Quantitative Corner Case Feature Analysis of Hybrid Automata with ForFET-SMT

Antonio Anastasio Bruto da Costa, Pallab Dasgupta, Nikolaos Kekatos

Quantitative Corner Case Feature Analysis of Hybrid Automata with ForFET^{SMT}

Antonio Anastasio Bruto da Costa[1], Pallab Dasgupta[1], and Nikolaos Kekatos[2]

[1] Dept. of Comp. Sci. and Engg., Indian Institute of Technology Kharagpur, India
[2] Verimag, Univ. Grenoble Alpes, France

Abstract. The analysis and verification of hybrid automata (HA) models against rich formal properties can be a challenging task. Existing methods and tools can mainly reason whether a given property is satisfied or violated. However, such qualitative answers might not provide sufficient information about the model behaviors. This paper presents the ForFET^{SMT} tool which can be used to reason quantitatively about such properties. It employs *feature automata* and can evaluate quantitative property corners of HA. ForFET^{SMT} uses two third-party formal verification tools as its backbone: the SpaceEx reachability tool and the SMT solver dReach/dReal. Herein, we describe the design and implementation of ForFET^{SMT} and present its functionalities and modules. To improve the usability of the tool for non-expert users, we also provide a list of quantitative property templates.

1 Introduction

Formal verification techniques can provide guarantees of correctness and performance for hybrid and cyber-physical systems (CPS). There are several robust verification tools for CPS, e.g. SpaceEx [12], dReal/dReach [13] and their goal is to guarantee that specifications are satisfied through a rigorous mathematical analysis of the system. An appropriate modeling formalism for the design of such systems is Hybrid Automata [2] (HA). HA can exhibit nondeterministic behaviors and have been used to model control systems and analog mixed-signal (AMS) circuit designs [4, 9].

Despite the progress made on formal verification algorithms, it is not easy to formalize specifications of hybrid automata such that they can be verified automatically. One main reason concerns the semantic mismatch between industrial requirements and formal requirements. Typically, formal specifications are expressed in a formal language, like temporal logic, whereas industrial requirements are described in natural language. Much literature exists on temporal logic especially on Linear Temporal Logic (LTL) [19]. Languages such as MITL [3], STL [17] and its extensions such as xSTL [18] have been used for specifying specifications involving continuous signals. Robustness measures have also been defined over properties written in these languages [10]. Robustness metrics defined for an STL/MITL property measure the *distance* of runs of the system from unsafe regions defined by the property. These metrics may be leveraged to express quantitative measures such as *overshoot*, *settling time*, or other

timing and *value* quantities. The parameterized version of STL (PSTL) allows temporal and predicate constants to be parameterized, transforming the quantitative analysis into parameter learning. These languages are primarily designed to express specification correctness, and it can be tedious to use them to express quantitative measures. Unlike temporal logics like MITL, STL (or PSTL), the language of *features* [1] is designed to explicitly specify quantitative measures. The quantity is expressed as a computation resulting from matching a specified behaviour.

While standard analysis tools [12] support answering reachability questions, they do not implicitly handle features. This is typically overcome by constructing monitor automata for the property and taking its product with the HA [11, 15]. The resulting automaton can be large resulting in long analysis times and scalability issues. *ForFET* [6] is a tool for computing an over-approximation for features, where the evaluation of a feature, written in the *Feature Indented Assertion* (FIA) language, over runs of a HA is automated.

Features assume urgent match semantics. This is reflected in the product construction of a feature monitor automaton and HA. However, specifications contain *mixed urgent and non-urgent semantics*. For instance, *reaction-time*, defined as "the time elapse, after q is true, from a time-point when p becomes false to the first time-point when s is false", contains an enabling action q, the source from which measurement starts p (non-urgent) which can occur multiple times in the trace, and the next s (urgent). Furthermore, developing formal specifications requires expertise and investment of time and effort. Having a library of *standard feature specification templates* would facilitate formulating quantities to evaluate the design. In a quantitative analysis, knowledge of the stimulus that produces the best and worst-case quantity (minimum or maximum) provides insight into the system and on how to modify the design to more robustly adhere to specifications.

The tool ForFET^{SMT} addresses the needs described above, extending ForFET, with the following:

- Feature corner analysis using SMT.
- Support for parameterized features and an extended language for features having mixed urgent and non-urgent semantics.
- Library of standard feature specification patterns.
- Usability and support: i) two translators, written in Matlab and Octave, for converting models from SpaceEx formalism to ForFET's modeling language, ii) richer interaction environment, iii) support for custom paths for workspace, models, and third-party tools.

2 ForFET^{SMT} Tool Design

ForFET^{SMT} is a corner case feature evaluation tool[3] for hybrid automata. Features are quantitative measures computed over runs of a HA, providing more information than properties by explicitly indicating the robustness of the system in the context of a design quantity. Additionally, ForFET^{SMT} can provide

[3] Available at the repository `https://github.com/antoniobruto/ForFET2`

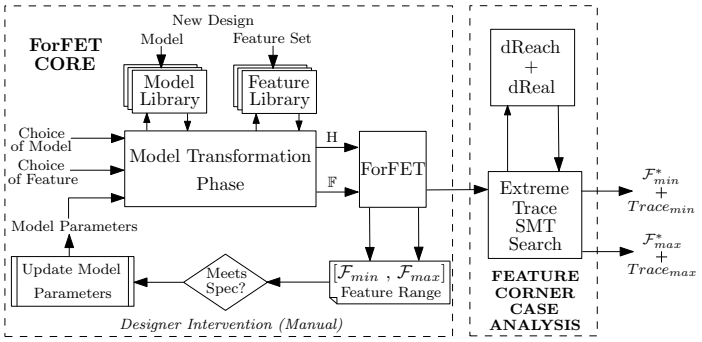


Fig. 1: Methodology Outline – Feature Corner Case Analysis

a concrete trace for each corner of the quantity. ForFET^{SMT} is developed in C and uses reachability analysis tool SpaceEx [12], model language translator Hyst [5] and SMT analysis tool dReal [13]. The tool ForFET^{SMT} is outlined in Figure 1.

2.1 Translators from SpaceEx to HASLAC

The SpaceEx modeling language has become the quasi-standard interchange format for defining and describing HA in the formal verification community [4]. It offers a graphical user interface, respects the SX grammar and the models are written as XML files [8]. ForFET^{SMT} accepts HA models written in HASLAC. To bridge this mismatch and facilitate the use of ForFET^{SMT} with existing SpaceEx models and HA benchmarks, we provide two translators, written in MATLAB and Octave respectively. The translators require a SpaceEx model (necessary) and a configuration file (optional). They come with an XML parser (partly written in Java) and perform syntactic translation while also handling modeling differences.

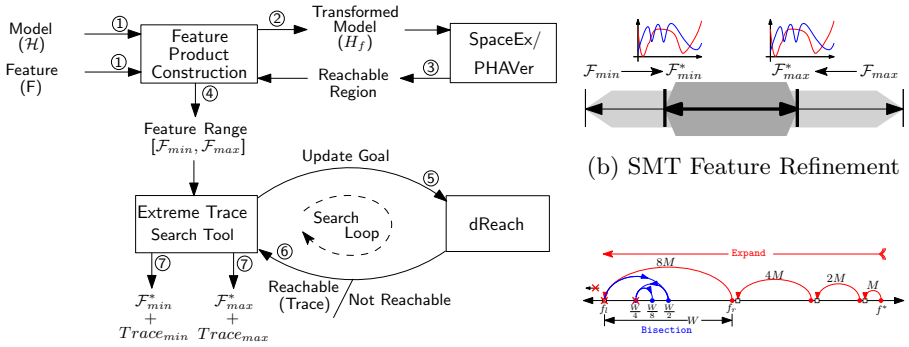
2.2 Mixed Urgent/Non-Urgent Semantics

ForFET assumes urgent semantics when deciding matches of feature components. We add the keyword *first_match* to the FIA syntax to explicitly indicate urgent semantic interpretations. This enables features having mixed urgent and non-urgent interpretations to be expressed. The usefulness of including mixed semantics is demonstrated through the standard feature specification patterns[4].

2.3 Standard Feature Specification Patterns

In [16, 14, 11], the authors provide specification templates (also called pattern templates) to describe commonly used natural language specifications. Using such templates could assist experts and non-experts when translating properties into formal specifications. As such, we provide a library of feature specifications based on the notion of standard patterns for real-time systems [16]. These features are parameterized. For a new design, it is straightforward to generate an instance of the feature, for which ForFET^{SMT} automates the translation into monitors, formally defined as feature automata. These feature automata are in-

[4]See attached tool manual for ForFET^{SMT}



(a) Functional Overview

(b) SMT Feature Refinement

(c) Computing the left corner

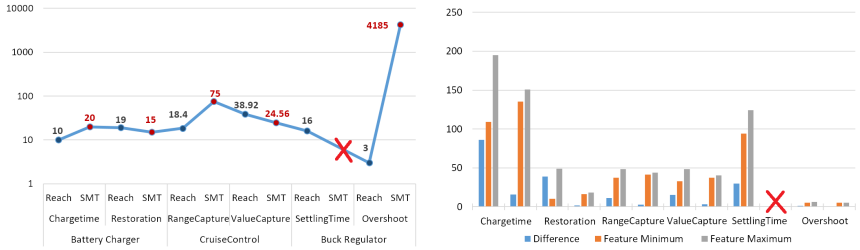
Fig. 2: ForFET^{SMT}: *Corner Case Analysis*

tegrated with the HA model under study via parallel composition and then used for formal analysis.

ForFET^{SMT} includes templates for standard timing features[4] for *invariance*, *absence/manifestation*, *response*, *reaction*, *duration*, and *separation*. The quantities representing the feature evaluation is an indication of the behaviour of the system. We can use this evaluation in the following two ways: (i) the condition of *"no-match"* for a feature can indicate when a specification is satisfied, (ii) while on the other hand, the upper or lower bound of the feature range can indicate failures of a specification.

2.4 Corner Case Analysis

A functional overview of ForFET^{SMT} is shown in Figure 2. The user should provide two inputs (step 1): a hybrid automaton model $\mathcal{H}$ and a feature specification $\mathcal{F}$ (single or a set of features), ForFET^{SMT} computes the product automaton (Step 2) according to [7]. Step 3 involves using SpaceEx [12] to compute reach-sets for the transformed model $\mathcal{H}_F$. This results in a feature range $[f_{min}, f_{max}]$ computed as an evaluation of the feature expression on the runs matching the feature sequence-expression (Step 4). The feature range is refined iteratively through a search using an SMT solver (Steps 4 to 7). HyST [5] converter is used internally to translate the model $\mathcal{H}_f$ into an acceptable format for use with dReach. In each interaction, called a *query*, between our tool and dReach, a goal statement is constructed to direct dReach to prove the existence/non-existence of a feature value in a given domain. Each query in Step 5 includes the model description for $\mathcal{H}_f$, a goal statement, and a maximum transition hop count K, which is translated by dReach into SMT clauses. The response of the SMT solver (Step 6) is unsatisfiable or a single timed trace of the hybrid automaton if satisfiable. dReal generates a trace as a JSON file with time-stamped valuations for the variables of the automaton, which is parsed to identify the feature values for the trace. The search concludes in Step 7 with a refined feature range $[f^*_{min}, f^*_{max}]$ as well as a trace corresponding to each feature range corner value.



(a) Time to compute feature range (b) Feature range

Fig. 3: Computing features using SpaceEx and dReach/dReal in ForFET^{SMT}

Challenges using dReal: In general, HA use urgent locations to represent ordered discrete transformations. In a trace, dReal provides a series of indexed time-ordered tuples representing a trace satisfying the query. In our experience, when the model $\mathcal{H}_f$ contains urgent locations, dReal generates a `NULL` tuple representing a visit to an urgent location. Visualization tools provided by the authors of dReal do not support drawing traces containing a `NULL` tuple. To enable visualization for all traces generated by the tool, ForFET^{SMT} post-processes traces generated by dReal. It eliminates all `NULL` tuples and re-indexes them to be consistent with the syntax expected by the visualization tool.

3 Tool Evaluation

In this Section, we present selected results on three case studies, i.e. a battery charger, a cruise control, and a buck regulator. We have tested a wide variety of features, capturing state-dependent, time-dependent, sequential-properties and combinations of them. Some of these properties can also be encoded as control specifications, e.g. overshoot or settling time. More details about the models, specifications, features, and analysis results can be found in the tool manual attached. Fig. 3 describes the analysis results obtained from ForFET^{SMT}. Fig. 3a displays the computational time when using reachability analysis and SMT solving. The feature range is computed first using SpaceEx and is then refined using SMT. In most cases, reachability analysis and SMT require similar time to compute the expected feature range. However, in models with a lot of switching like the Buck Regulator, SMT might be more vulnerable to the state-space explosion. For the *settling time* feature, the analysis timed out after 4 hours. However, the additional computation overhead leads to tighter feature ranges. In Fig. 3b, we present the resulting feature ranges after using both SpaceEx and dReach/dReal.

4 Conclusion

In this paper, we have presented the ForFET^{SMT} tool, that is a formal feature evaluation tool for hybrid automata, emphasizing on its architecture and utilities. Features form a promising and practical research direction as they can be used on top of or alongside standard monitoring and hybrid reachability tools to provide quantitative measures about HA behaviors. ForFET^{SMT} makes use of the HASLAC language for writing HA models and is linked to SpaceEx reachability tool and dReal/dReach SMT solver. Using such an SMT solver to compute

features can produce concrete traces for feature corner points and lead to the generation of tighter feature ranges.

Acknowledgement

The authors acknowledge the support of Semiconductor Research Corporation (SRC) through task 2740.001.

References

1. Ain, A., Bruto da Costa, A., Dasgupta, P.: Feature indented assertions for analog and mixed-signal validation. IEEE TCAD PP(99), 1–1 (2016)
2. Alur, R., et al.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138, 3–34 (1995)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (Jan 1996)
4. ARCH, A.: Benchmarks for continuous and hybrid system verification (2015), `http://cps-vo.org/group/ARCH/benchmarks`
5. Bak, S., et al.: HyST: A source transformation and translation tool for hybrid automaton models. In: HSCC. ACM, Seattle, Washington (Apr 2015)
6. Bruto da Costa, A.A., Dasgupta, P.: ForFET: A Formal Feature Evaluation Tool for Hybrid Systems. In: Proc. of ATVA. pp. 437–445 (2017)
7. Bruto da Costa, A.A., Frehse, G., Dasgupta, P.: Formal feature interpretation of hybrid systems. IEEE TCAD 37(11), 2474–2484 (2018)
8. Cotton, S., Frehse, G., Lebeltel, O.: The spaceex modeling language (2010)
9. Dang, T., et al.: Verification of analog and mixed-signal circuits using hybrid system techniques. In: FMCAD, pp. 21–36 (2004)
10. Deshmukh, J.V., et al.: Robust online monitoring of signal temporal logic. Formal Methods in System Design 51(1), 5–30 (2017)
11. Frehse, G., et al.: A toolchain for verifying safety properties of hybrid automata via pattern templates. In: ACC. pp. 2384–2391 (June 2018)
12. Frehse, G., et al.: SpaceEx: Scalable Verification of Hybrid Systems. In: CAV (2011)
13. Gao, S., et al.: dreal: An SMT solver for nonlinear theories over the reals. In: CADE. pp. 208–214 (2013)
14. Kapinski, J., et al.: St-lib: A library for specifying and classifying model behaviors. In: SAE Technical Paper. SAE International (04 2016)
15. Kekatos, N.: Formal Verification of Cyber-Physical Systems in the Industrial Model-Based Design Process. Ph.D. thesis (2018)
16. Konrad, S., Cheng, B.H.C.: Real-time specification patterns. In: ICSE. pp. 372–381. ICSE '05, ACM (2005)
17. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: FORMATS-FTRTFT. pp. 152–166. Springer (2004)
18. Nickovic, D., et al.: AMT 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic. In: TACAS. pp. 303–319 (2018)
19. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE Computer Society (1977)