{V}(X \cup V \cup C)$. □

For feature `Settling Time` of Example 1, the FA is shown in Figure 3. The FA has two timer variables, t for measuring time along the entire run, and lt for measuring the time spent in each location of the FA, indicative of delays separating subexpressions in the feature sequence expression. Each location of the FA represents the match of some feature subexpression. q_i represents that the temporal sequence of events

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

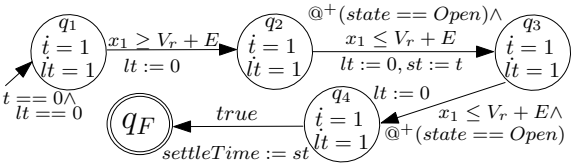

Fig. 3: Feature Automaton for feature `Settling Time`.

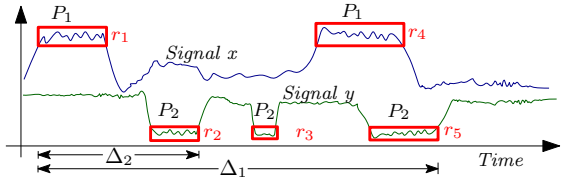

Fig. 4: An illustration of first-match region semantics.

and PORVs leading up to (but not including) the i^{th} sub-expression has been observed. The transition between q_i and q_{i+1} is guarded by the Boolean expression of PORVs and events corresponding to the i^{th} sub-expression; the associated set of assignments to local variables are computed along this transition. Progressing along transitions between locations of the automaton corresponds to matching each sub-expression. When the n^{th} sub-expression matches, the entire sequence expression has matched and the automaton transitions to state q_{n+1}. At q_{n+1}, all local variables hold values assigned to them along the match, and the feature is computed along the unguarded transition from q_{n+1} to $q_{\mathcal{F}}$, the accept location of the automaton.

Definition 8. ***Acceptance of run*** ρ ***of*** $\mathcal{H}$ ***by*** M_F***:*** *Run* $\rho: \sigma_0 \mapsto^{t_0}_{f_0} \sigma_1 \mapsto^{t_1}_{f_1} \sigma_2 \mapsto^{t_2}_{f_2} \cdots$ *of* $\mathcal{H}$ *is accepted by feature automaton* M_F *for feature* F *with sequence expression* s_1 ## τ_1 s_2 ## τ_2 ... ## τ_{n-1} s_n *iff* $\exists\ \sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_n}$*, such that* $i_j \leq i_{j+1}$*, where,* $\sigma_{i_1} \vDash s_1$*;*

- $\sigma_{i_2} \vDash s_2 \bigwedge (lt = \sum_{k=i_1}^{i_2-1} t_k) \in \tau_1$*; ...*
- $\sigma_{i_n} \vDash s_n \bigwedge (lt = \sum_{k=i_{n-1}}^{i_n-1} t_k) \in \tau_{n-1}$ □

Theorem 1. *Given a feature,* F *in FIA, for HA* $\mathcal{H}$*, the feature automaton* $M_F = (Q, Z, X, V, C, E, q_{\mathcal{F}})$ *for* F *correctly captures the following feature semantics:*

A If a run ρ *of* $\mathcal{H}$ *yields a match* $\mathcal{M}$*, then the run* ρ *is accepted by feature automaton* M_F *with the same valuation as* $Eval(\mathcal{M}, \rho, \mathcal{F})$*.*

B If a run ρ *of* $\mathcal{H}$ *is accepted by* M_F *with valuation* γ*, then* ρ *has a match* $\mathcal{M}$*, such that* $Eval(\mathcal{M}, \rho, \mathcal{F}) = \gamma$*.*

Proof. We prove the theorem in two parts as follows:

Part A: Let $\rho: \sigma_0 \mapsto^{t_0}_{f_0} \cdots \mapsto^{t_{i_1-1}}_{f_{i_1-1}} \sigma_{i_1} \mapsto^{t_{i_1}}_{f_{i_1}} \cdots \mapsto^{t_{i_2-1}}_{f_{i_2-1}} \sigma_{i_2} \mapsto^{t_{i_2}}_{f_{i_2}} \cdots$ be a run of $\mathcal{H}$ that matches the sequence expression $S = s_1$ ##τ_1 s_2 ## τ_2 ... ##τ_{n-1} s_n, with match $\mathcal{M} = \langle i_1, \ldots, i_n \rangle$. Let M_F be the feature automaton constructed for feature F.

The initial location of M_F is q_1. In the prefix $\sigma_0 \mapsto^{t_0}_{f_0} \cdots \mapsto^{t_{i_1-1}}_{f_{i_1-1}} \sigma_{i_1}$ of ρ, $\sigma_{i_1} \vDash_\rho s_1$ and G(μ_1) = s_1, hence the state q_2 is reachable in the feature automaton with state σ_{i_1} of ρ, with associated assignments to local variable made along the transition. For configuration $\langle q_j, \sigma_{i_j} \rangle$ reachable in M_F, $\langle q_{j+1}, \sigma_{i_{j+1}} \rangle$ is reachable for all $1 \leq j \leq n$. Given that $\langle q_j, \sigma_{i_j} \rangle$ is reachable, (q_j, μ_j, q_{j+1}) is an edge in M_F, and $\mathcal{M}$ is a match, we have $\sigma_{i_j} \vDash_\rho s_j$ and $G(\mu_j) \equiv s_j \wedge (t_{i_{j-1}} + ... + t_{i_j-1}) \in \tau_j$, $\langle q_{j+1}, \sigma_{i_j} \rangle$ is reachable, via one or more transitions through pause states. Now, since $\sigma_{i_j} \mapsto^{t_{i_j}}_{f_{i_j}} \cdots \mapsto^{t_{i_{j+1}-1}}_{f_{i_{j+1}-1}} \sigma_{i_{j+1}}$, configuration $\langle q_{j+1}, \sigma_{i_{j+1}} \rangle$ is also reachable. Inductively, when configuration $\langle q_n, \sigma_{i_n} \rangle$ is reached, $\langle q_{n+1}, \sigma_{i_n} \rangle$ is also reachable. Since $G(\mu_{n+1}) = true$, $\langle q_{\mathcal{F}}, \sigma_{i_n} \rangle$ is reachable. Along each transition, resets corresponding to local variable assignments appropriately update the values of the local variables which form part of the feature automaton state. The feature expression is computed on the transition to location $q_{\mathcal{F}}$.

Part B: The run ρ of $\mathcal{H}$ is accepted by M_F. Therefore, $\exists\ \sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_n}$ in ρ such that $i_j \leq i_{j+1}$ and $\sigma_{i_1} \vDash_\rho s_1$, $\sigma_{i_2} \vDash_\rho s_2 \ \wedge\ t_{i_1} + ... + t_{i_2-1} \in \tau_1$, and so on ... until, $\sigma_{i_n} \vDash_\rho s_n \ \wedge\ t_{i_{n-1}} + ... + t_{i_n-1} \in \tau_n$. The feature valuation computed over state σ_{i_n}, on acceptance, is γ. By Definition 5, $\mathcal{M} = \langle i_1, i_2, ..., i_n \rangle$ is a match of $\mathcal{F}$ with valuation $Eval(\mathcal{M}, \rho, \mathcal{F}) = \gamma$ in the feature range of Definition 6. □

B. Product Automaton Construction

The product construction of the HA $\mathcal{H}$ and the FA M_F yields a special type of automaton. In the classical product construction, non-determinism present in the component automata carries over to the product. The more traditional product construction, defined in [19], is conservative and reports unintentional matches, while at times missing matches that were otherwise intended. It is thus not complete. In this article a non-standard product is defined that has clearer semantics for matches than that described in earlier work. To accomplish this, we introduce first-match region semantics.

We explain this with an example. Consider the designer's intention to specify the pattern, *"P1 is true and thereafter P2 is true"*, and measure the time delay between the two. This translates to the sequence-expression "`P1,t1:=$time ##[0:$] P2,t2:=$time`", where `P1` and `P2` are PORVs over analog signals `x` and `y` respectively, and the feature computation is `(t2-t1)`. The truth intervals of the PORVs are shown as r_1, r_2, r_3, r_4 and r_5 in Figure 4. With the semantics of [19], the maximum feature value would be Δ_1 (matching points in r_1 with points in r_5). But the intent is to match points where `P2` is true immediately subsequent to points where `P1` is true, giving a maximum feature value of Δ_2. This cannot be captured by the semantics of [19]. First match region semantics matches r_1 only with r_2, giving a maximum feature value of Δ_2. Region r_4 matches with r_5. Region r_3 doesn't contribute to any match.

Definition 9. First-match Region Semantics

Given a sequence expression, S=s_1##τ_1 s_2##τ_2...## τ_{n-1} s_n*, and* $\mathbb{M} = \{\langle 1_1, 1_2, \ldots, 1_n \rangle, \langle 2_1, 2_2, \ldots, 2_n \rangle \ \ldots\}$*, the set of all matches of* S *in run* $\rho: \sigma_0 \mapsto^{t_0}_{f_0} \sigma_1 \mapsto^{t_1}_{f_1} \sigma_2 \mapsto^{t_2}_{f_2} \cdots$ *of the hybrid system* $\mathcal{H}$*,* $\langle i_1, i_2, \ldots, i_n \rangle \in \mathbb{M}$ *follows first match region semantics iff:*

$\forall_{j=2}^{n} \exists_{\theta_l \in [0, T_{i_j}]}\ \exists_{\theta_r \in [T_{i_j}, \infty)}$

$(\forall_{t<\theta_l} \langle i_1, \ldots, k \rangle$ *is not a match for* $S_1^j, T_k = t)$ *and*

$(\forall_{\theta_l \leq t' \leq \theta_r} \langle i_1, \ldots, k' \rangle$ *is a match for* $S_1^j, T_{k'} = t')$.

where, $T_m = \Sigma_{z=0}^{m-1} t_z$. □

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

The product definition presented here ensures that only runs following first match semantics reach the final location in M_F. The exclusion of other runs that would have matched in a traditional product is intentional, and imposed to accurately embody first match semantics for feature computation in FIA in the generated product. Additionally, the FA doesn't follow the traditional structure of an observer automaton for verification. These reasons taken together motivate the need for a non-standard product construction. We denote the valuation of variables in the state σ in a run ρ as $\eta(\sigma)$, and the valuation of the variable v in the state σ as $\eta(\sigma[v])$.

Definition 10. ***Level Sequenced Hybrid Automaton (LSHA) -*** *$M_F \bowtie \mathcal{H}$: The product of feature automaton $M_F = (Q_S, Z, X_H, V, C, E, Init_S, q_{\mathcal{F}})$ and HA $\mathcal{H} = (Q_H, X_H, Lab_H, Init_H, Dom_H, Edg_H, Act_H)$, is defined as the HA $\mathcal{H}_F = (Q_F, X_F, Lab_F, Init_F, Dom_F, Edg_F, Act_F)$ where,*

- $Q_F = Q_H \cup \{q_{\mathcal{F}}\} \cup \{Z \times Q_H\}$;
- $X_F = X_H \cup V \cup C \cup \{level\}$;
- $Lab_F = Lab_H$, is the set of **synchronization labels**;
- $Init_F = Init_H \times Init_S \times \{level == 0\}$;
- $Dom_F(l) = Dom_H(l) \times \mathbb{R}^{|V \cup C|+1}$ *if* $l \in Q_H$, $= \mathbb{R}^{|X_F|}$ *if* $l \in \{Z \times Q_H\} \cup \{q_{\mathcal{F}}\}$
- $Edg_F \subseteq Q_F \times Lab_F \times \mu_{\mathcal{V}(X_F) \times \mathcal{V}(X_F)} \times Q_F$ is defined by the following rules, *where* $l \in Q_H$ *and* $q_i, q_{i'} \in Q_S/Z$, μ_H *and* μ_i *are the transition relations* $\mu_H \subseteq \mathcal{V}(X_H) \times \mathcal{V}(X_H)$ *and* $\mu_i \subseteq \mathcal{V}(Q_H \cup X_H \cup V \cup C) \times \mathcal{V}(Q_H \cup X_H \cup V \cup C)$, *with* ω_i *as defined in Definition 7. The relation* $l \vDash \mu_i$, *to be read* μ_i *is applicable in* l, *is true iff* $\exists_{C^i_j \in s_i} \Upsilon^i_j == l$; *and for edge* $e = (l, a, \mu_H, l')$, $e \vDash \mu_i$ *iff for* $s_i = D_i \ \wedge \ E_i$ *either* $E_i \equiv @^-(state == l)$ *or* $E_i \equiv @^+(state == l')$. :

$$\frac{l \xhookrightarrow[\mu_H]{a} l'}{l \xhookrightarrow[\mu_H \times \mu_{ID_{\{V \cup C\}}} \times \{(level==0) \wedge (level':=level)\}]{a} l'} \tag{13.1}$$

$$\frac{l \xhookrightarrow[\mu_H]{a} l' \bigwedge q_i \xhookrightarrow[\mu_i]{} q_{i'} \bigwedge l \nvDash \mu_i}{l \xhookrightarrow[\mu_H \times (\mu_{ID_{\{level\} \cup V \cup C}} \cap \{level==i-1\})]{a} l'} \tag{13.2}$$

$$\frac{q_i \xhookrightarrow[\mu_i]{} q_{i'} \bigwedge l \vDash \mu_i}{l \xhookrightarrow[\mu_{ID_{X_H}} \times \omega_i \times \{(level==i-1) \wedge (level':=i)\}]{} l} \tag{13.3}$$

$$\frac{q_i \xhookrightarrow[\mu_i]{} q_{i,1} \bigwedge (l \vDash \mu_i \vee e \vDash \mu_i) \bigwedge q_{i,1} \in Z}{l \xhookrightarrow[\mu_{ID_{X_H}} \times \omega_i \times \{(level==i-1) \wedge (level':=i)\}]{} (q_{i,1}, l)} \tag{13.4}$$

$$\frac{q_{i,j} \xhookrightarrow[\mu_{i,j}]{} q_{i,j+1} \bigwedge q_{i,j}, q_{i,j+1} \in Z}{(q_{i,j}, l) \xhookrightarrow[\mu_{ID_{X_H \cup \{level\}}} \times \omega_{i,j}]{} (q_{i,j+1}, l)} \tag{13.5}$$

$$\frac{q_{i,j} \xhookrightarrow{\mu_{i,j}} q_{i'} \bigwedge q_{i,j} \in Z \bigwedge q_{i'} \notin Z \bigwedge e \nvDash \mu_i}{(q_{i,j}, l) \xhookrightarrow[\mu_{ID_{X_H \cup \{level\}}} \times \omega_{i,j}]{} l} \tag{13.6}$$

$$\frac{q_{i,j} \xhookrightarrow{\mu_{i,j}} q_{i'} \bigwedge q_{i,j} \in Z \bigwedge q_{i'} \notin Z \bigwedge e \vDash \mu_i}{(q_{i,j}, l) \xhookrightarrow[\mu_{ID_{X_H \cup \{level\}}} \times \omega_{i,j}]{} l'} \tag{13.7}$$

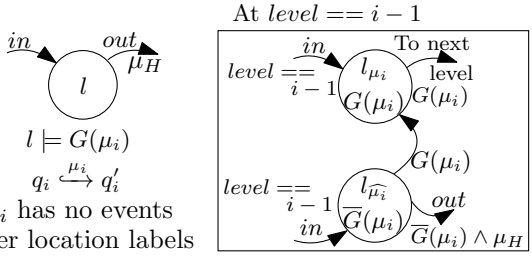

Fig. 5: Splitting for location $l \in Q_H$ when $l \vDash G(\mu_i)$, in and out represent incoming and outgoing transitions of q_H.

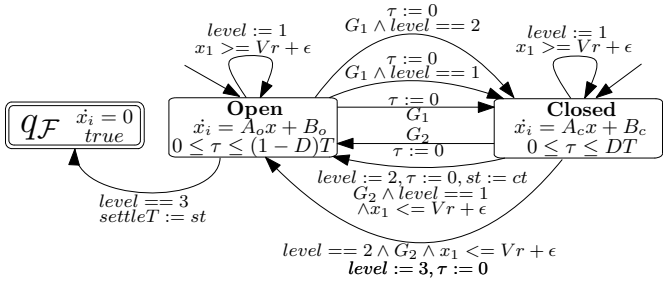

Fig. 6: LSHA for feature `Settle Time`. G_1 and G_2 are transition guards between the locations as in Figure 1.

- The function Act_F assigns a set of activities to each location. The expression associated with the flow $flow_{F_x}^q : \mathbb{R}^{\geq 0} \to \mathbb{R}^n$ for each $x \in X_F$ in location $l \in Q_F$ is defined as follows:

$$\begin{aligned} flow_{F_x}^l &= 0 \text{ for each } x \in X_F \text{ if } l \in Z \times Q_H, \\ &= 0 \text{ for each } l \in Q_F \text{ if } x \in V, \\ &= 1 \text{ for each } l \in Q_F \text{ if } x \in C, \\ &= flow_{H_x}^l \text{ if } l \in Q_H \bigwedge x \in X_H \end{aligned}$$

In order to enforce first match semantics, for each $q_i \xhookrightarrow[\mu_i]{} q_{i'} \bigwedge q_H \vDash \mu_i$, $q_i, q_{i'} \in Q_S$, $i > 1$, $q_H, q'_H \in Q_H$, at level $i-1$, q_H is replaced in Q_F according to the transformation in Figure 5. Herein, q_{H_i} is identical to q_H and differs only in the invariant as shown. $\overline{G}(\mu_i)$ is the closure of the compliment of conditions satisfying s_i. □

The behaviour asserted by the feature sequence-expression is built into the product automaton called a *level-sequenced hybrid automaton* (LSHA). In the LSHA, the level is a syntactic structure derived from the sequence-expression. The value of the variable *level* indicates how much of the sequence expression has matched. The variable $level$ is set to i when sub-expression i matches. Transitions from one level to another assign an appropriate value to $level$ indicative of the subscript of the sub-expression matched, while also executing assignments to local variables associated with the match. Initially, when $level$ is 0, corresponding to location q_1 of M_F, the automaton waits for a match of s_1. When s_1 matches, $level$ is non-deterministically incremented. Due to first-match semantics, a constrained form of non-determinism is applicable when $level > 0$. The non-determinism in the control allows computation of a continuum of matches. When the feature has matched (ending with the match of the last sub-sequence), the feature is computed and control moves to the final location of M_F.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

The use of variable $level$ in $\mathcal{H}_F$ allows us to avoid the typical blow-up that results from the standard product construction used in [19]. Given a HA with N locations and a feature F with K sub-expressions, ignoring pause locations, the product automaton in [19] has $N \times (K+1)$ locations, while the one here has *atmost* $N \times (1 + 2 \times (K-1))$ locations. Note that in [19] first-match semantics isn't used. Also, the number of locations in the LSHA here would reduce to $N + K + 1$ if sub-expressions contain events over location labels, and further to $N + 1$ if restrictions on monotonicity are lifted. We observe that, although the theoretical worst case bound for the product defined here is worse than that of a traditional product, in practice the use of leveling enables faster analysis of features. For instance without the variable $level$ the vanilla product from [19] (where location copies are made for each level) when analyzed by SpaceEx, under equivalent analysis settings, takes 1m:30s and twice the memory with 7 locations as opposed to the 20s with 3 locations and the variable $level$ for the `Settle Time` feature in Example 1. The LSHA for `Settle Time` is shown in Figure 6.

C. Feature Evaluation

The product automaton is, by construction, a HA. We compute all values of $\mathcal{F}_{name}$ reachable in location $q_{\mathcal{F}}$ of $\mathcal{H}_F = M_F \bowtie \mathcal{H}$. Using reachset computation tools on $\mathcal{H}_F$, a projection of the entire reachset $\mathcal{R}$ in location $q_{\mathcal{F}}$ on $\mathcal{F}_{name}$ gives us an overapproximation of the reachable range of values for $\mathcal{F}_{name}$. A reachset computation tool may computes $\mathcal{R}$ in various ways. In this article we use tool SpaceEx. The feature range $[\mathcal{F}_{min}, \mathcal{F}_{max}]$ is computed as follows:

$$\mathcal{F}_{min} = \min_{\forall \sigma \in R} \eta(\sigma[\mathcal{F}_{name}]);$$
$$\mathcal{F}_{max} = \max_{\forall \sigma \in R} \eta(\sigma[\mathcal{F}_{name}]);$$

where $\eta(\sigma[\mathcal{F}_{name}])$ is the valuation of $\mathcal{F}_{name}$ in state σ.

IV. Method-2: Feature Analysis of Corner Cases

Methodology-1 demonstrates how reachability analysis tools are used to compute estimates on the range of feature values. However, these tools do not show how extremal values can be reached. Additionally, due to over-approximation errors, corners of the feature interval generated by these tools may not be realizable. SMT solvers can generate reachability proofs for a reachable goal state. For a feature, this means that a proof of how feature value $\hat{f}$ is realized is constructed in terms of a concrete trace for which the evaluating the feature yields $\hat{f}$. SMT solvers for reals [24] use decision procedures that use overapproximation techniques. The analysis is bounded in the values of SMT variables and in the number of discrete automaton transitions (a hop bound). Hence the outcome is an overapproximation of a bounded reachability question. If realistic and sufficiently large bounds are used, the boundedness is acceptable, since for most realistic systems the domains of variables in the HA model are bounded. To improve reliance on the results obtained, bounds used must come from knowledge of the design and results must be interpreted in terms of these bounds. Due to overapproximations, the proof of reachability for a goal in SMT could be fictitious, nevertheless it provides insight to build simulations to verify the reported scenario. In practice we find that overapproximations produced by reachset computation tools like SpaceEx are larger than that produced when using SMT solvers.

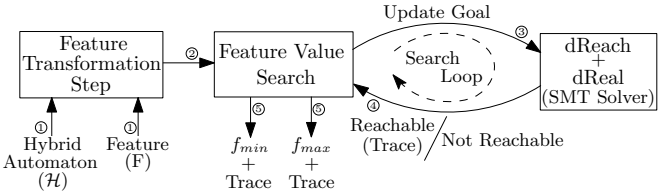

Fig. 7: Outline of the Feature Value Search using SMT.

It is important to note that application of Methodology 1 in Section III reduces the problem of feature analysis to a reachability question. The generality of the algorithm allows it to be used with a variety of reachability analysis tools. SMT solvers, by nature, are less inclined to perform flow-pipe analysis (which generates an overapproximation of the state-space) and more inclined towards finding a *single* run that satisfies a goal constraint. This therefore becomes a challenge when we relate the notion of identifying the interval of values a feature can take for the HA, when using SMT solvers. We answer the following questions:

1) How would a reachability question for computing the feature interval be posed as an SMT solver goal?
2) Any such goal will only yield a single feature value, and not a range. How would one then compute the extreme feature values?

Once the range of feature values is identified using SMT, the SMT solver can provide a satisfying trace, thereby solving the input selection problem for analyzing corner cases for features.

A summary of the methodology used to compute the feature range using SMT is shown in Figure 7. The feature to be analyzed is expressed in FIA. The HA model along with the feature is taken through the transformation step (Methodology-1). The tuned model is an implicit representation of all legal executions of the automaton, biased toward computing the feature attribute.

The SMT question about feature value f is: *Is there a run of the automaton that results in feature value f?* On the other hand, the Feature Range Analysis must answer the question: *What is the range of feature attribute values for $\mathcal{H}$?* To bridge this gap, we use a two part reduction described in Sections IV-A and IV-B.

A. SMT based modelling of the Hybrid Automaton Dynamics

The HA, along with the various model constraints such as locations, their dynamics and invariants, transitions between locations and transition guards and resets, are modeled as clauses in SMT. We use the translator dReach [25], which in turn uses SMT solver dReal [24] for modeling and analyzing hybrid behaviours over reals. dReal internally maintains the coupling between HA variables during its computation steps using these constraints. We now discuss the caveats of the decision outcomes presented by dReach/dReal. dReal solves the δ-*decision problem*, to decide if a given formula is false

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

or δ-true (dually, whether it is true or δ-false). An SMT formula is δ-true if it remains true under δ-bounded numerical perturbations to atomic clauses in the formula [24]. For a feature, this means that a feature goal is reachable under δ-bounded numerical perturbations to the goal, and sentences describing the system [25]. Since realistic hybrid systems interact with the physical world, it is impossible to avoid slight perturbations. Hence, this is a very useful result as it gives feature values that are reachable under reasonable choices for δ [24]. The δ-decision problem has been shown to be decidable for first order sentences over bounded reals with arbitrary Type 2 computable functions (real functions that can be approximated numerically, such a polynomials, trignometric functions, and Lipchitz-continuous ODEs). dReal guarantees the result of unsatisfiability of a goal G over K transitions (hops) with δ perturbations on sentences describing the system.

The model is *unrolled* in terms of number of transitions, upto the given bound K. For instance, if our goal were to reach location $q_{\mathcal{F}}$ in the location graph of Figure 6, a minimum of five transitions would be required, resulting from an unrolling of the model six times starting with location *Closed*, to reach location $q_{\mathcal{F}}$, reachable only when $level$ is 3, as shown below:

$$\langle Closed, level = 0\rangle \rightarrow \langle Closed, level = 1\rangle \rightarrow \langle Open, level = 2\rangle$$
$$\langle q_{\mathcal{F}}, level = 3\rangle \leftarrow \langle Open, level = 3\rangle \leftarrow \langle Closed, level = 2\rangle$$

The encoding of a HA as SMT clauses for dReal can be found in Ref. [25].

B. Feature Range Exploration

To compute the extremal feature values and their corresponding traces, search techniques are used to explore the feasible set of feature values and progressively refine the corners of the feature range. SMT solvers take a *goal* statement, and a hop bound as input and respond indicating whether or not the goal is reachable. The feature range computed on a hop-bounded SMT-based search is therefore an under-approximation of what is obtained using an infinite trace length. An increase in the bounds can result in a larger feature range. A low value of the hop bound K can yield a severely under-approximated feature range (ignoring feature values reachable via transition paths longer than K). In Section V we discuss a heuristic for choosing a value for K. Using an appropriately large K ensures that the computed feature range safely over-approximates the reachable interval of feature values.

To begin, no feature value $\mathcal{F}$ is initially known. If a behaviour contributing to the feature exists, its feature value will either be positive (including zero), or negative. Therefore, initially the search uses goals $\mathcal{F} < 0$ and $\mathcal{F} \geq 0$ as pivots about which to begin. The algorithm pushes a pivot as far as possible in each direction to find the corners of the feature range. However, since it is not known how *far* to push the pivot, a combination of exponential expansion and bisection is used to identify the corners of the feature range.

For pivot value f^*, a goal that trivially checks if $\mathcal{F} < f^*$ may return a feature value very close to f^* resulting in repeatedly finding values of f^* that are within close proximity

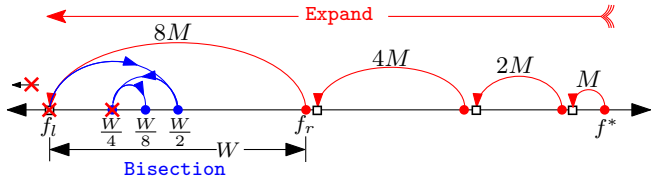

Fig. 8: Using `Expand-Bisect` to compute the left corner

.

of each other. Therefore the algorithm explores the feature space in steps of size $2^{i-1} \times M$, where i is the i^{th} expansion step. M is a search parameter chosen by the designer. For instance, in the computation of the settling time for a buck regulator, $M = 10^{-6}$, because the feature value is in the order of μs. Expansion tries to find a feature value further than the current pivot f^* by an amount M, and pushes the new value further by a recursive call to itself on the new pivot with a step size of $2 \times M$. When no new pivot is found, a bounded interval search ensues to find a feature value within a distance of M from the last known pivot f^*. The search refines the interval tapering it towards the corner of the feature range. Figure 8 depicts this strategy, starting at a pivot f^* with boxes indicating steps during the expansion, red circles are new pivots, and red crosses indicate no feature value found. The bisection search terminates when the interval containing the feature corner has a width of ϵ (the error tolerance) or less.

A bounded interval search for the left corner is now described. For the interval $W = [f_l, f_r]$, the midpoint mid is used as a pivot. The algorithm looks for a feature value, f^*, in the leftmost interval $[f_l, mid]$. If found, it proceeds to search the interval $[f_l, f^* - \epsilon]$. The shrinking of the right interval boundary by ϵ is consistent with the precision requirements of the algorithm and excludes the already found feature value f^* from the search. If no feature value is found in $[f_l, mid]$, then the search moves to $[mid, f_r]$. If a feature value is found in $[mid, f_r]$, then it forms the basis of a new pivot and the algorithm is recursively called on the interval $[mid, f^* - \epsilon]$. Every interval search uses the last feature value found as the pivot and excludes it from future searches. If no new feature value is reachable about the last pivot, then the last known reachable feature value and its corresponding trace (returned by the SMT solver as evidence of a satisfiable instance) are returned. The steps of exploration for the right corner mirror those for the left corner.

Testing if a feature value f^* is reachable involves invoking the SMT solver to answer the question specified as the goal G in the context of the model $\mathcal{H}_F$. A 'YES' answer of the solver returns a trace, ensuring that the dynamics, invariants and guard conditions of $\mathcal{H}_F$ are not violated. A 'NO' answer, returns an `EMPTY` indicating that no behaviour of $\mathcal{H}_F$ yields a feature value specified in G.

If an interval for the feature is known, via a primary reachability analysis using a coarse overapproximation in a tool like SpaceEx, then the bounded bisection algorithm can be applied to search for the feature corners within the known estimate of the feature range.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

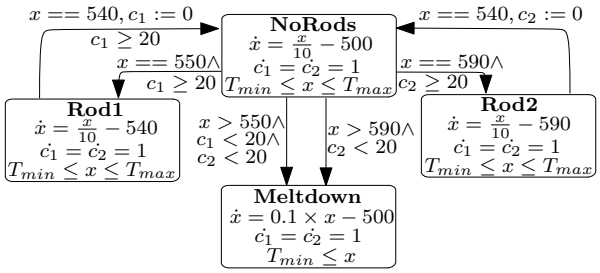

Fig. 9: Temperature Control of an Atomic Reactor.

V. Case Studies and Experimental Results

This section discusses the various models we have used in our analysis, and data from our results, comparing the feature analysis of HA using the reachability analysis tool SpaceEx versus using the SMT tool dReach. We compare the results of analyzing the following models and features:

1) **Battery Charger [19]:** Time for the battery to charge to its rated voltage; Time for the battery to restore charge while in its maintenance mode.
2) **Buck Regulator [23]:** Time for the output of the Buck regulator to settle; Peak voltage overshoot of the voltage response curve for the regulator.
3) **Nuclear Reactor Temperature Control [26]:** Unsafe Operating Temperature of the reactor.
4) **Adaptive Cruise Control [27]:** Time to capture cruise speed from a specific velocity; Time to capture cruise speed while in any velocity within a range of velocities.

Here, we pay special attention to the condition of meltdown for an atomic reactor cooling strategy. The temperature control strategy for a nuclear reactor [26] is designed to insert a cooling rod into the reactor with the aim of maintaining the temperature of the reactor below the threshold for meltdown and above the threshold for sustaining the nuclear reaction. Mechanical constraints prevent both rods from being inserted simultaneously, and requires each rod to be given a resting period of 20 time units before re-insertion. An adaptation of the HA of [26] is shown in Figure 9. The feature for analyzing the condition of meltdown is expressed as follows:

Example 2. *Unsafe Operating Temperature:* Reactor temperatures that if reached can lead to reactor meltdown.

```
feature unsafe();
begin
    var temp;
    (c1<=20 && c2<=20 && x>=550), temp = x
        ##[0:$] (c2<=20 && x>=590)
    |-> unsafe = temp;
end
```

The condition of meltdown occurs when the reactor temperature, x, rises above the safe threshold, T_{safe}. The reactor can be in a state in which $x < T_{safe}$, but has crossed a point-of-no-return, i.e. neither control rod can be inserted, inevitably leading to a state of reactor meltdown. A safety property that checks for the safe operation of the reactor, with a traditional model checking approach, would only yield one of the many possible failures. However, it is of greater interest to identify the minimum temperature at which such a failure can occur. Knowledge of this corner enables a designer to design a suitable strategy for managing the rods. A feature analysis, unlike traditional model checking, yields these corner cases. Furthermore, for such a feature where only boundary events characterizing a failure are known, the second technology proposed (feature analysis with SMT) can provide the precise event sequence that yields a feature match, thereby filling the gap specified in the `##[0:$]` construct. Figure 10 shows a corner case obtained using the methodology of Section IV, in which the red vertical line marks a point-of-no-return. Observe that from this point `x` rises, passing through *safe* temperatures and beyond into the unsafe region of meltdown. In this scenario, both rod-timers are below their thresholds, preventing their insertion.

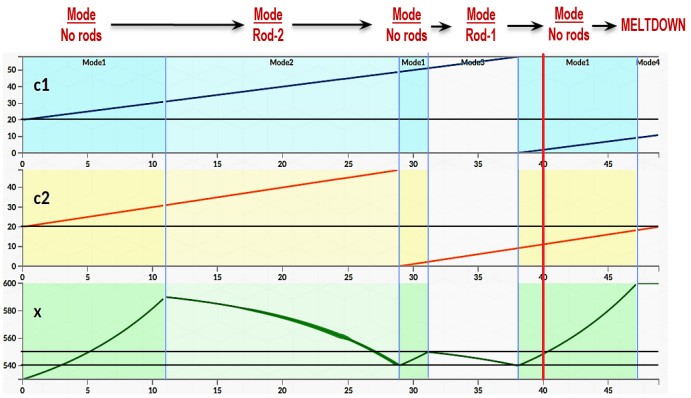

Fig. 10: Unsafe Operation of a dual rod temperature control in an atomic reactor: an extreme value trace.

Note that in Examples 1 and 2, both features used a single local variable, but were able to express very interesting behaviours. However, in general more local variables may be required to express complex quantities over more intricate behaviours [14].

A feature based formal analysis was performed on the models and features outlined, the results of which are described in Table I. Table II compares the results of using various SpaceEx analysis parameters for analyzing the `Overshoot` feature of a 7V Buck Regulator. We demonstrate the analysis of both strategies for feature analysis on four systems that cover both the AMS domain and the control domain. Both strategies have been implemented in a unified tool-flow. The tool is run on an Intel(R) Core(TM) 2 Duo CPU T6400 having two cores, each running at 2.00Ghz with 4GB of DDR2 RAM. For each system, the HA model and the features described are inputs to the tool. The tool then computes the LSHA for feature analysis. The feature range is then computed using the reachability tool SpaceEx (with the STC scenario and 8 template directions, octagons), and the SMT-based search using Methodology-2.

In Table I, the size of the transformed automata in terms of the number of locations (in set Q_F) and number of variables (in set X_F) are also shown. The column titled *Algorithm* indicates how the feature range was computed, with "Reach" indicating the use of Methodology-1 with SpaceEx as the compute engine, and "SMT" indicating the use of Methodology-2

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Feature Name	Size of Set Q_F	Size of Set X_F	Algo	CPU-Time (m : s)	Feature Range Min	Feature Range Max
Test Case: Battery Charger						
Charge Time	9	7	**Reach**	0:10	1h49m	3h15m
			SMT	0:20	2h15m	2h31m
Restoration Time	9	7	**Reach**	0:19	10m12s	48m58s
			SMT	0:15	16m40s	18m30s
Test Case: Cruise Control Model						
Speed Capture Precise (k=40)	10	8	**Reach**	0:19	37.23s	48.43s
			SMT	1:16	41.18s	43.68s
Speed Capture Range, (k1=20, k2=40)	10	8	**Reach**	0:39	33s	48.43s
			SMT	0:25	37.3s	40.4s
Test Case: Nuclear Reactor Control						
Unsafe Operation Temperature	8	6	**Reach**	0:07	549.9°	599.9°
			SMT	0:52	550°	600°
Test Case: 5V Buck Regulator						
Settle Time	4	7	**Reach**	0:16	94.17μs	124.167 μs
			SMT	*Out of Memory*		
Overshoot	4	7	**Reach**	0:03	5V	6.138V
			SMT	69:45	5V	5.14V

TABLE I: Results for Formal Feature analysis.

No. of Template Directions	Flow-pipe Tolerance	SpaceEx Algo	Iterations Taken	CPU-Time (secs)	Feature Range Min	Feature Range Max
4	1	STC	37	0.22	1.76V	12.65V
		LGG	11	0.24	4.41V	9.96V
	0.1	STC	36	0.22	5.14V	12.11V
		LGG	55	0.63	6.63V	9.90V
	0.01	STC	60	0.52	6.78V	9.12V
		LGG	57	0.82	6.91V	9.0V
8	1	STC	85	4.53	2.00V	12.04V
		LGG	11	1.52	4.41V	9.32V
	0.1	STC	28	0.66	5.17V	12.06V
		LGG	55	2.75	6.63V	9.19V
	0.01	STC	72	3.77	6.82V	9.08V
		LGG	57	4.78	6.92V	9.0V

TABLE II: SpaceEx analysis of `Overshoot` of a 7V Buck Regulator.

to refine the range computed using the former. For each feature both corners of the feature range are reported along with the time taken to compute the range for each methodology. Both SpaceEx and the SMT analysis are bounded by introducing a global clock with an upperbound on time. The global clock is a variable that is part of the state of the LSHA. In SpaceEx, with the STC scenario, given the way SpaceEx does fix-point computation this variable must be bounded to achieve termination. Note that the need for the bound comes from the tool used and is not a limitation of this methodology. For the SMT analysis, in addition to a bound on the variables, we use a transition hop bound (K) of 15 transitions for all test cases except for the Buck Regulator, for which a bound of 50 transitions was used. In practice K is incrementally increased until we are satisfied with the result. We use knowledge of the diameter of the LSHA graph and the number of subexpressions in the feature sequence expression to decide on a value for K. We reiterate that K is a bound on the discrete transitions of the HA. Within a location a large number of clauses may be generated for the evolution of the HAs continuous variables. It is important to note that the SpaceEx tool computes the feature range in one sweep of the reach set; however, multiple iterations of the SMT tool (between 15 to 20 in our experiments) are involved in computing the feature range using "SMT". Additionally for the feature computing the *Unsafe Operation Temperatue*, the feature ranges produced by SpaceEx and the SMT tool show errors of 0.1. We attribute this to the precision of representation for floating-point numbers used by the tools.

Note that for the feature "Settle Time" of the Buck Regulator, the methodology using SMT exceeds memory bounds on our systems. We attribute this to the fact that the Buck Regulator frequently switches between locations of the automaton, with more than 50 transitions made within a very short span of time (time from the perspective of the Buck Regulator). The solver takes an inordinate amount of time to compute this. Due to the large number of transitions taken, the number of SMT clauses generated becomes too large for the solver to handle and leads to the solver running out of memory. We conclude that for systems having a high switching frequency, SpaceEx can be used with a resolution smaller that 10^{-6}, for which it takes in the order of a few seconds to a few minutes to compute the feature range (depending on the chosen resolution).

For the models used here, it is shown that the feature range produced by the SMT solver is typically tighter than that obtained using SpaceEx. Both methodologies were employed using similar error tolerances. Note that the methodology using the SMT solver requires more CPU resources as indicated by a higher value in the column for *CPU-Time*. The feature transformation methodology itself scales well with reachability tools and SMT. The time for analysis is dependent on the tools used. The tool SpaceEx has been used extensively for the analysis of HA and scales well for the models on which we have demonstrated the feature analysis approach. SpaceEx, in benchmarks has shown to be capable of handling systems with more than 100 variables [10]. The time and memory to compute a feature range grows exponentially with an increase in the hop bound when using SMT, and is attributed to a growth in the number of SMT clauses for larger hop bounds.

VI. Conclusion

Features help capture the designer's intent to quantify how the system behaves. A feature defines a real-valued evaluation function over a specific set of traces. By design they are more flexible than assertions (such as in STL) for specifying quantitative measures, at the cost of being more rigid in their expression of sets of traces (restricted to sequences of predicates and events).

This article aims to assist designers in generating better designs, by automating the task of feature analysis and providing useful feedback of corner case behaviours using SMT. Features are automatically transformed into feature automata that are composed with the model and the composition is analyzed by off-the-shelf reachability solvers. The improved first-match semantics employed for features in this article more directly reflects the intent of designers and is incorporated into a

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857361, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

new product automaton construction. Although the worst case bounds for the proposed product construction (using leveling) are worse than those of a more traditional product, in practice we see a 4x speedup and half the memory utilization during feature analysis with reachability solvers. We also provide an algorithm for computing ranges of feature values that uses SMT, which in practice produces tighter feature ranges. In some cases, typically associated with models having fewer location switches, the SMT-based algorithm also yields results faster or in time comparable to SpaceEx. The present work assumes piecewise monotonic and piecewise affine dynamics to accommodate existing tools. These assumptions can be lifted as tools mature to support urgent semantics and more complex dynamics. Efforts for such extensions [28] are underway.

REFERENCES

[1] R. Alur *et al.*, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, pp. 3–34, 1995.

[2] A. Chutinan and B. Krogh, "Computational techniques for hybrid system verification," *IEEE TAC*, vol. 48, pp. 64–75, 2003.

[3] T. Dang *et al.*, "Verification of analog and mixed-signal circuits using hybrid system techniques," in *Proc. of FMCAD*, ser. LNCS, 2004, vol. 3312, pp. 21–36.

[4] *1800-2012 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, IEEE Std., 2012. [Online]. Available: http://standards.ieee.org/findstds/standard/1800-2012.html

[5] *1850-2010 - IEEE Standard for Property Specification Language (PSL)*, IEEE Std., 2010. [Online]. Available: https://standards.ieee.org/findstds/standard/1850-2010.html

[6] S. Mukherjee and P. Dasgupta, "Incorporating local variables in mixed-signal assertions." in *IEEE Int. Conf. TENCON*, 2009.

[7] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. of FORMATS-FTRTFT*, vol. 3253, 2004, pp. 152–166.

[8] S. Steinhorst and L. Hedrich, "Model checking of analog systems using an analog specification language," in *Proc. of DATE*, 2008, pp. 324–329.

[9] H. Roehm *et al.*, "STL model checking of continuous and hybrid systems," in *Proc. of ATVA 2016*, 2016, pp. 412–427.

[10] G. Frehse *et al.*, "SpaceEx: Scalable Verification of Hybrid Systems," in *Proc. of CAV*, 2011.

[11] J. Ouaknine and J. Worrell, "Some recent results in metric temporal logic," in *Proc. of FORMATS*, pp. 1–13.

[12] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proc. of FORMATS*, 2010, pp. 92–106.

[13] S. Jha *et al.*, "Safe autonomy under perception uncertainty using chance-constrained temporal logic," *J. Autom. Reason.*, vol. 60, no. 1, pp. 43–62, Jan. 2018.

[14] A. Ain *et al.*, "Feature indented assertions for analog and mixed-signal validation," *IEEE TCAD*, vol. 35, no. 11, pp. 1928–1941, Nov 2016.

[15] S. Jha *et al.*, "Telex: Passive STL learning using only positive examples," in *Proc. of Runtime Verification 2017*, 2017, pp. 208–224.

[16] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Proc. of CAV*, ser. CAV'10, 2010, pp. 167–170.

[17] Y. Annapureddy *et al.*, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Proc. of TACAS*, 2011, pp. 254–257.

[18] A. A. B. da Costa and P. Dasgupta, "Formal interpretation of assertion-based features on AMS designs," *IEEE Design & Test*, vol. 32, no. 1, pp. 9–17, 2015.

[19] A. A. B. da Costa, P. Dasgupta, and G. Frehse, "Formal Feature Analysis of Hybrid Automata," in *Proc. of MEMOCODE*, 2016.

[20] A. A. B. da Costa and P. Dasgupta, "ForFET: A Formal Feature Evaluation Tool for Hybrid Systems," in *Proc. of ATVA*, 2017, pp. 437–445.

[21] ——, "Generating AMS Behavioral Models with Formal Guarantees on Feature Accuracy," in *Proc. of VLSID*, 2017.

[22] E. Asarin *et al.*, "Hybridization methods for the analysis of nonlinear systems," *Acta Inf.*, vol. 43 (7), pp. 451–476, Jan. 2007.

[23] L. V. Nguyen and T. T. Johnson, "Benchmark: DC-to-DC Switched-Mode Power Converters," in *ARCH14-15*, vol. 34, 2015, pp. 19–24.

[24] S. Gao *et al.*, "dReal: An SMT Solver for Nonlinear Theories over the Reals," in *Automated Deduction - CADE-24*, 2013, pp. 208–214.

[25] S. Kong *et al.*, "dReach: δ-Reachability Analysis for Hybrid Systems," in *TACAS*, 2015, pp. 200–205.

[26] R. Alur, T. A. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embedded systems," vol. 22, pp. 181 – 201, 04 1996.

[27] E. Abraham, "Benchmarks of continuous and hybrid systems," 2015. [Online]. Available: http://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

[28] S. Minopoli and G. Frehse, "From simulation models to hybrid automata using urgency and relaxation," in *Proc. of HSCC, 2016*, 2016, pp. 287–296.

Antonio Anastasio Bruto da Costa (S'15) received the Bachelor of Engineering degree in Computer Engineering from the Goa Engineering College in 2010. He has an M.Tech. in Computer Science and Engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, where he is currently pursuing his Ph.D degree.

His research interests include Formal Verification, Distributed Computing and Artificial Intelligence.

Goran Frehse (M'16) received a Diploma (M.S. equivalent) in Electrical Engineering and Information Technology from Karlsruhe Institute of Technology, Germany, and a PhD in computer science from Radboud University Nijmegen, Netherlands. Since Sept. 2006, he is an associate professor at Univ. Grenoble Alpes.

He is the architect and lead developer of two well-known model checking tools for hybrid and cyber-physical systems, PHAVer and SpaceEx. Since 2016, he holds a research fellowship (Chaire Initiative Universitaire Alpes) at the Univ. Grenoble Alpes.

Pallab Dasgupta (SM'99) received the B.Tech, M.Tech and Ph.D degrees in computer science and engineering from the Indian Institute of Technology Kharagpur (IIT Kharagpur), Kharagpur, India. He is currently a Professor with the Department of Computer Science and Engineering, IIT Kharagpur, where he is currently the Dean of Sponsored Research. His current research interests include formal verification and computer-aided design.

Prof. Dasgupta is a Fellow of the Indian National Academy of Engineering and a Fellow of the Indian Academy of Science.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.