Multi-Requirement Enforcement of Non-Functional Properties on MPSoCs Using Enforcement FSMs – A Case Study

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

Stefan Wildermann
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

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

Abstract

Embedded system applications usually have to meet real-time, energy or safety requirements on programs typically concurrently executed on a given MPSoC target platform. Enforcing such properties, e.g., by adapting the number of processors allocated to a program or by scaling the voltage/frequency mode of involved processors, is a difficult problem to solve, especially with a typically large varying environmental input (workload) per execution. In a previous work [4], we formalized the related enforcement problem using (a) finite state machines to model enforcement strategies, (b) discrete-time Markov chains to model the uncertain environment determining the system's workload, and (c) the system response that defines the feedback for the reactive enforcer. In this paper, we apply that approach to specify and verify multi-requirement enforcement strategies and assess a case study for enforcing two independent requirements at the same time, i.e., latency and energy consumption. We evaluate and compare different enforcement strategies using probabilistic verification for the use case of an object detection application.

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 Runtime Requirement Enforcement, Verification, Finite State Machine, Markov Chain, Energy Consumption, Probabilistic Model Cheking, PCTL, MPSoC

Digital Object Identifier 10.4230/OASIcs.NG-RES.2022.2

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

1 Introduction

Many, particularly embedded system applications come along with different requirements that should be met during their execution on many-core systems. One example are temperature constraints because of a limited power budget. Causes of variations include interference with other applications, i.e., due to shared resources. In addition, the workload induced by the environment input data[1] can vary significantly and with high uncertainty.

[1] This paper takes care of the uncertainty of input from the environment that is typically not under the control of a system, thus the influence is exogenous. The uncertainty of execution state on an MPSoC platform is typically caused by sharing of resources on an MPSoC platform, thus endogenous. This problem can be treated systematically by techniques for isolating application programs dynamically at run-time such as invasive computing [18] and is therefore not treated here.

As a solution, *run-time requirement enforcement (RRE)* techniques such as [19] have emerged. Such techniques adjust a set of configurations like voltage/frequency setting or the degree of parallelism in reaction to observed changes in the system state or input workload. Different techniques have been proposed to implement run-time managers for dynamic adaptation of program execution [5, 7, 11, 12]. Many of them have drawbacks that either no formal guarantees can be given regarding their effectiveness of holding the specified requirements or they make simplifying assumptions regarding the controller or the many-core system under control. To overcome such disadvantages, we use *finite state machines (FSMs)* to formally specify enforcement strategies, as they are expressive in specification and computation, which helps in formulating complex strategies. Furthermore, they do not make any restricting or simplifying assumptions regarding the system-under-control, and finally they can be analyzed using formal verification techniques [4].

In this respect, we proposed a methodology [4] to verify the satisfaction of requirements on non-functional properties of program executions on MPSoCs which are controlled by FSM-based RREs. A requirement is often specified by a corridor of allowed or desired values of a non-functional property of program execution, e.g., a latency or power consumption corridor. In order to quantitatively assess a RRE strategy, we specify and formally verify important verification goals, e.g., whether a RRE is able to either strictly or at least loosely stay within the bounds of one or a set of requirements. Furthermore, how many subsequent program executions could violate a given set of requirements before eventually returning into specified requirement corridors. After introducing a *discrete-time Markov chain (DTMC)* for modeling the variation of environmental input, we model the *requirement response* of an MPSoC when executing an application program as a function that specifies for each combination of environment state and enforcer output whether and which requirements are fulfilled and which are violated.

Based on the concatenation of formal models, we can define stochastic verification problems using *Probabilistic Computation Tree Logic (PCTL)* [1], a probabilistic variant of CTL, and apply stochastic model checking, i.e., PRISM [9] to analyze and compare different enforcer strategies quantitatively.

Unlike other approaches that enforce only one requirement at a time [16, 2, 14, 15, 13], our method [4] can be used to enforce more than one requirement – although this was not shown there. In this work, we therefore use the methodology in [4] to analyze enforcers for multiple independent requirements at a time. As an example, we propose multi-requirement enforcement strategies to satisfy more than one requirement (e.g., latency and energy requirements) at a time. We then assess a case study to evaluate and compare between different strategies using probabilistic formal verification based on the PRISM model checker.

The remainder of this paper is structured as follows. Section 2 introduces the formal models for specifying the enforcement FSM and the requirement response. Section 3 describes the evaluation of a proposed set of uni- and multi-requirement enforcement FSMs based on an image processing case study and discusses the verification results. Finally, in Section 4, we conclude this work.

2 Run-time Requirement Enforcement (RRE)

The enforcement of requirements should be achieved even under a variation of environmental inputs. Such an input can be described for each discrete periodic execution k of a program by an environment feature vector $x(k) \in \mathcal{X}$, where $\mathcal{X}$ denotes the *environment space*. *Run-time requirement enforcement (RRE)* techniques such as [19, 3] can be used to enforce the

satisfaction of a set of requirements during each execution even in the presence of input variation. In the following, we term such an assignment of resources and their parametrization to execute a given program a configuration c and the set of possible system configurations to execute a program *configuration space* C. We consider *Feedback-based RRE* techniques such as [7] that react to a violation or satisfaction of a non-functional requirement φ based on feedback from the *system-under-control* and adapting the configuration $c(k+1)$ for the next execution $(k+1)$ accordingly [4]. Figure 1 illustrates the proposed model described in the following.

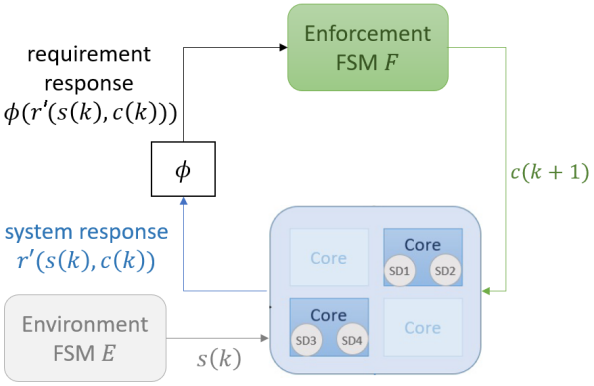

Figure 1 Schematic illustration of feedback-based RRE [4].

2.1 Enforcement FSM F

Feedback-based RREs can be modeled by finite state machines. Following [4], a respective enforcement FSM is defined as follows:

▶ **Definition 1.** *An* enforcement FSM (F) *is a deterministic finite state machine (Moore machine) that can be described by a 6-tuple* $(Z, z_0, I, \delta, C, \gamma)$ [4]*:*

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

An enforcement FSM is *uni-requirement* when it enforces one requirement $(h = 1)$, and it is called *multi-requirement* when it enforces more than one requirement $(h > 1)$.

2.2 Requirement Enforcement

A MPSoC platform, or *system-under-control* is abstracted by a single function called *system response function* $r : \mathcal{X} \times C \to \mathbb{R}^h$ [4]. The *system response* at execution k is considered as a vector of h relevant execution qualities $r(x(k), c(k)) = (o_1(k), \ldots, o_h(k))$, e.g., corresponding to observed latency, power consumption, etc., during the kth execution.

In [17], requirements on non-functional execution qualities o_j, $j = 1, \ldots, h$ are specified using intervals, e.g., resulting in two propositions φ_j^{LB} and φ_j^{UB}

$$\varphi_j(o_j(k)) = \varphi_j^{LB}(o_j(k)) \wedge \varphi_j^{UB}(o_j(k)) = (LB_{o_j} \leq o_j(k)) \wedge (o_j(k) \leq UB_{o_j}) \tag{1}$$

where LB_{o_j} and UB_{o_j} denote a lower, respectively upper bound on the execution quality o_j.

The system response $r(x(k), c(k)) = (o_1(k), \ldots, o_h(k))$ is finally mapped to a *requirement response* using a requirement response function

$$\begin{aligned}\phi\left(o_1(k), \ldots, o_h(k)\right) := \big(&\varphi^{LB}(o_1(k)), \varphi^{UB}(o_1(k)), \ldots, \\ &\varphi^{LB}(o_h(k)), \varphi^{UB}(o_h(k))\big) \in \{0,1\}^{2h}\end{aligned} \tag{2}$$

This binary vector serves as the input to the enforcement FSM F, and thus the input alphabet is $I \subseteq \{0,1\}^{2h}$. Based on the requirement response, F computes as a reaction the next configuration $c(k+1) \in C$ to enforce the desired non-functional properties for the next execution.

2.3 Environment FSM E

In order to apply verification techniques on a proper enforcement of requirements, we need a formal model of the environment that influences the system-under-control. Thus, the environment is modeled using a discrete-time Markov Chain called *environment FSM* [4]. However, the number of possible input values can be very large, possibly leading to a state explosion. In [4], we proposed to partition the environment space $\mathcal{X}$ into equivalence classes or partitions p leading to the same requirement response ϕ as a potential solution.

▶ **Definition 2.** *An* environment finite state machine E *is a discrete-time Markov chain (DTMC) defined by a 3-tuple* (S, a, Δ) [4]*:*

- S *is the finite set of states. Each* $s \in S$ *is assigned exactly one partition* $p \in P$ *of the environment space* $\mathcal{X}$*.*
- $a : S \to [0,1]$ *is a function that assigns each state* $s \in S$ *its initial state probability* $a(s)$*.*
- $\Delta \in [0,1]^{|S|\times|S|}$ *is a transition probability matrix.*

3 Multi-Requirement Enforcement Case Study

In the following, we present a simple image processing application as shown in Figure 2. The job of the object detection algorithm is to detect a given object in each image frame by applying a SIFT feature matching algorithm. Subsequently, we present different RRE variants and verify a number of PCTL verification goals related to h requirements to be enforced using probabilistic model checking.

3.1 Object Detection Application

The object detection application, shown in Figure 2, is an image processing application that conducts a pipelined processing of a periodic input image stream. The goal of this application is to detect a given object in each image frame by applying a scale-invariant feature transform (SIFT) matching algorithm [10]. The application consists of an actor chain. Each actor processes one input image frame k at a time. The image source (IS) actor reads in the input images periodically at a defined rate, then follows the gray-scale conversion (GS) actor, and after that the edge detection (ED) and the corner detection (CD) actors to determine

respectively the edges and corners in an image. After that, the SIFT orientation (SO) actor applies invariance to image rotation. The four SIFT description actors SD_1 to SD_4 extract the features in an image. They can be executed in parallel on $n = 4$ cores, after partitioning the number of features x of a given image evenly into each actor.

For the following experiments, let each of the periodic executions of each SD actor have a latency requirement $\varphi_L = \varphi_L^{LB} \wedge \varphi_L^{UB} = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ which is typical in real-time systems and an energy consumption requirement $\varphi_{En} = \varphi_{En}^{LB} \wedge \varphi_{En}^{UB} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$, for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ. Intuitively, $\varphi_L^{LB} = (0 \text{ ms} \leq o_L)$ and $\varphi_{En}^{LB} = (0 \text{ mJ} \leq o_{En})$ are always satisfied.

For the enforcement of such requirements, the execution power mode m (voltage/frequency) of the SD actors' cores through Dynamic Voltage and Frequency Scaling (DVFS) is used, and we assume, a maximum of $n = 4$ cores can be activated in each of $m = 20$ different power modes. However, during the execution of an image, we assume all cores run in the same power mode m, thus resulting in a configuration $\langle n, m \rangle$. Upon each execution, the output of each SD actor is then sent to the SIFT matching (SM) actor to detect common features between the given reference object and the features in the current input image. Then, the RAN-SAC (RS) actor calculates the transformation between both images based on the matched features. The image is finally delivered by an image destination (ID) actor.

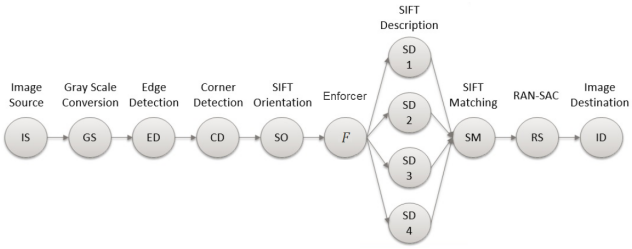

Figure 2 Object detection algorithm implemented as a graph of actors for pipelined processing of streams of images.

3.2 Specifying Enforcement FSMs

In the following, we introduce and compare five enforcement FSMs exemplarily, each having as many states as configurations $|C|$, thus $Z = \{0, \cdots, 31\}$, the input $i \in I = \{0,1\}^{2h} = \{0,1\}^4$ with $i = \phi(r'(s,c)) = \phi(o_L, o_{En}) = ((LB_{o_L} \leq o_L), (o_L \leq UB_{o_L}), (LB_{o_{En}} \leq o_{En}), (o_{En} \leq UB_{o_{En}}))$, an assumed initial state $z_0 = 17$ and the configuration space C of cardinality $|C| = 32$.

- *1-step latency-requirement enforcement FSM* F_1: [3] proposes a technique called 1-step enforcement that decreases, resp. increases the current state, respectively configuration by exactly one step in case of a satisfaction (φ_L), resp. violation ($\overline{\varphi_L}$) of a latency requirement to be enforced. A corresponding enforcement FSM $F_1 = (Z, z_0, I, \gamma, C, \delta_1)$ is shown in Figure 3. It has the following transition relation δ_1:

$$z(k+1) = \begin{cases} 0 & \varphi_L \wedge (z(k) = 0) \\ z(k) - 1 & \varphi_L \wedge (z(k) \neq 0) \\ z(k) + 1 & \overline{\varphi_L} \wedge (z(k) \neq 31) \\ 31 & \overline{\varphi_L} \wedge (z(k) = 31) \end{cases} \tag{3}$$

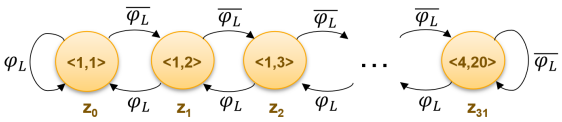

Figure 3 1-step latency-requirement enforcement FSM F_1 that only utilizes φ_L. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *1-step energy-requirement enforcement FSM* F_2: Decreases, resp. increases the current state, respectively configuration reflecting the next lower, resp. higher power by exactly one step in case of a satisfaction (φ_{En}), resp. violation ($\overline{\varphi_{En}}$) of a energy requirement to be enforced. A corresponding enforcement FSM $F_2 = (Z, z_0, I, \gamma, C, \delta_2)$ is shown in Figure 4. It has the following transition relation δ_2:

$$z(k+1) = \begin{cases} 0 & \overline{\varphi_{En}} \wedge (z(k) = 0) \\ z(k) - 1 & \overline{\varphi_{En}} \wedge (z(k) \neq 0) \\ z(k) + 1 & \varphi_{En} \wedge (z(k) \neq 31) \\ 31 & \varphi_{En} \wedge (z(k) = 31) \end{cases} \tag{4}$$

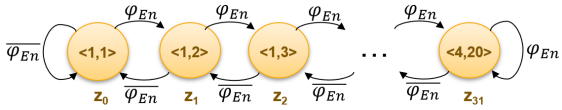

Figure 4 1-step energy-requirement enforcement FSM F_2 that only utilizes φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *1-step multi-requirement enforcement FSM* F_3: Stays in the current state, respectively configuration if both requirements are satisfied or none of the requirements are satisfied, and decreases the current state by exactly one step if only φ_{En} is violated and increases if only φ_L is violated. A corresponding enforcement FSM $F_3 = (Z, z_0, I, \gamma, C, \delta_3)$ is shown in Figure 5. It has the following transition relation δ_3:

$$z(k+1) = \begin{cases} 0 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) = 0) \\ z(k) - 1 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) \neq 0) \\ z(k) & ((\varphi_L \wedge \varphi_{En}) \vee (\overline{\varphi_L} \wedge \overline{\varphi_{En}})) \wedge (z(k) \neq 0) \\ z(k) + 1 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) \neq 31) \\ 31 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) = 31) \end{cases} \tag{5}$$

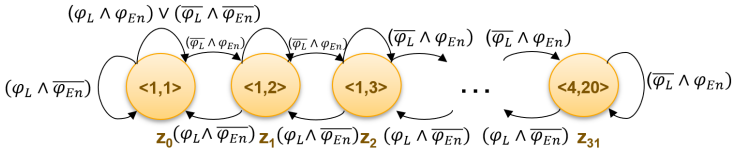

Figure 5 1-step multi-requirement enforcement FSM F_3 for enforcing two single-bound requirements φ_L and φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *Latency-oriented multi-requirement enforcement FSM* F_4: Decreases the current state by one step when only φ_{En} is violated, increases the current state by two steps when φ_L is violated, and stays in the same state otherwise. A corresponding enforcement FSM $F_4 = (Z, z_0, I, \gamma, C, \delta_4)$ is shown in Figure 6. It has the transition relation δ_5:

$$z(k+1) = \begin{cases} 0 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) = 0) \\ z(k) - 1 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) \neq 0) \\ z(k) & (\varphi_L \wedge \varphi_{En}) \wedge (z(k) \neq 0) \\ z(k) + 2 & \overline{\varphi_L} \wedge (z(k) < 30) \\ 31 & \overline{\varphi_L} \wedge (z(k) \geq 30) \end{cases} \tag{6}$$

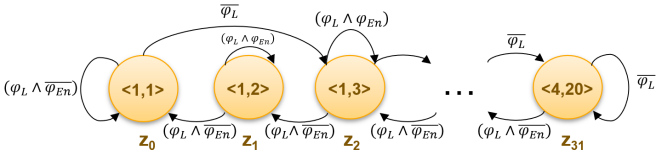

Figure 6 Latency-oriented multi-requirement enforcement FSM F_4 for enforcing two single-bound requirements φ_L and φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *Energy-oriented multi-requirement enforcement FSM* F_5: Decreases the current state by two steps when φ_{En} is violated, increases the current state by one step when only φ_L is violated, and stays in the same state otherwise. A corresponding enforcement FSM $F_5 = (Z, z_0, I, \gamma, C, \delta_5)$ is shown in Figure 7. It has the following transition relation δ_4:

$$z(k+1) = \begin{cases} 0 & \overline{\varphi_{En}} \wedge (z(k) \leq 1) \\ z(k) - 2 & \overline{\varphi_{En}} \wedge (z(k) > 1) \\ z(k) & (\varphi_L \wedge \varphi_{En}) \wedge (z(k) \neq 0) \\ z(k) + 1 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) \neq 31) \\ 31 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) = 31) \end{cases} \tag{7}$$

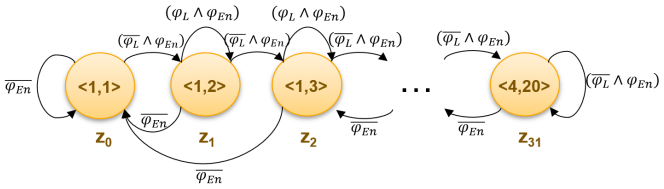

Figure 7 Energy-oriented multi-requirement enforcement FSM F_5 for enforcing two single-bound requirements φ_L and φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

3.3 Verification Goals Specification

As we do not want to reason about or verify the satisfaction of verification goals during single runs or single traces of input stimuli, but depending on a DTMC model of the typically uncertain environment (i.e., E), we can use PCTL [1], a probabilistic variant of CTL, to specify stochastic verification goals. In the following, $\mathcal{P}_{=?}[\varphi_j]$ denotes the probability of satisfying a proposition φ_j.

For verification, we will use PRISM [9], a probabilistic model checker, to perform verification of a number of interesting verification goals on enforcers.

Finally, LTL formulas have a *bounded* variant in PCTL [6], which adds an upper bound λ on the number of successive steps or iterations in our model. In our previous work [4], we proposed and formulated a set of interesting verification goals using temporal logic. In this work, we use one of them to compare between the proposed enforcement FSMs. That being: $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi)]$ which denotes the probability of a requirement φ to never hold in any of λ consecutive executions. We will use this verification goal to verify the proposed enforcement strategies for the latency requirement $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L)]$, the energy consumption requirement $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_{En})]$, and finally both requirements together $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L \vee \neg\varphi_{En})]$.

3.4 Deriving Environment FSMs E

We first partition the environment space $\mathcal{X}$ into a set of partitions P, by computing the system response $o = (o_L, o_{En}) = r(x, c)$ for all different inputs $x \in \mathcal{X}$ for each $c \in C$, and then deriving the partitions p of inputs $x \in \mathcal{X}$ that result in the same requirement response $\phi(r(x, c))$ for each $c \in C$.

For the above example application, the authors in [20] proposed to perform a design space exploration (DSE) to compute the maximum number of features $x_{max}(UB_{o_L}, c)$ that can be processed within a given deadline UB_{o_L} in configuration c, and a single state enforcer that proactively (rather than reactively) chooses upon arrival of the kth frame always the configuration $c \in C$ consuming the minimal amount of energy for that input. A similar DSE is conducted for the energy consumption o_{En} for an energy consumption upper bound $UB_{o_{En}}$.

Based on this information, we can compute the requirement response $\phi(r(x, c))$ for each input $x \in \mathcal{X}$ and configuration $c \in C$ for each execution k as follows:

$$\phi(r(x,c)) = \begin{cases} (1,1,1,1) & (\varphi_L^{LB}, \varphi_L^{UB}, \varphi_{En}^{LB}, \varphi_{En}^{UB}) \\ (1,1,1,0) & (\varphi_L^{LB}, \varphi_L^{UB}, \varphi_{En}^{LB}, \overline{\varphi_{En}^{UB}}) \\ (1,0,1,1) & (\varphi_L^{LB}, \overline{\varphi_L^{UB}}, \varphi_{En}^{LB}, \varphi_{En}^{UB}) \\ (1,0,1,0) & (\varphi_L^{LB}, \overline{\varphi_L^{UB}}, \varphi_{En}^{LB}, \overline{\varphi_{En}^{UB}}) \end{cases} \tag{8}$$

Based on this partitioning, and using a frame sequence R with $|R| = 1{,}000$ frames, we follow the procedure explained in [4] to obtain the environment FSM E_1, shown in Figure 8, which is specified after computing the partitions based on a latency requirement $\varphi_L = \varphi_L^{LB} \wedge \varphi_L^{UB} = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ and an energy consumption requirement $\varphi_{En} = \varphi_{En}^{LB} \wedge \varphi_{En}^{UB} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$, for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ. Intuitively, $\varphi_L^{LB} = (0 \text{ ms} \leq o_L)$ and $\varphi_{En}^{LB} = (0 \text{ mJ} \leq o_{En})$ are always satisfied.

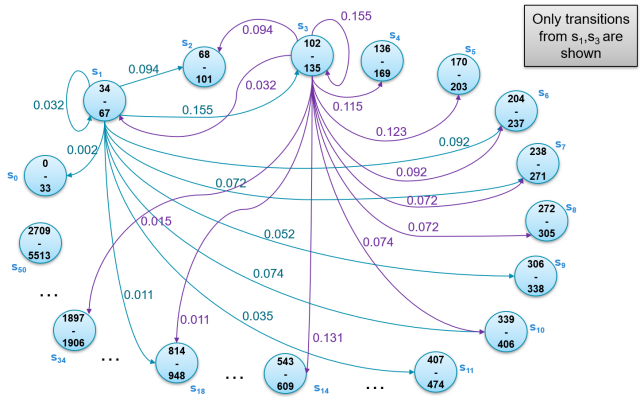

Figure 8 Resulting environment FSM E_1 and for an image processing algorithm for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ generated from a trace R of $|R| = 1000$ frames. The highest encountered number of features x (workload to be processed) per frame in the trace R is 5,513.

3.5 Verification Results

We specified the enforcement and environment FSMs for verification by the PRISM model checker using the PRISM modeling language [8].

- $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L)]$ is the probability of φ_L to never hold in any of λ consecutive executions. Figure 9 plots this probability for increasing values of λ for all five introduced enforcement FSMs. We notice that F_2 and F_5 have the highest probabilities of violating the latency requirement among all other enforcement FSMs. This is plausible, since F_2 only utilizes the energy requirement response as an input, and the multi-requirement enforcement FSM F_5 is energy-oriented.
  We also notice that F_4 is superior regarding satisfying the satisfaction of latency requirement, since it is a latency-oriented multi-requirement enforcement FSM that jumps two steps forward when φ_L is violated. Finally, even F_1 utilizes only the latency requirement response to transition between states, it has a higher probability of violating the latency requirement than F_3 and F_4, because it goes backwards one step when φ_L is satisfied, unlike the multi-enforcement FSMs F_3 and F_4 which stay in the same state when φ_L is satisfied.
- $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_{En})]$ is the probability of φ_{En} to never hold in any of λ consecutive executions. Figure 10 plots this probability for increasing values of λ for all five introduced enforcement FSMs. We notice that F_5 has the lowest probability for violating the energy requirement, because it is a multi-requirement FSM that is oriented for energy saving where it goes

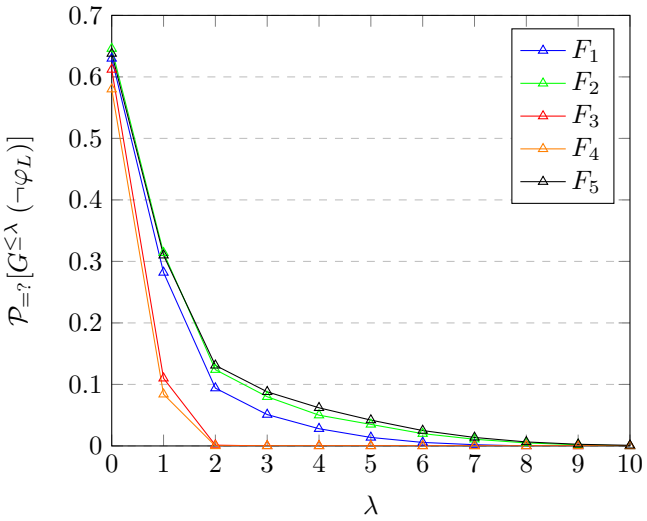

Figure 9 Probability that φ_L never holds in any of λ consecutive executions ($\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L)]$) where $\varphi_L = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms.

backward two steps when φ_{En} is violated. Next comes F_2, which only utilizes the energy requirement response as an input. We also notice that other enforcement FSMs violate the energy requirement with higher probabilities than F_2, F_5, since they either do not consider the energy requirement response like F_1 or they do not prioritize it over the latency requirement φ_L like F_3 and F_4. Finally, we notice that the latency-oriented multi-requirement FSM F_4 has the highest probability of violating the energy requirement. Even though F_4 considers the energy requirement response, F_1 still has a lower probability in violating φ_{En}. This shows the need for systematic methods to generate enforcement FSMs that satisfy given requirements.

- Finally, we analyze the probability of φ_L or φ_{En} to never hold in any of λ consecutive executions by computing $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L \vee \neg\varphi_{En})]$. Figure 11 plots this probability for increasing values of λ for all introduced enforcement FSMs. As can be observed, F_3 exhibits the lowest probability of violating any requirement. This is because it has the third lowest probability of violating the energy requirement φ_{En} and the second lowest probability of violating the latency requirement φ_L, see Figure 9 and Figure 10. F_2 and F_5 show the next best behavior after F_4, because they have the lowest probability of violating the energy requirement φ_{En}, and at the same time showing a comparable probability concerning the latency requirement, see Figure 9 and Figure 10. Finally, although F_4 is the best regarding the satisfaction of the latency requirement, it is the worst in satisfying both requirements together, as it has a very low probability of satisfying the energy requirement φ_{En}, which outweighs the latency requirement satisfaction, see Figure 10.

4 Conclusion and Future Work

In this paper, we proposed multi-requirements enforcement strategies for enforcing multiple non-functional requirements at a time, like latency and energy consumption. We formulated several verification goals that ask for the probability of violating the latency, energy or both requirements for a consecutive number of steps. Doing so allows to formally verify and

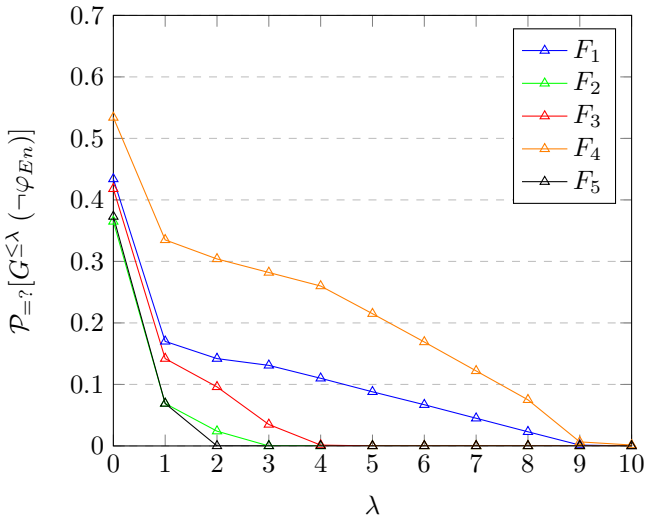

Figure 10 Probability that φ_{En} never holds in any of λ consecutive executions ($\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_{En})]$) where $\varphi_{En} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$ for an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ.

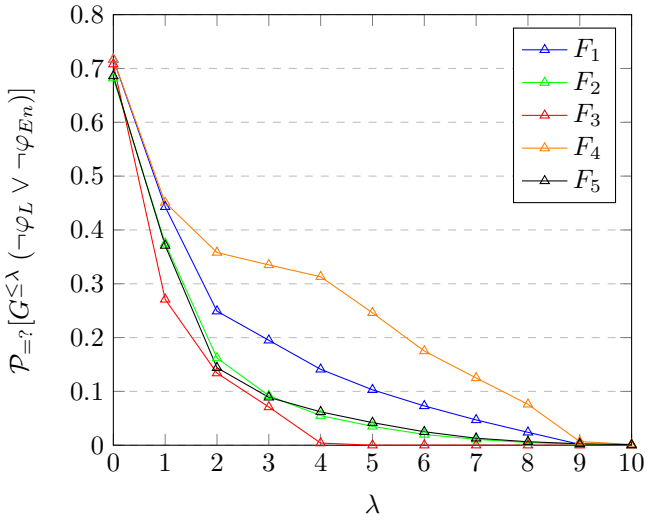

Figure 11 Probability that φ_L or φ_{En} never holds in any of λ consecutive executions ($\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L \vee \neg\varphi_{En})]$) where $\varphi_L = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ and $\varphi_{En} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$, for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ.

quantitatively compare different FSMs for requirement enforcement. We used PRISM for such probabilistic verification problems. Our evaluation shows that it is very difficult to understand which FSMs are superior with respect to individual and which for combined enforcement of multiple requirements. In the future, we would like to work on techniques for the automatic generation of enforcement FSMs with multiple probabilistic verification goals (a multi-objective optimization problem) for the enforcement of one or multiple requirements at a time.

References

1 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.

2 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.

3 Khalil Esper, Stefan Wildermann, and Jürgen Teich. A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems. In *Second Workshop on Next Generation Real-Time Embedded Systems, NG-RES@HiPEAC 2021, January 20, 2021, Budapest, Hungary*, volume 87 of *OASIcs*, pages 1:1–1:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

4 Khalil Esper, Stefan Wildermann, and Jürgen Teich. Enforcement FSMs – Specification and Verification of Non-Functional Properties of Program Executions on MPSoCs. In *19th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE'21)*, 2021.

5 Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering*, pages 299–310, 2014.

6 Hans Hansson and Bengt Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

7 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.

8 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Quantitative analysis with the probabilistic model checker PRISM. *Electron. Notes Theor. Comput. Sci.*, 153(2):5–31, 2006.

9 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.

10 David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

11 Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Automated control of multiple software goals using multiple actuators. In Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman, editors, *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 373–384. ACM, 2017.

12 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.

13 Srinivas Pinisetty, Partha S. Roop, Vidula Sawant, and Gerardo Schneider. Security of pacemakers using runtime verification. In *16th ACM/IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2018, Beijing, China, October 15-18, 2018*, pages 51–61. IEEE, 2018.

14 Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard von Hanxleden. Runtime enforcement of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 16(5s):178:1–178:25, 2017.

15 Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Stavros Tripakis, and Reinhard von Hanxleden. Runtime enforcement of reactive systems using synchronous enforcers. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10–14, 2017*, pages 80–89. ACM, 2017.

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

17 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.

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

19 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.

20 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.