

# Automatic Synthesis of Stabilizing Controllers for Discrete Time Linear Hybrid Systems

Leonardo Picchiami<sup>1</sup>

<sup>1</sup>Computer Science Dept., Sapienza University of Rome, via Salaria 113, 00198, Italy

## Abstract

Many Cyber-Physical Systems are either mission or safety critical, thus the controllers for such systems must be provably correct. To this aim, in the last decades many methodologies have been developed which are able to automatically generate correct-by-construction controllers for Cyber-Physical Systems.

In this paper, we focus on one of such methodologies, implemented in the QKS tool, which is able to explicitly take into account the specification of the discretization of the state space due to analog-to-digital conversion. Controllers output by QKS are able to drive any controllable state inside the system goal. However, such controllers may allow a goal state to go outside the goal region, in order to be able to bring it back into the goal region later. QKS does not provide any means to detect such behavior.

Here we propose to modify QKS in order to provide an additional feature, which only outputs *stabilizing* controllers, *i.e.*, controllers which never allow goal states to go outside the goal region. If this is not possible for the input system, an empty controller is returned.

We prove the feasibility of our approach on well-known control theory case studies, namely the multi input buck DC-DC converter and the inverted pendulum. Our experimental results show that we are able to output stabilizing controllers using limited additional resources with respect to the original QKS.

## Keywords

Automatic Controller Synthesis, Stabilizing Controllers, CPS

## 1. Introduction

The development of correct-by-construction software controllers for Cyber-Physical Systems (CPSs) is a strong requirement in many safety/mission-critical contexts such as, for example, avionics [1, 2, 3, 4], smart grids [5, 6, 7, 8, 9], automotive [10, 11, 12, 13] or biological systems [14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. In fact, many CPSs are Software-Based Control Systems (SBCSs), which are typically subdivided into two subsystems: *plant* and *controller*. The plant is a physical system (*e.g.*, electric devices), whereas the controller is a software component running on a microcontroller. The controller, in a *closed-loop system*, interacts with the plant to satisfy formal specifications, *i.e.*, functional requirements (*liveness* and *safety*) needed for correctness of the system. In our setting, we want the controller to drive the system towards a *goal region* (*liveness*), so that only safe states are traversed (*safety*). To this aim, the controller implements a

---


IPS-RCRA 2021: 9th Italian Workshop on Planning and Scheduling and 28th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion

✉ picchiami@di.uniroma1.it (L. Picchiami)

ORCID iD 0000-0001-5477-6419 (L. Picchiami)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

control law  $ctrlLaw$ , which, by looking at the current state, decides which action must be taken to bring the system closer to the goal, while retaining safety. Such control law is guaranteed to correctly work when the current state is within its control region  $CtrlReg$ .

The usual control-loop workflow is as follows: the plant sensors measure an input  $x$ . SBCS compute the Analog-to-Digital (AD) conversion getting a quantized value  $\hat{x}$ . The controller checks if  $\hat{x}$  belongs to the control region. If it is not in  $ctrlRegion$ , then the procedure of *Fault Detection, Isolation and Recovery* is triggered. Otherwise, the system carries out the *Digital-to-Analog* (DA) conversion and sends the command to the plant actuators.

Designing a controller is an hard task [24]. In recent years, many articles have been published aiming at *automatically synthesizing* controllers, often basing on ideas borrowed from logics [25, 26, 27, 28, 29, 30] and model checking techniques [31, 32, 33, 34, 35, 36, 37, 38, 39, 40]. This has led to the introduction of many tools to automatically generate controllers, such as, e.g., PESSOA [41] and Synthia [42].

In this paper, we focus on the *Quantized feedback Kontrol Synthesizer* (QKS) tool [43]. QKS is a tool to automatically synthesize control software for an input system (plant) modeled as a *Discrete-Time Linear Hybrid System* (DTLHS), i.e., a system described by continuous as well as discrete variables. QKS also takes as input the liveness and safety specification, i.e., the goal region to be reached by the system, as well as how many bits  $b$  are available for the AD and DA conversions. As an output, QKS returns the C code implementation of the control software, which is guaranteed to drive any system state in the control region to the goal in a finite number of steps, also taking into account the AD and DA conversion.

The controller output by QKS is such that, once the system has reached the goal, it may also exit from the goal itself. If this happens, the controller is again able to bring the system back to the goal again. This may be undesired in some applications, where instead, it is required that the system *stabilizes* in the goal, so preventing the system from leaving it. A system is able to continuously meet properties that ensure the correctness and prevent errors as long as it remains in the goal. So, in safety/mission-critical contexts (e.g., medical devices, drugs or banking systems), we need controllers that implement the  $l$ -stability. Otherwise, we might obtain systems that can be dangerous, for example, for a human or a company.

Note that, in this paper, we present a new notion of stabilizing controllers. In the literature, such a notion has often been used in a different unrelated setting, e.g., bounding fluctuations of controller trajectories [44, 45]. Here, instead, we focus on controllers which “traps” a system on the goal, if possible. To overcome the undesired behavior outlined above, in this paper, we present an extension to QKS, which only outputs controllers which stabilize the system up  $l \geq 0$  steps, being  $l$  a further input for QKS.

Other extensions of QKS have been presented in the literature with different objectives than the ones we have here. Examples are in [46] (performing an on-the-fly controller generation), [47] (introducing a parallel algorithm for the controller synthesis) and [48] (aiming at reducing the size of control software implementation). To the best of our knowledge, this is the first time that an algorithm for stabilizing controllers for QKS is presented.

The paper outline is as follows. We recall the mathematical notations which formalizes the synthesis of software controllers (Section 2). Then, we introduce and formalize the concept of stabilizing controllers (Section 3). Under this framework, we present a new algorithm to generate stabilizing controllers (Section 4). Finally, we report an experimental comparison

between the existing algorithm of controller synthesis provided by QKS and the synthesis algorithm of stabilizing controllers (Section 5). To this end, our case studies are the well-known Multi-input Buck DC-DC converter and the Inverted Pendulum.

## 2. Formal Framework

To make this paper self-contained, we briefly summarize all concepts needed for the formalization of stabilizing controllers. For a complete discussion, see [43].

### 2.1. Linear Predicate

Let  $[n] = \{1, \dots, n\}$  be the set of positive integers less than  $n$  and let  $X = [x_1, \dots, x_n]$  be a sequence of distinct variables. Every variable  $x \in X$  is defined on either a finite (e.g., boolean variables) or infinite (e.g., discrete or continuous variables) domain, denoted by  $\mathcal{D}_x$ . Boolean variables are denoted as  $x^b$ , continuous variables as  $x^r$ , and discrete variables as  $x^d$ . Along the same lines,  $X^b$  (resp.,  $X^d, X^r$ ) represents the sequence of the boolean (resp, discrete and real) variables among the ones in  $X$ . The domain of a sequence of variables  $X$  is denoted by  $\mathcal{D}_X = \prod_{x \in X} \mathcal{D}_x$ .

A *linear expression*  $L(X)$  is a linear combination of variables in  $X$ , i.e.,  $\sum_{x \in X} a_x x$  where  $a_x$  are rational constants. A *linear constraint* is an expression of the form  $L(X) \bowtie b$  where  $b$  is a rational constant and  $\bowtie \in \{\leq, \geq, =\}$ . A *predicate* is any logical combination of OR and AND operations among linear constraints. A *conjunctive predicate* is predicate with AND operations only.

A *valuation* over  $X$  is a function  $v : X \rightarrow \mathcal{D}_X$ , assigning to each variable  $x$  a value in the corresponding domain  $\mathcal{D}_x$ . Given a valuation  $v$ , we write  $X^*$  for the sequence  $[v(x_1), \dots, v(x_n)] \in \mathcal{D}_X$ , and we call it valuation as well. A *satisfying assignment* to predicate  $P(X)$  is a valuation of  $X^*$  such that  $P(X)$  holds. If  $P(X^*)$  holds for some  $X^*$ , then  $P$  is *feasible*.

Given a constraint  $C(X)$  and a fresh variable  $y$ , the *guarded constraint*  $y \rightarrow C(X)$  (if  $y$  then  $C(X)$ ) denotes the predicate  $(y = 0) \vee C(X)$ . By negating the guard variable, denoted by  $(\bar{y} \rightarrow C(X))$ , we get the predicate  $(y = 1) \vee C(X)$  (if not  $y$  the  $C(X)$ ). A *guarded predicate* is a conjunction of either constraints or guarded constraints. If all variables in  $X$  are bounded (which implies that a linear expression  $L(X)$  has a finite supremum), then any guarded constraint may be translated in a linear constraint as follows:  $y \rightarrow L(X) \leq b$  is equivalent to  $(\text{sup}(L(X)) - b)y + L(X) \leq \text{sup}(L(X))$ , while  $\bar{y} \rightarrow L(X) \leq b$  is equivalent to  $(b - \text{sup}(L(X)))y + L(X) \leq b$ . By recalling that  $L(X) \geq b$  is equivalent to  $-L(X) \leq -b$  and  $L(X) = b$  is equivalent to  $L(X) \leq b \wedge L(X) \geq b$ , all guarded constraint may be translated to linear predicates.

### 2.2. Labeled Transition Systems

A *Labeled Transition System* (LTS) is a tuple  $\mathcal{S} = (S, A, T)$  where  $S$  is a (possibly infinite) set of states,  $A$  is a (possibly infinite) set of actions and,  $T : S \times A \times S \rightarrow \mathbb{B}$  is the transition relation of  $\mathcal{S}$ .

Let  $s \in S$  and  $a \in A$ ,  $\text{Adm}(\mathcal{S}, s) = \{a \in A \mid \exists s' : T(s, a, s')\}$  is the set of *admissible actions* starting from a state  $s$ , while  $\text{Img}(\mathcal{S}, s, a) = \{s' \in S : T(s, a, s')\}$  is the set of next states of  $s$  through the action  $a$  (non-deterministic action). A *transition* is a triple  $(s, a, s') \in S \times A \times S$  such that  $T(s, a, s')$  holds. If  $s = s'$ , then the transition  $(s, a, s)$  is a self-loop.

A *path* for an LTS  $\mathcal{S}$  is a sequence of states-actions  $\pi = s_0 a_0 s_1 a_1 \dots$  where  $\forall t \geq 0$   $T(s_t, a_t, s_{t+1})$  holds. In addition,  $|\pi|$  is the *length* of a given path that represents the number of actions within  $\pi$ .  $\pi^{(S)}(t)$  denotes the  $(t + 1)$ -th state within  $\pi$  and,  $\pi^{(A)}(t)$  denotes the  $(t + 1)$ -th action over  $\pi$ .

### 2.3. Discrete Time Hybrid System

A *Discrete Time Hybrid System* (DTHS)  $\mathcal{H} = (X, U, Y, N)$  is a tuple defined as follows:

- $X = X^r \cup X^d$  is a finite sequence of real as well as discrete variables (possibly including boolean variables). It models the current state. Along the same lines,  $X'$  denotes the next state.
- $U = U^r \cup U^d$  is a finite sequence of input variables.
- $Y = Y^r \cup Y^d$  is a finite sequence of auxiliary variables typically used to model local variables.
- $N(X, U, Y, X')$  is the predicate that defines the system transition relation.

An LTS can model the dynamics of a DTLHS as follows. Given a DTLHS  $\mathcal{H} = (X, U, Y, N)$ , the *LTS of  $\mathcal{H}$*  is defined as  $LTS(\mathcal{H}) = (\mathcal{D}_X, \mathcal{D}_U, \tilde{N})$ , where  $\tilde{N} : \mathcal{D}_x \times \mathcal{D}_U \times \mathcal{D}_x \rightarrow \mathbb{B}$  is a function such that  $\tilde{N}(x, u, x) \equiv \exists y \in \mathcal{D}_y N(x, u, y, x')$ , *i.e.*, it represents the transition relation for the dynamics of the system. In such a way, a state for  $\mathcal{H}$  is also a state for  $LTS(\mathcal{H})$ , and a path for  $\mathcal{H}$  is also a path for  $LTS(\mathcal{H})$ .

### 2.4. LTS control problem

A *controller  $K$*  is a solution for a control problem of a given DTLHS  $\mathcal{H}$ . It restricts the set of enabled behaviors of  $\mathcal{H}$  by selecting only those actions that bring the system to a goal region infinitely often. In the following, we recall the main formalization for controllers and control problems.

#### 2.4.1. Definition of controller

Given a set of states  $S$  and a set of actions  $A$ , a *controller* is a function  $K : S \times A \rightarrow \mathbb{B}$  such that,  $\forall s \in S$  and  $\forall a \in A$ , if  $K(s, a)$  holds, then  $\exists s' \in S$  such that  $T(s, a, s')$  holds. If  $K(s, a)$  holds, we say that action  $a$  is *enabled* in state  $s$  by  $K$ . The set of states for which the at least one action is enabled by a given controller  $K$  is defined as  $\text{dom}(K) = \{s \in S \mid \exists a K(s, a)\}$ .

Given a controller  $K$  on  $S$  and  $A$ , the *closed-loop system* for an LTS  $\mathcal{S} = (S, A, T)$  is the LTS  $\mathcal{S}^{(K)} = (S, A, T^{(K)})$  where  $T^{(K)}(s, a, s') = T(s, a, s') \wedge K(s, a)$ . In addition, a controller  $K$  is also a *control law* if,  $\forall s \in S$ ,  $K(s, a)$  holds at most for one action.

Given an LTS  $\mathcal{S}$ , a path  $\pi$  is a *fullpath* if either it is infinite or its last state  $(\pi^{(S)}(|\pi|))$  does not have any successor state, (*i.e.*,  $\text{Adm}(\mathcal{S}, \pi^{(S)}(|\pi|)) = \emptyset$ ). The set of fullpaths of  $\mathcal{S}$  starting from a state  $s$  (*i.e.*,  $\pi^{(S)}(0) = s$ ) with an action  $a$  (*i.e.*,  $\pi^{(A)}(0) = a$ ) is denoted by  $\text{Path}(\mathcal{S}, s, a)$ .

Given a fullpath  $\pi$  and a region  $G \subseteq S$ ,  $j(\mathcal{S}, \pi, G)$  denotes the *shortest distance* to reach  $G$  through  $\pi$ , i.e.,  $j(\mathcal{S}, \pi, G) = \min\{n \mid n > 0 \wedge \pi^{(S)}(n) \in G\}$  if  $\pi(i) \in G$  for some  $i$ , otherwise  $j(\mathcal{S}, \pi, G) = +\infty$ . Furthermore, let  $J(\mathcal{S}, G, s, a) = \sup\{j(\mathcal{S}, \pi, G) \mid \pi \in \text{Path}(\mathcal{S}, s, a)\}$  be the shortest distance to  $G$  starting with state  $s$  and action  $a$ . This allows us to define  $J^*(\mathcal{S}, G, s) = \sup\{J(\mathcal{S}, G, s, a) \mid a \in \text{Adm}(\mathcal{S}, s) \text{ for some } a \in A\}$  as the *worst case distance* from  $G$ .

#### 2.4.2. Definition of control problem

An *LTS control problem* is a triple  $\mathcal{P} = (\mathcal{S}, I, G)$ , where  $\mathcal{S} = (S, A, T)$  is an LTS and  $I, G \subseteq S$ . A (*strong*) *solution* for  $\mathcal{P}$  is a controller  $K$  for  $\mathcal{S}$  such that  $I \subseteq \text{dom}(K)$  and  $\forall s \in \text{dom}(K)$ ,  $J^*(\mathcal{S}^{(K)}, G, s)$  is finite. An *optimal solution* for a control problem  $\mathcal{P}$  is a controller  $K^*$  such that  $\forall K$   $J^*(\mathcal{S}^{(K^*)}, G, s) \leq J^*(\mathcal{S}^{(K)}, G, s)$ . Along the same lines, the *most general optimal (m.g.o.) solution* is the solution  $\bar{K}$  such that, for all optimal solutions  $K^*$  to  $\mathcal{P}$ ,  $\forall s \in S$  and  $\forall a \in A$ ,  $K(a, s) \rightarrow \bar{K}(a, s)$  holds, i.e., if  $K(s, a)$  holds, then  $\bar{K}(s, a)$  holds. The m.g.o. solution is unique.

### 2.5. Control Problem for DTLHS

Since an LTS models the dynamics of a DTLHS, a *control problem for a DTLHS* is formally reduced to a control problem for an LTS. Thus, a control problem for a DTLHS  $\mathcal{H} = (X, U, Y, N)$  is denoted as  $(\text{LTS}(\mathcal{H}), I, G)$  for some  $I, G \subseteq \mathcal{D}_X$ .

In order to manage AD and DA conversions, we define a *quantization function* for real interval  $[a, b]$  as a non-decreasing function  $\gamma : [a, b] \rightarrow \mathbb{Z}$ , that maps real values inside a continuous real interval in integer values.

Let a DTLHS  $\mathcal{H} = (X, U, Y, N)$  and let  $W = X \cup U$ , a quantization  $\mathcal{Q}$  is a pair  $(A, \Gamma)$  where:

- $A$  is a predicate over  $W$  that bounds real variables in  $W$ , i.e.,  $A = \bigwedge_{w \in W} A_w = \bigwedge_{w \in W} a_w \leq w \leq b_w$ .  $A_w$  defines the *admissible region* for variable  $w$ . For any  $V \in W$ ,  $A_V = \prod_{v \in V} A_v$ .
- $\Gamma = \{\gamma_w \mid w \in W \wedge \gamma_w \text{ is a quantization function for } A_w\}$ , i.e.,  $\Gamma$  is a set of quantization functions for all real variables in  $\mathcal{H}$ . For  $W = [w_1, \dots, w_k]$  and  $v = [v_1, \dots, v_k]$ , we will write  $\Gamma(v)$  for the tuple  $[\gamma_{w_1}(v_1), \dots, \gamma_{w_k}(v_k)]$ .

A *quantized feedback control solution* for a DTLHS control problem  $\mathcal{P} = (\mathcal{H}, I, G)$ , given a quantization  $\mathcal{Q} = (A, \Gamma)$  for  $\mathcal{H} = (X, U, Y, N)$ , is a  $K$  solution to  $\mathcal{P}$  s.t.:

- if  $(x, u) \notin A_X \times A_U$ , then  $K(x, u) = 0$ ;
- otherwise,  $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ , being  $\hat{K} : \Gamma(A_X) \times \Gamma(A_U) \rightarrow \mathbb{B}$ , i.e.,  $K$  works by only looking at the (integer) values coming after the AD conversion.

### 2.6. Automatically Generating Quantized Feedback Solutions for DTLHS Quantized Control Problems

In the following, we focus on QKS (Quantized Kontrol Synthesizer, [43]), which takes as input a quantization control problem  $\mathcal{P} = (\mathcal{H}, I, G)$  for a DTLHS  $\mathcal{H} = (X, U, Y, N)$  with quantization

$\mathcal{Q} = (A, \Gamma)$  and output a correct-by-construction quantized feedback control solution to  $\mathcal{P}$ . To this aim, QKS goes through the procedure shown in Algorithm 1. Namely, in step 1, initial region  $I$  and goal region  $G$  are quantized. In step 2, a minimal control abstraction is computed as an LTS  $\hat{\mathcal{H}}(\Gamma(A_X), \Gamma(A_U), \hat{N})$ . In a nutshell, in the minimal control abstraction, each abstract transition in  $\hat{N}$  stems from some concrete transition in  $\mathcal{H}$  and viceversa, i.e.,  $\hat{N}(\hat{s}, \hat{a}, \hat{s}')$  if and only if there exists  $y \in Y$  s.t.  $N(s, a, s')$  holds. Some special cases (self loops and actions going outside the admissible region  $A$ ) are treated separately [43]. In step 3, the output quantized controller  $\hat{K}$  is computed with its domain  $\hat{D}$  and a boolean value  $b$  which is true iff  $\hat{I} \subseteq \text{dom}(\hat{K})$ , i.e., if  $\hat{K}$  is able to control all states in the initial region. Note that, if  $b$  is false, QKS provides a necessary condition (not detailed here) for controller existence. The quantized controller synthesis for DTLHSs is actually an undecidable problem [49].

---

**Algorithm 1** QFC Synthesis Algorithm

---

**Input:** DTLHS control problem  $(\mathcal{H}, I, G)$ , quantization  $\mathcal{Q} = (A, \Gamma)$

**function:**  $qCtrSynt(\mathcal{H}, \mathcal{Q}, I, G)$

- 1:  $\hat{I} \leftarrow \Gamma(I), \hat{G} \leftarrow \Gamma(G)$
  - 2:  $\hat{\mathcal{M}} \leftarrow \text{minCtrAbs}(\mathcal{H}, \mathcal{Q})$
  - 3:  $(b, \hat{D}, \hat{K}) \leftarrow \text{strongCtr}(\hat{\mathcal{M}}, \hat{I}, \hat{G})$  **return**  $(b, \hat{D}, \hat{K})$
- 

Algorithm 2 details the computation performed in step 3 of Algorithm 1. Namely, function  $\text{strongCtr}$  starts with an empty controller (step 1). Then, a loop is performed in which, at the  $i$ -th iteration, the states which are at worst case distance  $i$  from the goal for some action  $a$  ( $F$  in step 3) are added to  $K$ , provided that no other actions already existed for those states (with distance  $j < i$ , step 4). All computations are performed representing sets as OBDDs (Ordered Binary Decision Diagrams) [50, 51].

---

**Algorithm 2** Most General Optimal Controller Synthesis

---

**Input:** An LTS control problem  $(\mathcal{S}, I, G), \mathcal{S} = (S, A, T)$ .

**function:**  $\text{strongCtr}(\mathcal{S}, I, G)$

- 1:  $K(s, a) \leftarrow 0, D(s) \leftarrow G(s), \tilde{D}(s) \leftarrow 0$
  - 2: **while**  $D(s) \neq \tilde{D}(s)$  **do:**
  - 3:    $F(s, a) \leftarrow \exists s' T(s, a, s') \wedge \forall s' [T(s, a, s') \Rightarrow D(s')]$
  - 4:    $K(s, a) \leftarrow K(s, a) \vee (F(s, a) \wedge \nexists a K(s, a))$
  - 5:    $\tilde{D}(s) \leftarrow D(s), D(s) \leftarrow D(s) \vee \exists a K(s, a)$
  - 6: **return**  $\langle \forall s [I(s) \Rightarrow \exists a K(s, a)], \exists a K(s, a), K(s, a) \rangle$
- 

### 3. Stabilizing Controllers

The quantized controllers defined in Section 2 are designed so that to drive all controlled states inside the goal infinitely often. That is, for a DTLHS control problem  $\mathcal{P} = (\mathcal{H}, I, G)$ , if  $s \in G$ , then a controller  $K$  can possibly enable an action  $a$  leading to a state  $s' \notin G$ , i.e.,

$K(s, a) \wedge \exists s' [\tilde{N}(s, a, s') \wedge s' \notin G]$ . However,  $K$  will eventually bring all descendants of  $s'$  back into  $G$ , i.e., if  $\mathcal{S} = LTS(\mathcal{H})$ ,  $\forall a \in A$ .  $K(s', a) \rightarrow \forall \pi \in Path(\mathcal{S}^{(K)}, s', a)$ .  $\exists n \pi^S(n) \in G$ .

In some safety- or mission-critical applications, this could not be enough, as it is desired that, once the goal is reached, the controller does not let any state escape from it, i.e.,  $\forall s \in G \cap dom(K)$ , if  $K(s, a)$  then  $\forall s'. N(s, a, s') \rightarrow s' \in G$ . If a controller succeeds in doing this, then it is said to be a *stabilizing controller*.

It is easy to see that, for a given quantized DTLHS control problem, if a stabilizing controller  $K$  exists, then QKS returns  $K$ . In fact, function *strongCtr* in Algorithm 2 always returns the action with the lowest (worst case) distance from the goal, thus if there exists an action  $a$  which directly maintains a goal state  $g \in G$  inside the goal,  $a$  is enabled in  $g$  by  $K$ . If another action  $b$  is enabled in  $g$  by  $K$ , it must not have the same distance as  $a$ , thus it is stabilizing as well.

Unfortunately, the procedure used by QKS does not detect if the output controller is actually stabilizing or not. In this paper, we present an automatic procedure which modifies function *strongCtr* so that the controlled states are all stabilized in the goal for at least  $l \geq 0$  steps, where  $l$  is an additional input to both *strongCtr* and *qCtrSynt* (Algorithm 1). That is, the output  $K_l$  must be an  $l$ -stabilizing controller such that:

1.  $K_l$  is a strong controller for the problem  $\mathcal{P}$ .
2.  $\forall s \in dom(K_l) \forall a \in Adm(\mathcal{S}, s)$   $K_l(s, a)$  holds if and only if  $\forall i \geq 0, \pi \in Path(\mathcal{S}^{(K_l)}, s, a)$ .  $\pi^{(S)}(i) \in G \rightarrow [\forall j = 1, \dots, l - 1. \pi^{(S)}(j + i) \in G]$ .

## 4. Synthesis of Stabilizing Controllers

In order to compute the  $l$ -stabilizing controller for a quantized DTLHS control problem  $\mathcal{P} = (\mathcal{H}, I, G)$  with quantization  $\mathcal{Q}$ , we modified function *qCtrSynt* of Algorithm 1 so that: i) it takes an integer  $l \geq 0$  as an additional input; ii) in step 3, it calls function *strongStabCtr* instead of function *strongCtr*, also passing  $l$  to it.

---

### Algorithm 3 Stabilizing Controller Synthesis

---

**Input:** An LTS control problem  $(\mathcal{S}, I, G)$ ,  $\mathcal{S} = (S, I, G)$ , integer  $l \geq 0$ .

**function:** *strongStabCtr*( $\mathcal{S}, I, G, l$ )

- 1:  $K_l(s, a) \leftarrow 0, D(s) \leftarrow 0, \tilde{D}(s) \leftarrow 1$
  - 2: **while**  $D(s) \neq \tilde{D}(s)$  **do**:
  - 3:    $F(s, a) \leftarrow \exists s' T(s, a, s') \wedge \forall s' [T(s, a, s') \Rightarrow (D(s') \vee (G(s') \wedge$
  - 4:        $\wedge \forall s_1 a_1, \dots, s_l a_l s_1 = s' \wedge \forall i = 1, \dots, l [T(s_i, a_i, s_{i+1}) \Rightarrow G(s_{i+1})]]]$
  - 5:    $K_l(s, a) \leftarrow K_l(s, a) \vee (F(s, a) \wedge \nexists a K_l(s, a))$
  - 6:    $\tilde{D}(s) \leftarrow D(s), D(s) \leftarrow D(s) \vee \exists a K_l(s, a)$
  - 7: **return**  $\langle \forall s [I(s) \Rightarrow \exists a K_l(s, a)], \exists a K_l(s, a), K_l(s, a) \rangle$
- 

We then added function *strongStabCtr* to QKS, which is detailed in Algorithm 3. Functions *strongStabCtr* and *strongCtr* are similar, but with important differences. Both use a set  $D(s)$  (set of controlled states) to accumulate all found controllable states in the current iteration and a set  $\tilde{D}(s)$  to store all controlled states up to the previous iteration. Basically, the idea is to end

the computation when the procedure does not find other controllable states. Iteratively, both procedures look for new pairs state-action. After that, they update the  $\bar{D}(s)$  to the previous iteration and add new controllable states to  $D(s)$ .

The main difference is in selecting pairs state-action. The state-of-the-art condition requires that a state  $s$  is controllable if, for a given action  $a$ , it presents at least a transition to successor state and, every successor state is inside the set of controlled states. A controlled state can be inside the goal region or belongs to a path that brings the system to the goal region. Since the set of the controlled state is initialized as the set of states within the goal region, the condition works.

By contrast, stabilizing controllers are required to check also the stabilizing subpath inside  $G$ . This new condition establishes that each state is controllable if:

- it presents at least a successor state;
- every successor state either is inside the set of controlled states or belongs to the goal region along with a next stabilizing subpath of length  $l$  where each state is inside the goal region.

In such a way, a state can be controlled even when it belongs to a path that brings the system up to  $G$  and stabilizes it inside  $G$ . Making a comparison between these two conditions, we can see how stabilizing controllers incrementally restrict possible paths of the system, because any shortest stabilizing subpath than  $l$  inside the goal region is discarded.

## 5. Experimental Results

To establish the viability of our approach, we provided a C implementation of our algorithm inside QKS. We exploited the OBDD library as for the standard strong m.g.o. synthesis algorithm. Likewise, our algorithm will result in a controller implemented as a C function.

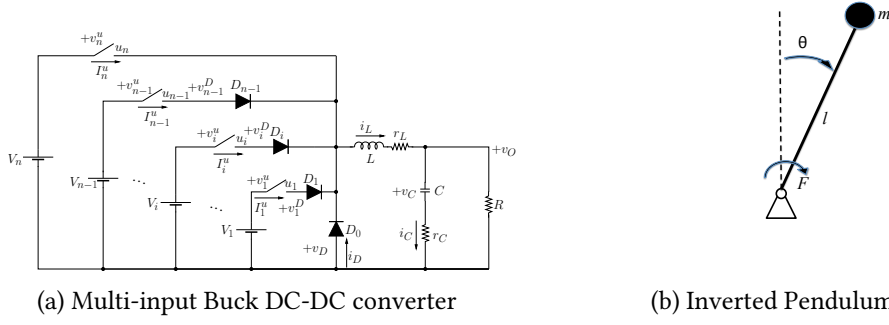
We performed several experiments on two case studies: *Multi-Buck DC-DC converter* [52, 53] and *Inverted Pendulum* [54]. We provided experimental results on several configurations of Multi-Buck DC-DC converter in terms of switches. For the inverted pendulum, instead, we have one configuration. We used several machines on High-Performance Computing (HPC) infrastructure to synthesize the output controller. The quantization of the control problem requires specifying the number of bits for the quantization schema. We carried out several runs by using 9, 10 and 11 bits. Moreover, we incremented the size of  $l$  according to obtained results. When QKS did not find a stabilizing controller for a given  $l$ , we terminated the experiments on that model.

### 5.1. Multi Buck DC-DC converter

#### 5.1.1. Case study description

Buck DC-DC converter is a mixed-mode circuit that converts DC input voltage to desired output DC output voltage. For example, it is used to scale down to typical laptop battery voltage (12-24) getting a few low voltages needed by laptop processor (e.g., [53]). Another example





**Figure 1:** Case studies for experiments

concerns to support Dynamic Voltage and Frequency Scaling (DVFS) in multicore processors (e.g., [55]). A typical software approach manages the switches  $u_1, \dots, u_n$  through a microcontroller. In such a model, shown in Figure 1a, we have  $n$  power supplies with voltage  $V_1, \dots, V_n$ .  $n$  switches with voltage  $v_1^u, \dots, v_n^u$ . Current values  $I_1^u, \dots, I_n^u$ .  $n$  input diodes  $D_0, \dots, D_{n-1}$  and current  $i_0^D, \dots, i_{n-1}^D$ . The circuit state variables are  $i_L$  and  $v_O$ . In the following we recall the guarded predicate which model the multi-buck as a DTLHS, where coefficients  $a_i$  depend on circuit parameters as follows:  $a_{1,1} = -\frac{r_L}{L}$ ,  $a_{1,2} = -\frac{1}{L}$ ,  $a_{1,3} = -\frac{1}{L}$ ,  $a_{2,1} = \frac{R}{r_c + R} [-\frac{r_c r_L}{L} + \frac{1}{C}]$ ,  $a_{2,2} = \frac{-1}{r_c + R} [\frac{r_c R}{L} + \frac{1}{C}]$ ,  $a_{2,3} = -\frac{1}{L} \frac{r_c R}{r_c + R}$ . For a complete discussion see [48]).

$$\begin{array}{lll}
 i_L' = (1 + T a_{1,1}) i_L + T a_{1,2} v_O + T a_{1,3} v_D & v_D = v_n^u - V_n & q_0 \rightarrow (i_D \geq 0) \\
 v_O' = T a_{2,1} i_L + (1 + T a_{2,2}) v_O + T a_{2,3} v_D & i_L = i_D + \sum_{i=1}^n I_i^u & \bar{q}_0 \rightarrow (v_D \leq 0) \\
 \bigwedge_{i \in [n]} q_i \rightarrow (v_i^D = R_{\text{on}} I_i^u) & q_0 \rightarrow (v_D = R_{\text{on}} i_D) & \bar{q}_0 \rightarrow (v_D = R_{\text{off}} i_D) \\
 \bigwedge_{i \in [n]} \bar{q}_i \rightarrow (v_i^D = R_{\text{off}} I_i^u) & \bigwedge_{i \in [n]} q_i \rightarrow (I_i^u \geq 0) & \bigwedge_{i \in [n]} \bar{q}_i \rightarrow (v_i^D \leq 0) \\
 \bigwedge_{j \in [n-1]} u_j \rightarrow (v_j^u = R_{\text{on}} I_j^u) & \bigwedge_{j \in [n-1]} \bar{u}_j \rightarrow (v_j^u = R_{\text{off}} I_j^u) & \bigwedge_{i \in [n]} v_D = v_i^u + v_i^D - V_i
 \end{array}$$

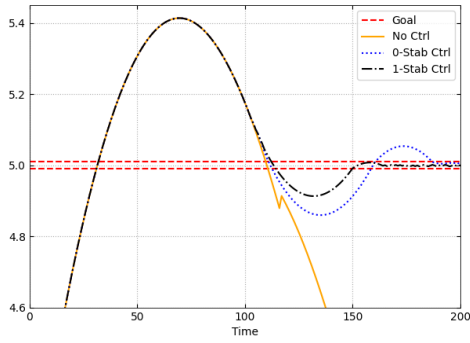
### 5.1.2. Experiments evaluation

We used three different configurations of the non-robust buck DC-DC converter. We carried out experiments with  $n = 1, 2, 3$  in the number of switches. For every configuration, as a baseline, we synthesized a controller also using the standard strong m.g.o. synthesis algorithm ( $l = 0$ ).

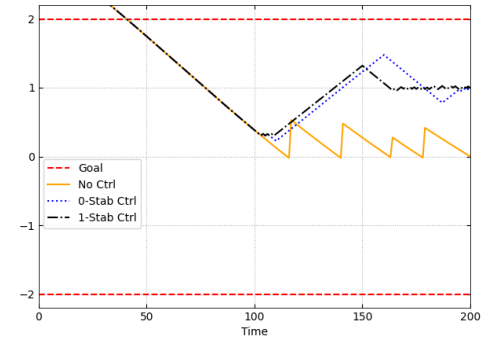
Our experimental results are shown in Table 1. Columns in Table 1 have the following meaning:  $n$  is the number of power supplies in the multi-buck, Bits is the number of bits used for the quantization of the DTLHS state variables,  $l$  is the value for the  $l$ -stabilizing controller, Nodes is the number of internal nodes of the OBDD representing the resulting  $l$ -stabilizing controller, Actions and States are the number of actions enabled and the number of states controlled by the resulting  $l$ -stabilizing controller, and finally Memory and Time provide the RAM usage in MB and the execution time in seconds, respectively.

As shown in Table 1, we are able to synthesize only 0-stabilizing controllers using 9 bits. By contrast, using 10 and 11 bits, our runs result in a strong controller with stabilizing length  $l = 1$  for  $n = 1, 2, 3$  power supplies. In any case, we did not obtain a stabilizing controller for  $l = 2$ .

We simulated the system with  $n = 1$  power supplies and  $b = 10$  bits for the AD quantization.



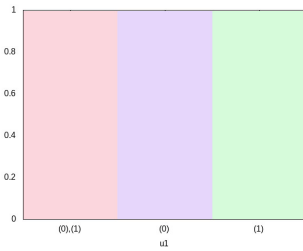
(a)  $v_O$  simulation trajectories



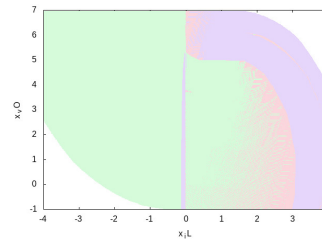
(b)  $i_L$  simulation trajectories

**Figure 2:** Multi-Buck DC-DC simulations with  $n=1$  and  $b=10$  bits

The starting state of the simulation is  $v_O = 4$  and  $i_L = 3$ . We obtained the trajectories reported in Figure 2 over a time horizon  $T \in [0, 200]$  in three different settings: without a controller, control-loop with the 0-stabilizing controller, and control-loop with the 1-stabilizing controller. In Figure 2a, we can observe how both controllers drive the system toward the goal region. However, the 1-stabilizing controller stabilizes the system faster than the 0-stabilizing controller, and it provides better stability inside the goal region. Conversely, Figure 2b shows that in any case, the current value reaches the goal, but still,  $l$ -stabilizing controllers, incrementally in terms of  $l$ , lead to better stability.



(a) Actions



(b) 1-Stabilizing Controller

**Figure 3:** Multi-Buck DC-DC experiment with  $n=1$  and  $b=10$  bits

In Figure 3 we provide a visual representation for  $n = 1$  power supplies and  $b = 10$  bits for the AD quantization. Figure 3a encodes action 1 with green, 0 with purple and any action with pink. It illustrates the distribution of actions inside the control region. Figure 3b shows the distribution of actions in the control region.

The synthesis of stabilizing controllers involves a higher computational cost. In fact, the size of  $l$  increases the amount of memory and time needed for a single computation. In such a context, both the memory usage and execution time depend on the number of bits used for the quantization schema, the size of the model and the length of the stabilizing subpath specified by the user.

**Table 1**  
Multi Buck DC-DC converter experiments

$n$	Bits	$l$	Nodes	Actions	States	Memory (MB)	Time (s)
1	9	0	3246	255309	223703	218.07	2.213460e+03
1	9	1	1	0	0	298.36	2.269390e+03
1	10	0	12207	1048770	943713	552.08	9.285320e+03
1	10	1	11865	1042325	943713	876.57	2.681168e+04
1	10	2	1	0	0	1952.13	8.743820e+03
1	11	0	32597	4312433	3840526	1832.43	3.896438e+04
1	11	1	31216	4277932	3840526	3184.59	9.040446e+04
1	11	2	1	0	0	7956.45	3.257370e+04
2	9	0	8063	318057	237420	337.75	5.760150e+03
2	9	1	1	0	0	433.08	5.258140e+03
2	10	0	25977	1290474	986115	978.55	2.171697e+04
2	10	1	25145	1260858	986115	1475.83	4.703410e+04
2	10	2	1	0	0	2975.62	1.545672e+04
2	11	0	62733	5314364	3981081	3520.31	9.021701e+04
2	11	1	60308	5202979	3981081	5392.5	1.530524e+05
2	11	2	1	0	0	12096.40	6.382998e+04
3	9	0	12150	946463	240977	558.31	1.249221e+04
3	9	1	1	0	0	682.47	1.150078e+04
3	10	0	37201	3817353	997023	1829.64	4.669121e+04
3	10	1	37101	3767342	997023	2502.92	5.028375e+04
3	10	2	1	0	0	4849.4	3.937717e+04
3	11	0	91502	15715546	4022278	6855.14	1.629664e+05
3	11	1	92178	15390590	4022278	9474.27	2.455190e+05
3	11	2	1	0	0	20331.05	1.588620e+05

## 5.2. Inverted Pendulum

### 5.2.1. Case study description

The controller synthesis for the Inverted Pendulum (IP) [54] is widely studied through QKS (e.g., [56, 57, 48]). The system, shown in Figure 1b, is modelled by using an angle  $\theta$  and the angular velocity  $\dot{\theta}$  as a state variables. The input variable is the torquing force that influences the torquing velocity in both directions. The input of the system is the torquing force  $u \cdot F$  which can influence the velocity in any direction.  $u$  is the direction and  $F$  is the intensity of the force. In this problem, we try to find a discrete controller whose actions can be: “apply the force clockwise” ( $u = 1$ ), “apply the force counterclockwise” ( $u = -1$ ) and “do nothing” ( $u = 0$ ). The behaviour of the system is based on the pendulum mass  $m$ , the length of the pendulum  $l$  and the gravitational acceleration  $g$ . The motion of the system is described by the differential equation  $\ddot{\theta} = \frac{g}{l} \sin \theta + \frac{1}{ml^2} uF$ . In the following we recall the guarded predicate which model the inverted pendulum as a DTLHS, where  $y_\xi$  variables are in  $Y$  and  $I_j$  for  $j = 1, \dots, 4$  represent a partition of the interval  $[-\pi, \pi]$ . For a complete discussion see [48].

$$\begin{aligned}
& (x'_1 = x_1 + 2\pi y_q + T x_2) \wedge (x'_2 = x_2 + T \frac{g}{l} y_{\sin} + T \frac{1}{ml^2} u F) \\
& \wedge \bigwedge_{i \in [4]} y_i \rightarrow f_i^-(y_\alpha) \leq y_{\sin} \leq f_i^+(y_\alpha) \wedge \bigwedge_{i \in [4]} y_i \rightarrow y_\alpha \in I_i \wedge \sum_{i \in [4]} y_i \geq 1 \\
& \wedge x_1 = 2\pi y_k + y_\alpha \wedge -\pi \leq x'_1 \leq \pi
\end{aligned}$$

### 5.2.2. Experiments evaluation

Our experimental results are shown in Table 2. Columns in Table 2 have the same meaning of the columns with the same name in Table 1. We note that, in such a case study, only 0-stabilizing controllers exist.

**Table 2**

Inverted pendulum experiments

Bits	$l$	Nodes	Actions	States	Memory (MB)	Time
9	0	12238	453843	262144	324.87	7.200670e+03
9	1	1	0	0	455.29	6.550130e+03
10	0	24878	1831288	1048576	916.39	3.348904e+04
10	1	1	0	0	1495.06	3.536044e+04
11	0	51110	7335238	4194304	3229.16	1.658496e+05
11	1	1	0	0	5602.21	1.640529e+05

## 6. Conclusions

We provided a new concept of controllers along with a specific synthesis algorithm. Ensuring the system meets liveness/safety property is a key goal in safety/mission-critical contexts. On such a basis, the development of correct-by-construction controllers increasing safety guarantees can become a challenge for many CPSs. On the other hand, the experiments showed several settings in which we can not obtain stabilizing controllers. Moreover, each performed run required more computational resources (*i.e.*, time and memory) than the standard synthesis algorithm. However, the need for correctness of safety/mission-critical systems justifies the usage of a higher quantity of computational resources.

As future works, we may go in several directions. Firstly, we plan to provide a simulation environment as a tool to observe the resulting trajectories of the input system in the closed-loop with the controller provided by QKS. Secondly, to extend the algorithm of weak controller synthesis to give the user some information about stabilizing controllers (if the strong algorithm fails). Thirdly, to perform computational evaluations to establish the computational cost in terms of user-defined input parameters. Last but not least, we plan to provide more experimental results on several case studies.

## Acknowledgments

This work was partially supported by: Italian Ministry of University and Research under grant “Dipartimenti di eccellenza 2018–2022” of the Department of Computer Science of Sapienza University of Rome.

## References

- [1] A. Basu, S. Bensalem, M. Bozga, B. Delahaye, A. Legay, E. Sifakis, Verification of an AFDX infrastructure using simulations and probabilities, in: RV 2010, volume 6418 of *LNCIS*, Springer, 2010. doi:10.1007/978-3-642-16612-9\_25.
- [2] L. Planke, Y. Lim, A. Gardi, R. Sabatini, T. K., N. Ezer, A cyber-physical-human system for one-to-many uas operations: Cognitive load analysis, *Sensors* 20 (2020). doi:10.3390/s20195467.
- [3] Z. Wu, N. Huang, X. Zheng, X. Li, Cyber-physical avionics systems and its reliability evaluation, in: IEEE CYBER 2014, IEEE, 2014. doi:10.1109/CYBER.2014.6917502.
- [4] K. Sampigethaya, R. Poovendran, Aviation cyber-physical systems: Foundations for future aircraft and air transport, *Proc. IEEE* 101 (2013). doi:10.1109/JPROC.2012.2235131.
- [5] T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J. Gruber, B. Hayes, M. Prodanovic, L. Elmegaard, Demand-aware price policy synthesis and verification services for smart grids, in: SmartGridComm 2014, IEEE, 2014. doi:10.1109/SmartGridComm.2014.7007745.
- [6] I. Melatti, F. Mari, T. Mancini, M. Prodanovic, E. Tronci, A two-layer near-optimal strategy for substation constraint management via home batteries, *IEEE Trans. Ind. Elect.* (2021). doi:10.1109/TIE.2021.3102431.
- [7] B. Hayes, I. Melatti, T. Mancini, M. Prodanovic, E. Tronci, Residential demand management using individualised demand aware price policies, *IEEE Trans. Smart Grid* 8 (2017). doi:10.1109/TSG.2016.2596790.
- [8] T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J. Gruber, B. Hayes, M. Prodanovic, L. Elmegaard, User flexibility aware price policy synthesis for smart grids, in: DSD 2015, IEEE, 2015. doi:10.1109/DSD.2015.35.
- [9] T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J. Gruber, B. Hayes, L. Elmegaard, Parallel statistical model checking for safety verification in smart grids, in: SmartGridComm 2018, IEEE, 2018. doi:10.1109/SmartGridComm.2018.8587416.
- [10] D. Goswami, R. Schneider, A. Masrur, M. Lukasiewicz, S. Chakraborty, H. Voit, A. Anaswamy, Challenges in automotive cyber-physical systems design, in: SAMOS 2012, IEEE, 2012. doi:10.1109/SAMOS.2012.6404199.
- [11] S. Chakraborty, M. Al Faruque, W. Chang, D. Goswami, M. Wolf, Q. Zhu, Automotive cyber-physical systems: A tutorial introduction, *IEEE Des.&Test* 33 (2016). doi:10.1109/MDAT.2016.2573598.
- [12] L. Zhang, Modeling automotive cyber physical systems, in: DCABES 2013, IEEE, 2013. doi:10.1109/DCABES.2013.20.
- [13] S. Osswald, S. Matz, M. Lienkamp, Prototyping automotive cyber-physical systems, in:

Adjunct Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, 2014, pp. 1–6.

- [14] M. Hengartner, T. Kruger, K. Geraedts, E. Tronci, T. Mancini, F. Ille, M. Egli, S. Roebnitz, R. Ehrig, L. Saleh, K. Spanaus, C. Schippert, Y. Zhang, B. Leeners, Negative affect is unrelated to fluctuations in hormone levels across the menstrual cycle: Evidence from a multisite observational study across two successive cycles, *J. Psycho. Res.* 99 (2017). doi:10.1016/j.jpsychores.2017.05.018.
- [15] B. Leeners, T. Kruger, K. Geraedts, E. Tronci, T. Mancini, F. Ille, M. Egli, S. Roebnitz, L. Saleh, K. Spanaus, C. Schippert, Y. Zhang, M. Hengartner, Lack of associations between female hormone levels and visuospatial working memory, divided attention and cognitive bias across two consecutive menstrual cycles, *Front. Behav. Neuro.* 11 (2017). doi:10.3389/fnbeh.2017.00120.
- [16] B. Leeners, T. Krüger, K. Geraedts, E. Tronci, T. Mancini, M. Egli, S. Röblitz, L. Saleh, K. Spanaus, C. Schippert, Y. Zhang, F. Ille, Associations between natural physiological and supraphysiological estradiol levels and stress perception, *Front. Psychol.* 10 (2019). doi:10.3389/fpsyg.2019.01296.
- [17] F. Maggioli, T. Mancini, E. Tronci, SBML2Modelica: Integrating biochemical models within open-standard simulation ecosystems, *Bioinformatics* 36 (2020). doi:10.1093/bioinformatics/btz860.
- [18] T. Mancini, E. Tronci, I. Salvo, F. Mari, A. Massini, I. Melatti, Computing biological model parameters by parallel statistical model checking, in: *IWBBIO 2015*, volume 9044 of *LNCS*, Springer, 2015. doi:10.1007/978-3-319-16480-9\_52.
- [19] T. Mancini, F. Mari, A. Massini, I. Melatti, I. Salvo, S. Sinisi, E. Tronci, R. Ehrig, S. Röblitz, B. Leeners, Computing personalised treatments through in silico clinical trials. A case study on downregulation in assisted reproduction, in: *RCRA 2018*, volume 2271 of *CEUR W.P.*, CEUR, 2018.
- [20] E. Tronci, T. Mancini, I. Salvo, S. Sinisi, F. Mari, I. Melatti, A. Massini, F. Davi', T. Dierkes, R. Ehrig, S. Röblitz, B. Leeners, T. Krüger, M. Egli, F. Ille, Patient-specific models from inter-patient biological models and clinical records, in: *FMCAD 2014*, IEEE, 2014. doi:10.1109/FMCAD.2014.6987615.
- [21] S. Sinisi, V. Alimguzhin, T. Mancini, E. Tronci, B. Leeners, Complete populations of virtual patients for in silico clinical trials, *Bioinformatics* 36 (2020). doi:10.1093/bioinformatics/btaa1026.
- [22] S. Sinisi, V. Alimguzhin, T. Mancini, E. Tronci, F. Mari, B. Leeners, Optimal personalised treatment computation through in silico clinical trials on patient digital twins, *Fundam. Inform.* 174 (2020). doi:10.3233/FI-2020-1943.
- [23] S. Sinisi, V. Alimguzhin, T. Mancini, E. Tronci, Reconciling interoperability with efficient verification and validation within open source simulation environments, *Simul. Model. Pract. Theory* 109 (2021). doi:10.1016/j.simpat.2021.102277.
- [24] W. Brogan, *Modern Control Theory* (3rd Ed.), Prentice H., 1991. doi:10.5555/135299.
- [25] M. Cadoli, T. Mancini, F. Patrizi, SAT as an effective solving technology for constraint problems, in: *ISMIS 2006*, volume 4203 of *LNCS*, Springer, 2006.
- [26] M. Cadoli, T. Mancini, Combining relational algebra, SQL, constraint modelling, and local search, *TPLP* 7 (2007). doi:10.1017/S1471068406002857.

- [27] T. Mancini, P. Flener, J. Pearson, Combinatorial problem solving over relational databases: View synthesis through constraint-based local search, in: SAC 2012, ACM, 2012. doi:10.1145/2245276.2245295.
- [28] G. Gottlob, G. Greco, T. Mancini, Conditional constraint satisfaction: Logical foundations and complexity, in: IJCAI 2007, 2007.
- [29] T. Mancini, M. Cadoli, Detecting and breaking symmetries by reasoning on problem specifications, in: SARA 2005, volume 3607 of LNCS, Springer, 2005.
- [30] T. Mancini, M. Cadoli, D. Micaletto, F. Patrizi, Evaluating ASP and commercial solvers on the CSPLib, Constraints 13 (2008).
- [31] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT, 1999.
- [32] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, M. Venturini Zilli, Bounded probabilistic model checking with the Mur $\varphi$  verifier, in: FMCAD 2004, IEEE, 2004.
- [33] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, E. Tronci, System level formal verification via model checking driven simulation, in: CAV 2013, volume 8044 of LNCS, Springer, 2013. doi:10.1007/978-3-642-39799-8\_21.
- [34] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, System level formal verification via distributed multi-core hardware in the loop simulation, in: PDP 2014, IEEE, 2014. doi:10.1109/PDP.2014.32.
- [35] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, Anytime system level verification via random exhaustive hardware in the loop simulation, in: DSD 2014, IEEE, 2014. doi:10.1109/DSD.2014.91.
- [36] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, SyLVaaS: System level formal verification as a service, in: PDP 2015, IEEE, 2015. doi:10.1109/PDP.2015.119.
- [37] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, SyLVaaS: System level formal verification as a service, Fundam. Inform. 149 (2016). doi:10.3233/FI-2016-1444.
- [38] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, Anytime system level verification via parallel random exhaustive hardware in the loop simulation, Microprocessors and Microsystems 41 (2016). doi:10.1016/j.micpro.2015.10.010.
- [39] T. Mancini, F. Mari, A. Massini, I. Melatti, I. Salvo, E. Tronci, On minimising the maximum expected verification time, Inf. Proc. Lett. 122 (2017). doi:10.1016/j.ip1.2017.02.001.
- [40] T. Mancini, I. Melatti, E. Tronci, Any-horizon uniform random sampling and enumeration of constrained scenarios for simulation-based formal verification, IEEE TSE (2021). doi:10.1109/TSE.2021.3109842.
- [41] M. Mazo, A. Davitian, P. Tabuada, PESSOA: A tool for embedded controller synthesis, in: CAV 2010, volume 6174 of LNCS, Springer, 2010. doi:10.1007/978-3-642-14295-6\_49.
- [42] H. Peter, R. Ehlers, R. Mattmüller, Synthia: Verification and synthesis for timed automata, in: CAV 2011, volume 6806 of LNCS, Springer, 2011. doi:10.1007/978-3-642-22110-1\_52.
- [43] F. Mari, I. Melatti, I. Salvo, E. Tronci, Model based synthesis of control software from system level formal specifications, ACM TOSEM 23 (2014).
- [44] F. Dullerud, G.E. and Paganini, Stabilizing Controllers, Springer, 2000. doi:10.1007/978-1-4757-3290-0\_6.
- [45] V. M. Popov, R. Georghescu, Hyperstability of Control Systems, Springer, 1973. doi:10.

5555/578779.

- [46] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, On-the-fly control software synthesis, in: SPIN 2013, volume 7976 of *LNCS*, Springer, 2013. doi:10.1007/978-3-642-39176-7\_5.
- [47] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, A map-reduce parallel approach to automatic synthesis of control software, in: SPIN 2013, volume 7976 of *LNCS*, Springer, 2013. doi:10.1007/978-3-642-39176-7\_4.
- [48] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, On model based synthesis of embedded control software, in: EMSOFT 2012, ACM, 2012. doi:10.1145/2380356.2380398.
- [49] F. Mari, I. Melatti, I. Salvo, E. Tronci, Undecidability of quantized state feedback control for discrete time linear hybrid systems, in: ICTAC 2012, volume 7521 of *LNCS*, Springer, 2012. doi:10.1007/978-3-642-32943-2\_19.
- [50] R. Bryant, Binary decision diagrams and beyond: Enabling technologies for formal verification, in: ICCAD 1995, IEEE, 1995. doi:10.1109/ICCAD.1995.480018.
- [51] R. Drechsler, J. Shi, G. Fey, Synthesis of fully testable circuits from BDDs, *IEEE Trans. Comp.-Aided Des. Integr. Circ. & Sys.* 23 (2004). doi:10.1109/TCAD.2004.823342.
- [52] G. Schrom, P. Hazucha, J. Hahn, D. S. Gardner, B. A. Bloechel, G. Dermer, S. G. Narendra, T. Karnik, V. De, A 480-mhz, multi-phase interleaved buck dc-dc converter with hysteretic control, in: 2004 IEEE 35th annual power electronics specialists conference (IEEE Cat. No. 04CH37551), volume 6, IEEE, 2004, pp. 4702–4707.
- [53] W.-C. So, C. Tse, Y.-S. Lee, Development of a fuzzy logic controller for DC/DC converters: Design, computer simulation, and experimental evaluation, *IEEE Trans. Pow. Electr.* 11 (1996).
- [54] G. Kreisselmeier, T. Birkholzer, Numerical nonlinear regulator design, *IEEE Trans. Aut. Contr.* 39 (1994).
- [55] W. Kim, M. S. Gupta, G.-Y. Wei, D. M. Brooks, Enabling on-chip switching regulators for multi-core processors using current staggering, *Proceedings of the Work. on Architectural Support for Gigascale Integration* (2007).
- [56] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, Linearizing discrete-time hybrid systems, *IEEE TAC* 62 (2017). doi:10.1109/TAC.2017.2694559.
- [57] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, Automatic control software synthesis for quantized discrete time hybrid systems, in: CDC 2012, IEEE, 2012. doi:10.1109/CDC.2012.6426260.