History-Based Run-Time Requirement Enforcement of Non-Functional Properties on MPSoCs

Khalil Esper
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Abstract

Embedded system applications usually have requirements regarding non-functional properties of their execution like latency or power consumption. Enforcement of such requirements can be implemented by a reactive control loop, where an enforcer determines based on a system response (feedback) how to control the system, e.g., by selecting the number of active cores allocated to a program or by scaling their voltage/frequency mode. It is of a particular interest to design enforcement strategies for which it is possible to provide formal guarantees with respect to requirement violations, especially under a largely varying environmental input (workload) per execution. In this paper, we consider enforcement strategies that are modeled by a finite state machine (FSM) and the environment by a discrete-time Markov chain. Such a formalization enables the formal verification of temporal properties (verification goals) regarding the satisfaction of requirements of a given enforcement strategy.

In this paper, we propose *history-based* enforcement FSMs which compute a reaction not just on the current, but on a fixed history of K previously observed system responses. We then analyze the quality of such enforcement FSMs in terms of the probability of satisfying a given set of verification goals and compare them to enforcement FSMs that react solely on the current system response. As experimental results, we present three use cases while considering requirements on latency and power consumption. The results show that history-based enforcement FSMs outperform enforcement FSMs that only consider the current system response regarding the probability of satisfying a given set of verification goals.

2012 ACM Subject Classification Computer systems organization → Multicore architectures; Theory of computation → Linear logic; Theory of computation → Modal and temporal logics; Hardware → Finite state machines; Computer systems organization → Self-organizing autonomic computing; Theory of computation → Verification by model checking; Mathematics of computing → Probabilistic representations

Keywords and phrases Verification, Runtime Requirement Enforcement, History, Latency

Digital Object Identifier 10.4230/OASIcs.NG-RES.2024.4

Funding This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research-Foundation) – Project Number 146371743 – TRR 89 Invasive Computing.

1 Introduction

Embedded applications usually come with constraints on non-functional properties such as latency, power consumption, temperature, security, etc. A major uncertainty source that affects such properties is the varying workload of the input data[1]. Different run-time

[1] Other uncertainties such as caused by resource sharing can be handled systematically by techniques for isolating application programs dynamically at run-time such as invasive computing [1, 32] and therefore not considered here.

management methods exist for dynamic control of program executions. However, most of them have as disadvantages that they cannot provide formal guarantees regarding their capability to fulfill the given requirements.

Run-time requirement enforcement (RRE) techniques [33] have been proposed to enforce a set of non-functional properties of execution of a given application program within defined bounds. Such techniques dynamically adapt system configurations including, e.g., the voltage/frequency settings and/or the number of active cores in reaction to observed system responses. Based on that, *FSM-based RREs* [10–13, 30] have been proposed for formally specifying and verifying control strategies. Such approaches consider execution properties that can be modeled by *requirements* [31], i.e. expressions on non-functional properties such as permitted corridors on latency, power consumption, etc. Different verification goals can be specified and formally verified, e.g., the probability with which program executions satisfy a given set of requirements.

The FSM-based RRE approaches in [10–13, 30] define and use a binary *requirement response* vector that specifies for each given requirement whether it has been satisfied (1) or not (0) in the current execution. Based on such a system response, then determines the next state, respectively configuration to be applied during the next execution. However, it is a challenge to design enforcement FSMs that satisfy a set of verification goals with maximized probabilities, especially if the considered requirements are conflicting with each other like latency and power consumption. A potential for improvements is to let the enforcer consider not only the current, but also system responses from earlier execution iterations when deciding for the next configuration. In this regard, this paper proposes *history-based enforcement FSMs* that not only consider the current system response, but also a history of previous system responses for reaction.

This paper is organized as follows. Section 2 discusses the related work. In Section 3, we introduce the system model, formally specify history-based enforcement FSMs, and propose three examples of history-based enforcement FSMs that use a history of previous system responses for determining a reaction. Section 4 describes the evaluation of history-based enforcement FSMs for three different use case applications and compares between the proposed history-based enforcement FSMs and enforcement FSMs that do not consider previous responses for reaction. Finally, Section 5 concludes this work.

2 Related Work

Approaches based on heuristics [35], online learning [4, 23–25], or statistical regression [8, 15] are generally not able to provide any formal guarantees regarding the satisfaction or violation of non-functional properties of program executions. Finite state machines (FSMs) have been proposed to formally specify *functional* system properties [5, 14, 29]. Based on the concept of *Run-time Requirement Enforcement (RRE)* [33], FSMs have been proposed in [10–13, 30] for feedback-based enforcement of non-functional properties on MPSoCs.

Such FSM-based RREs utilize a *requirement response* vector that abstracts the system as a function that specifies for each requirement whether it has been fulfilled or violated in the current execution. Based on such a system response, the enforcer reacts by determining the configuration to be applied in the next execution iteration. However, all of the previous approaches only consider the current system response for reaction. In our work, we take into account a time window of K previous responses when deciding for the configuration for the next execution.

In control theory, the principle of *time-delayed feedback* [18, 36] has been proposed to increase the stability of a system. However, this concept has never been applied for controlling software systems. In addition, similar to [6, 16, 26, 27], approaches based on control theory can only give guarantees regarding the control stability or convergence, but not the satisfaction or violation of non-functional properties of program executions.

3 Method

In this section, we first present the considered system model and then formulate and propose three different history-based enforcement FSM strategies that take multiple previous system responses into account for taking a reaction.

3.1 System Model

Embedded systems, especially MPSoCs, often consider the execution of periodic applications, e.g., image, video, or periodic control applications. For each individual program execution, a set of non-functional requirements shall be respected, even under environmental changes, e.g., varying input. This input can be represented for each discrete execution k of an application by an *environment feature vector* $i(k) \in \mathcal{I}$, where $\mathcal{I}$ is called the *environment space* [11]. The program utilizes a number n of cores that can be dynamically changed as well as the their voltage/frequency setting m. Such a setting $\langle n, m \rangle$ is called a configuration c and the set of available configurations a *configuration space* C. FSM-based RREs [11] react based on a feedback from the system-under-control by adapting the configuration $c(k+1)$ for the $(k+1)$th execution accordingly. Figure 1 illustrates the considered system model which is described more closely in the following.

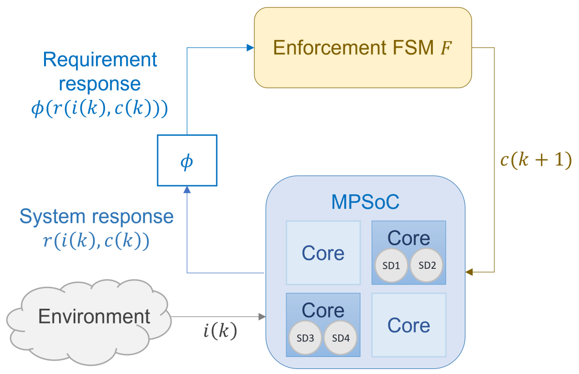


Figure 1 Illustration of a feedback-based RRE. A requirement response vector r is mapped to a binary requirement response vector ϕ that will be used by an enforcement FSM F to decide for the next configuration $c(k+1) \in C$. Reprinted from [11].

Depending on an input $i(k) \in \mathcal{I}$ and a system configuration $c(k) \in C$, let the k-th execution result into a set of H observable non-functional properties, e.g., latency and power consumption in case of $H = 2$ observable properties. The system-under-control can then be described by a *system response function* $r : \mathcal{I} \times C \to \mathbb{R}^H$ [11]. Thus, the *system response* $r(i(k), c(k)) = (o_1(k), \ldots, o_H(k))$ at execution k is a vector of H execution properties of interest, see Figure 1. Now, requirements on these properties, e.g., deadlines, must be fulfilled for each execution, where each property o_h, $h = 1, \ldots, H$ can be formulated using corridors from which the following two propositions φ_h^{LB} and φ_h^{UB} can be derived

$$\varphi_h^{LB}\left(o_h(k)\right) = \left(LB_h \leq o_h(k)\right) \tag{1}$$
$$\varphi_h^{UB}\left(o_h(k)\right) = \left(o_h(k) \leq UB_h\right) \tag{2}$$

where LB_h and UB_h refer to the lower and the upper bound, respectively, on the execution property o_h. The information regarding which proposition is satisfied and which is violated at the k-th execution is represented by a binary vector β named *requirement response*. It is obtained from the system response r using the requirement response function ϕ [11]

$$\beta(k) := \phi\left(o_1(k), \ldots, o_H(k)\right) = \big(\varphi^{LB}(o_1(k)), \varphi^{UB}(o_1(k)), \ldots, \\ \varphi^{LB}(o_H(k)), \varphi^{UB}(o_H(k))\big) \in \{0,1\}^{2H}. \tag{3}$$

This binary requirement response vector $\beta(k)$ constitutes the input to the enforcement FSM F that determines the next configuration $c(k+1) \in C$ to enforce the given non-functional properties for the next execution.

An *enforcement FSM (F)* can be formally modeled by a deterministic finite state machine (Moore machine) which is described by a 6-tuple $(Z, z_0, B, \delta, C, \gamma)$ [11]:

- Z is a finite set of states.
- $z_0 \in Z$ is the initial state.
- B is the input alphabet.
- δ is the transition relation: $\delta \subseteq B \times Z \times Z$ with (β, z, z') representing a transition from z to z' under input β.
- C is the output alphabet, also called configuration space.
- γ is the output function that maps each state to an output (i.e., a configuration): $\gamma : Z \to C$.

Finally, in order to quantitatively compare different enforcement strategies, verification goals can be formulated in temporal logic [7]. The two types of temporal logic are linear temporal logic and branching time logic. Linear temporal logic (LTL) describes events over a single time path in the FSM. Branching time logic such as computation tree logic (CTL) quantifies the possible paths from a given state in the FSM. Different levels of strictness of requirement enforcement can be differentiated, see [34]. Accordingly, different verification goals can be defined. For example, the CTL formula $AG(\varphi_h)$ for strict enforcement indicates that φ holds for every path and at every state on the path. For loose enforcement, $AF(\varphi_h)$ specifies that for every possible path there exists a state at which φ holds, see [11].

Probabilistic verification goals, based on PCTL [2], can also be formulated to specify stochastic verification goals for loose enforcement. We utilize probabilistic verification goals that are based on *steady-state probabilities* in Markov chains as they are helpful for obtaining requirement satisfaction probabilities of long execution runs of an application regardless of the initial state. The operator S is used in PRISM [20] to reason about the steady-state probability of a model [3]. We define the verification goal $S_{=?}[\varphi]$ as the steady-state probability of being in a satisfying state for the requirement φ. Finally, we refer to the set of all considered verification goals by VG.

3.2 History-based Enforcement FSMs

In this work, we propose enforcement strategies that not only react based on the current response vector $\beta(k)$, but additionally on a *history* of system responses $(\beta(k-1), \ldots, \beta(k-K))$ belonging to the previous K execution iterations. Note that the case of $K = 0$ represents the case of enforcement FSMs that are not history-based, i.e., they only react on $\beta(k)$ and do not consider previous system responses for reaction.

In this section, we introduce exemplarily three multi-requirement history-based enforcement FSMs F_1, F_2, F_3 that consider current response $\beta(k)$ and the previous response $\beta(k-1)$ ($K = 1$) to calculate a proper reaction. These FSMs execute on an MPSoC given with $n = 4$ available cores that can operate in $m = 20$ different power modes (voltage/frequency states). Thus, the size of the configuration space C available for enforcement is $|C| = 4 \cdot 20 = 80$. We also assume these configurations to be power-ascending so that the configuration c_j associated with $\langle n_j, m_j \rangle$ has a higher power consumption than that of configuration c_{j-1} where $0 \leq j < |C|$.

Let us consider the latency o_L and the power consumption o_P as properties of execution to be enforced, thus $H = 2$. For simplicity, we only utilize one-sided requirements so that the lower bounds are $LB_{o_L} = 0$ and $LB_{o_P} = 0$. Additionally, let the set of states Z be $|Z| = |C|$ such that the output function is a bijection $\gamma : Z \leftrightarrow C$ of sets Z and C. Thus, there is a one-to-one relation between enforcer states and configurations, such that each enforcer state $z \in Z$ uniquely outputs one configuration $c \in C$. Based on that, each enforcement FSM has as many states as $|Z| = |C| = 80$, thus, $Z = \{z_0, \cdots, z_{79}\}$, the input $\beta \in B = \{0,1\}^H = \{0,1\}^2$ with $\beta = \phi(r'(s,c)) = \phi(o_L, o_P) = ((o_L \leq UB_{o_L}), (o_P \leq UB_{o_P}))$, an assumed initial state $z_0 = 0$. Finally, we assume both the latency requirement $\varphi_L(k)$ and the power requirement $\varphi_P(k)$ are satisfied for executions $k < 0$.

3.2.1 Latency Violation-Oriented History-Based Enforcement FSM

This enforcement FSM decreases the current state by exactly one step in case of a violation of a power requirement in both the current execution ($\overline{\varphi_P(k)}$) and the previous one ($\overline{\varphi_P(k-1)}$) only when the latency requirement in both the current execution ($\varphi_L(k)$) and the previous one ($\varphi_L(k-1)$) is satisfied. It stays in the same configuration for the other cases when the latency requirement is satisfied in both executions ($\varphi_L(k)$) and ($\varphi_L(k-1)$). It increases by one step in case of a violation of an latency requirement in either the current execution ($\overline{\varphi_L(k)}$) or the previous one ($\overline{\varphi_L(k-1)}$). Finally, it increases by two steps in case of a violation of an latency requirement in both the current execution ($\overline{\varphi_L(k)}$) and the previous one ($\overline{\varphi_L(k-1)}$). A corresponding enforcement FSM $F_1 = (Z, z_0, I, \gamma, C, \delta_1)$ has the transition relation δ_1 shown in Table 1.

3.2.2 Power Violation-Oriented History-Based Enforcement FSM

This enforcement FSM increases the current state by exactly one step in case of a violation of a latency requirement in both the current execution ($\overline{\varphi_L(k)}$) and the previous one ($\overline{\varphi_L(k-1)}$) only when the power requirement in both the current execution ($\varphi_P(k)$) and the previous one ($\varphi_P(k-1)$) is satisfied. It stays in the same configuration for the other cases when the power requirement is satisfied in both executions ($\varphi_P(k)$) and ($\varphi_P(k-1)$). It decreases by one step in case of a violation of a power requirement in either the current execution ($\overline{\varphi_P(k)}$) or the previous one ($\overline{\varphi_P(k-1)}$). Finally, it decreases by two steps in case of a violation of a power requirement in both the current execution ($\overline{\varphi_P(k)}$) and the previous one ($\overline{\varphi_P(k-1)}$). A corresponding enforcement FSM $F_2 = (Z, z_0, I, \gamma, C, \delta_2)$ has the transition relation δ_2 shown in Table 2.

3.2.3 Multi-Requirement History-Based Enforcement FSM

This enforcement FSM does not favor any requirement when transitioning between enforcer states. A corresponding enforcement FSM $F_3 = (Z, z_0, I, \gamma, C, \delta_3)$ has the transition relation δ_3 shown in Table 3.

Table 1 The transition relation δ_1 of the latency-oriented history-based enforcement FSM F_1.

$z(k)$	$\beta(k-1)$		$\beta(k)$		$z(k+1)$
z_j	true	true	true	true	z_j
z_j	true	false	true	true	z_j
z_j	true	true	true	false	z_j
z_j	true	false	true	false	z_{j-1}
z_j	false	true	true	true	z_{j+1}
z_j	false	false	true	true	z_{j+1}
z_j	false	true	true	false	z_{j+1}
z_j	false	false	true	false	z_{j+1}
z_j	true	true	false	true	z_{j+1}
z_j	true	false	false	true	z_{j+1}
z_j	true	true	false	false	z_{j+1}
z_j	true	false	false	false	z_{j+1}
z_j	false	true	false	true	z_{j+2}
z_j	false	false	false	true	z_{j+2}
z_j	false	true	false	false	z_{j+2}
z_j	false	false	false	false	z_{j+2}

Table 2 The transition relation δ_2 of the power-oriented history-based enforcement FSM F_2.

$z(k)$	$\beta(k-1)$		$\beta(k)$		$z(k+1)$
z_j	true	true	true	true	z_j
z_j	true	false	true	true	z_{j-1}
z_j	true	true	true	false	z_{j-1}
z_j	true	false	true	false	z_{j-2}
z_j	false	true	true	true	z_j
z_j	false	false	true	true	z_{j-1}
z_j	false	true	true	false	z_{j-1}
z_j	false	false	true	false	z_{j-2}
z_j	true	true	false	true	z_j
z_j	true	false	false	true	z_{j-1}
z_j	true	true	false	false	z_{j-1}
z_j	true	false	false	false	z_{j-2}
z_j	false	true	false	true	z_{j+1}
z_j	false	false	false	true	z_j
z_j	false	true	false	false	z_{j-1}
z_j	false	false	false	false	z_{j-2}

4 Experimental Results

In this section, we introduce three applications for evaluating the proposed enforcement FSMs in Section 3.2. For comparison, we also perform a design space exploration (DSE) method from [13] to generate optimized enforcement FSMs for a given set of verification goals VG for each application, where these enforcement FSMs do not consider previous system responses for reaction. To perform the DSE, the NSGA-II [9] multi-objective evolutionary

Table 3 The transition relation δ_3 of the multi-requirement history-based enforcement FSM F_3.

$z(k)$	$\beta(k-1)$		$\beta(k)$		$z(k+1)$
z_j	true	true	true	true	z_j
z_j	true	false	true	true	z_{j-1}
z_j	true	true	true	false	z_{j-1}
z_j	true	false	true	false	z_{j-2}
z_j	false	true	true	true	z_{j+1}
z_j	false	false	true	true	z_j
z_j	false	true	true	false	z_j
z_j	false	false	true	false	z_{j-1}
z_j	true	true	false	true	z_{j+1}
z_j	true	false	false	true	z_j
z_j	true	true	false	false	z_j
z_j	true	false	false	false	z_{j-1}
z_j	false	true	false	true	z_{j+2}
z_j	false	false	false	true	z_{j+1}
z_j	false	true	false	false	z_{j+1}
z_j	false	false	false	false	z_j

algorithm provided by the optimization framework Opt4J [22] is used. Each run of the DSE features 100 iterations with a population size of 20 enforcement FSMs with a crossover probability of 0.9 and a mutation probability of 0.01. Each experiment was repeated three times to compensate for the randomness of the exploration.

4.1 Applications

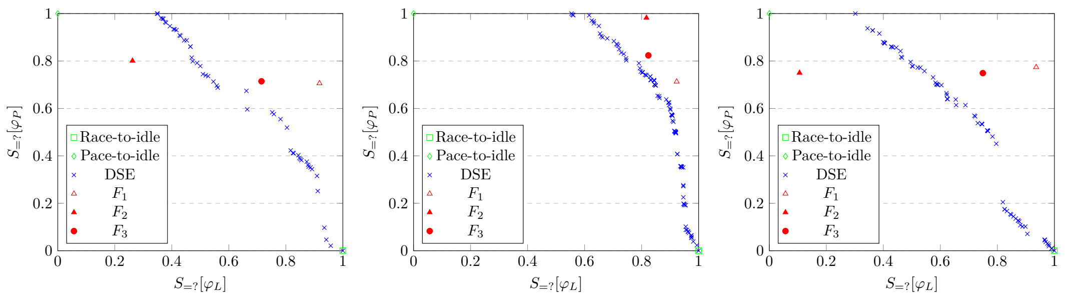
We consider three applications for evaluation. Each application is modeled by a graph of actors, where each actor processes an input $i(k)$ in each iteration k. The applications execute on a tiled many-core system that consists of a set of processing cores, peripherals like memories, and a network adapter, which are interconnected via a tile-local bus system. For this matter, a simulation framework called InvadeSIM [28], a many-core simulator for parallel applications is used.

4.1.1 Object Detection Application

An image processing application that detects a given object in each image frame by applying a scale-invariant feature transform (SIFT) matching algorithm. We use a driving car image sequence R of the KITTI-360 dataset [21] with $|R| = 100$ frames, a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 65$ W, an power lower bound $LB_{o_P} = 0$ mJ and an upper bound $UB_{o_P} = 5$ W.

4.1.2 String Search Application

This application stems from the ParMiBench benchmark suite [17] that searches in a given input text k with $i(k)$ lines for a given pattern. For this use case, we create a trace of $|R| = 100$ randomly generated texts, each having $i(k)$ lines. We use the bounds $LB_{o_L} = 0$ ms, $UB_{o_L} = 15$ ms, $LB_{o_P} = 0$ W and $UB_{o_P} = 1.5$ W.



(a) Object detection.

(b) String search.

(c) SHA.

Figure 2 Verification results for the proposed history-based enforcement FSMs F_1, F_2, F_3 for a history of $K = 1$, compared to DSE-optimized enforcement FSMs that do not consider the response history [13], and the heuristic techniques race-to-idle and pace-to-idle [19].

4.1.3 Secure Hash Application

Another application from the ParMiBench benchmark suite [17]. This security application computes the hash for the input k that consists of $i(k)$ messages. For this use case, we create a trace of $|R| = 100$ randomly generated inputs k, with $LB_{o_L} = 0$ ms, $UB_{o_L} = 9$ ms, $LB_{o_P} = 0$ W, and $UB_{o_P} = 3$ W.

4.2 Results

Figure 2 shows the verification results of the proposed history-based enforcers F_1, F_2, F_3 that consider the previous response $\beta(k-1)$ together with enforcement FSMs that are obtained from the DSE method in [13] and do not consider any previous response for reaction, as well as race-to-idle (i.e., running with the highest configuration c_{79}) and pace-to-idle (i.e., running with the slowest configuration c_0) [19].

We notice in Figure 2 that the history-based enforcement FSMs F_1 and F_2 are not dominated by any other enforcement FSM in all of the three applications. Also, the history-based enforcement FSM F_3 is not dominated in the case of string search application. This shows that reacting based on a history of previous system responses can enhance the probability of satisfying the considered verification goals. The reason is the larger design space of transition possibilities in the enforcement FSM.

Table 4 shows the average verification time, number of states, and transitions for 10 randomly-generated enforcement FSMs with different history options. Enforcement FSMs with $K = 0$ indicates that they only react on the current system response $\beta(k)$. History-based enforcement FSMs with $K = 1$ implies that they include the previous system response $\beta(k-1)$ for reaction as well as $\beta(k)$. Finally, history-based enforcement FSMs with $K = 2$ consider the system responses $\beta(k-2), \beta(k-1)$, and $\beta(k)$ for reaction. We notice that reacting based on previous system responses leads to a substantial increase in verification times. This is explained by the increase of number of states and transitions of the resulting enforcement FSM. We also notice that this increase is proportional to the length of the time window K of previously considered responses.

5 Conclusion

In this paper, we proposed to integrate a history of previous system responses into the design of enforcement strategies. The evaluation shows that such history-based enforcement FSMs have the potential to have higher probabilities of satisfying a given set of verification goals

Table 4 Average verification time, number of states, and transitions for 10 randomly-generated enforcement FSMs with different history options.

	$K = 0$			$K = 1$			$K = 2$		
Application	time (ms)	states	transitions	time (ms)	states	transitions	time (ms)	states	transitions
Object detection	133.0	262.7	710.7	357.0	1,193.3	3,176.3	6,460.1	4,064.3	10,786.6
String Search	416.1	1,299.4	5,166.7	24,786.9	10,354.9	41,301.9	840,498.1	41,477.8	166,058.5
SHA	179.3	453.9	1,513.5	2,894.3	2,800.1	9,303.8	63,172.1	9,726.3	32,336.3

than enforcement FSMs that do not consider any system response history. This offers system designers with a trade-off between complexity and performance. In the future, we aim to automatically optimize history-based enforcement FSMs for a given set of verification goals.

References

1 Nidhi Anantharajaiah, Tamim Asfour, Michael Bader, Lars Bauer, Jürgen Becker, Simon Bischof, Marcel Brand, Hans-Joachim Bungartz, Christian Eichler, Khalil Esper, Joachim Falk, Nael Fasfous, Felix Freiling, Andreas Fried, Michael Gerndt, Michael Glaß, Jeferson Gonzalez, Frank Hannig, Christian Heidorn, Jörg Henkel, Andreas Herkersdorf, Benedict Herzog, Jophin John, Timo Hönig, Felix Hundhausen, Heba Khdr, Tobias Langer, Oliver Lenke, Fabian Lesniak, Alexander Lindermayr, Alexandra Listl, Sebastian Maier, Nicole Megow, Marcel Mettler, Daniel Müller-Gritschneder, Hassan Nassar, Fabian Paus, Alexander Pöppl, Behnaz Pourmohseni, Jonas Rabenstein, Phillip Raffeck, Martin Rapp, Santiago Narváez Rivas, Mark Sagi, Franziska Schirrmacher, Ulf Schlichtmann, Florian Schmaus, Wolfgang Schröder-Preikschat, Tobias Schwarzer, Mohammed Bakr Sikal, Bertrand Simon, Gregor Snelting, Jan Spieck, Akshay Srivatsa, Walter Stechele, Jürgen Teich, Isaías A. Comprés Ureña, Ingrid Verbauwhede, Dominik Walter, Thomas Wild, Stefan Wildermann, Mario Wille, Michael Witterauf, and Li Zhang. *Invasive Computing.* FAU University Press, 2022.

2 Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the Logical Characterisation of Performability Properties. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 780–792. Springer, 2000.

3 Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings*, volume 1664 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 1999.

4 Dwaipayan Biswas, Vibishna Balagopal, Rishad A. Shafik, Bashir M. Al-Hashimi, and Geoff V. Merrett. Machine learning for run-time energy optimisation in many-core systems. In David Atienza and Giorgio Di Natale, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pages 1588–1592. IEEE, 2017.

5 Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime Enforcement for Reactive Systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *Lecture Notes in Computer Science*, pages 533–548. Springer, 2015.

6 Sophie Cerf, Raphaël Bleuse, Valentin Reis, Swann Perarnau, and Eric Rutten. Sustaining performance while reducing energy consumption: a control theory approach. In *Euro-Par 2021: Parallel Processing: 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1–3, 2021, Proceedings 27*, pages 334–349. Springer, 2021.

7 Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

8 Junio Cezar Ribeiro Da Silva, Lorena Leão, Vinicius Petrucci, Abdoulaye Gamatié, and Fernando Magno Quintão Pereira. Mapping computations in heterogeneous multicore systems with statistical regression on program inputs. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(6):1–35, 2021.

9 Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002.

10 Khalil Esper, Jan Spieck, Pierre-Louis Sixdenier, Stefan Wildermann, and Jürgen Teich. RAVEN: reinforcement learning for generating verifiable run-time requirement enforcers for MPSoCs. In *Fourth Workshop on Next Generation Real-Time Embedded Systems, NG-RES 2023, January 18, 2023, Toulouse, France*, volume 108 of *OASIcs*, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

11 Khalil Esper, Stefan Wildermann, and Jürgen Teich. Enforcement FSMs: specification and verification of non-functional properties of program executions on MPSoCs. In *MEMOCODE '21: 19th ACM-IEEE International Conference on Formal Methods and Models for System Design, Virtual Event, China, November 20–22, 2021*, pages 21–31. ACM, 2021.

12 Khalil Esper, Stefan Wildermann, and Jürgen Teich. Multi-requirement enforcement of non-functional properties on MPSoCs using enforcement FSMs – A case study. In *Third Workshop on Next Generation Real-Time Embedded Systems, NG-RES@HiPEAC 2022, June 22, 2022, Budapest, Hungary*, volume 98 of *OASIcs*, pages 2:1–2:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

13 Khalil Stefan Wildermann Esper and Jürgen Teich. Automatic synthesis of FSMs for enforcing non-functional requirements on MPSoCs using multi-objective evolutionary algorithms. *ACM Trans. Des. Autom. Electron. Syst.*, August 2023.

14 Ylièes Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

15 Xinwei Fang, Sinem Getir Yaman, Radu Calinescu, Julie Wilson, and Colin Paterson. Predicting nonfunctional requirement violations in autonomous systems. *ACM Transactions on Autonomous and Adaptive Systems*, 2023.

16 Connor Imes, David HK Kim, Martina Maggio, and Henry Hoffmann. POET: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86. IEEE Computer Society, 2015.

17 Syed Muhammad Zeeshan Iqbal, Yuchen Liang, and Håkan Grahn. Parmibench – An open-source benchmark for embedded multiprocessor systems. *IEEE Comput. Archit. Lett.*, 9(2):45–48, 2010.

18 Wolfram Just, Thomas Bernard, Matthias Ostheimer, Ekkehard Reibold, and Hartmut Benner. Mechanism of time-delayed feedback control. *Physical Review Letters*, 78(2):203, 1997.

19 David H. K. Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications, CPSNA 2015, Kowloon, Hong Kong, China, August 19-21, 2015*, pages 78–85. IEEE Computer Society, 2015.

20 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification – 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

21 Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *CoRR*, abs/2109.13410, 2021.

22 Martin Lukasiewycz, Michael Glaß, Felix Reimann, and Jürgen Teich. Opt4j: a modular framework for meta-heuristic optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1723–1730, 2011.

23 Sumit K. Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Ümit Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Trans. Design Autom. Electr. Syst.*, 25(3):28:1–28:26, 2020.

24 Sumit K. Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Ümit Y. Ogras. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Trans. Very Large Scale Integr. Syst.*, 27(12):2842–2854, 2019.

25 Maxime Mirka, Gilles Sassatelli, and Abdoulaye Gamatié. Online learning for dynamic control of openmp workloads. In *9th International Conference on Modern Circuits and Systems Technologies, MOCAST 2020, Bremen, Germany, September 7-9, 2020*, pages 1–6. IEEE, 2020.

26 Anway Mukherjee and Thidapat Chantem. Energy management of applications with varying resource usage on smartphones. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37(11):2416–2427, 2018.

27 Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 – June 07, 2013*, pages 174:1–174:9. ACM, 2013.

28 Sascha Roloff, Frank Hannig, and Jürgen Teich. *Modeling and Simulation of Invasive Applications and Architectures.* Computer Architecture and Design Methodologies. Springer, 2019.

29 Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

30 Jan Spieck, Pierre-Louis Sixdenier, Khalil Esper, Stefan Wildermann, and Jürgen Teich. Hybrid genetic reinforcement learning for generating run-time requirement enforcers. In *2023 21st ACM-IEEE International Symposium on Formal Methods and Models for System Design (MEMOCODE)*, pages 23–35, 2023.

31 Jürgen Teich, Michael Glaß, Sascha Roloff, Wolfgang Schröder-Preikschat, Gregor Snelting, Andreas Weichslgartner, and Stefan Wildermann. Language and Compilation of Parallel Programs for *-Predictable MPSoC Execution Using Invasive Computing. In *10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSOC 2016, Lyon, France, September 21-23, 2016*, pages 313–320. IEEE Computer Society, 2016.

32 Jürgen Teich, Jörg Henkel, Andreas Herkersdorf, Doris Schmitt-Landsiedel, Wolfgang Schröder-Preikschat, and Gregor Snelting. Invasive computing: An overview. In *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*, pages 241–268. Springer, 2011.

33 Jürgen Teich, Pouya Mahmoody, Behnaz Pourmohseni, Sascha Roloff, Wolfgang Schröder-Preikschat, and Stefan Wildermann. Run-Time Enforcement of Non-functional Program Properties on MPSoCs. In *A Journey of Embedded and Cyber-Physical Systems*, pages 125–149. Springer, 2021.

34 Jürgen Teich, Behnaz Pourmohseni, Oliver Keszöcze, Jan Spieck, and Stefan Wildermann. Run-Time Enforcement of Non-Functional Application Requirements in Heterogeneous Many-Core Systems. In *25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, January 13-16, 2020*, pages 629–636. IEEE, 2020.

35 Xiaohang Wang, Amit Kumar Singh, Bing Li, Yang Yang, Hong Li, and Terrence S. T. Mak. Bubble budgeting: Throughput optimization for dynamic workloads by exploiting dark cores in many core systems. *IEEE Trans. Computers*, 67(2):178–192, 2018. `doi:10.1109/TC.2017.2735967`.

36 Dong Yue and Qing-Long Han. Delayed feedback control of uncertain systems with time-varying input delay. *Automatica*, 41(2):233–240, 2005.