Model Based Synthesis of Control Software from System Level Formal Specifications

FEDERICO MARI, IGOR MELATTI, IVANO SALVO, and ENRICO TRONCI,
Sapienza University of Rome

Many *Embedded Systems* are indeed *Software Based Control Systems*, that is control systems whose controller consists of *control software* running on a microcontroller device. This motivates investigation on *Formal Model Based Design* approaches for automatic synthesis of embedded systems control software. We present an algorithm, along with a tool QKS implementing it, that from a formal model (as a *Discrete Time Linear Hybrid System*) of the controlled system (*plant*), *implementation specifications* (that is, number of bits in the *Analog-to-Digital*, AD, conversion) and *System Level Formal Specifications* (that is, safety and liveness requirements for the *closed loop system*) returns correct-by-construction control software that has a *Worst Case Execution Time* (WCET) linear in the number of AD bits and meets the given specifications. We show feasibility of our approach by presenting experimental results on using it to synthesize control software for a buck DC-DC converter, a widely used mixed-mode analog circuit, and for the inverted pendulum.

Categories and Subject Descriptors: D.2.2 [**Software**]: Design Tools and Techniques—*Computer Aided Software Engineering*; D.2.4 [**Software**]: Software/Program Verification—*Model Checking, Formal Methods*

General Terms: Verification

Additional Key Words and Phrases: Hybrid Systems, Correct-By-Construction Control Software Synthesis, Model-Based Design of Control Software

1. INTRODUCTION

Many *Embedded Systems* are indeed *Software Based Control Systems* (SBCS). An SBCS consists of two main subsystems: the *controller* and the *plant*. Typically, the plant is a physical system consisting, for example, of mechanical or electrical devices whereas the controller consists of *control software* running on a microcontroller (see Fig. 2). In an endless loop, the controller reads *sensor* outputs from the plant and sends commands to plant *actuators* in order to guarantee that the *closed loop system* (that is, the system consisting of both plant and controller) meets given *safety* and *liveness* specifications (*System Level Formal Specifications*). Missing such goals can cause failures or damages to the plant, thus making an SBCS a hard real-time system.

Software generation from models and formal specifications forms the core of *Model Based Design* of embedded software [Henzinger and Sifakis 2006]. This approach is particularly interesting for SBCSs since in such a case system level (formal) specifications are much easier to define than the control software behavior itself.

Fig. 1 shows the typical control loop skeleton for an SBCS. Measures from plant *sensors* go through an *Analog-to-Digital* (AD) conversion (*quantization*) before being processed (line 2) and commands from the control software go through a *Digital-to-Analog* (DA) conversion before being sent to plant *actuators* (line 8). Basically, the control software design problem for SBCSs consists in designing software implementing functions `Control_Law` and `Controllable_Region` computing, respectively, the command to be sent to the plant (line 7) and the set of states on which the `Control_Law` function works correctly (*Fault Detection* in line 3). Fig. 2 summarizes the complete closed loop system forming an SBCS.

Authors' address: Dipartimento di Informatica, Sapienza Università di Roma, Via Salaria 113, 00198 Roma, Italy.

DOI 10.1145/2559934 http://doi.acm.org/10.1145/2559934

1. **Every** T seconds (*sampling time)* **do**
2. **Read** AD conversion $\hat{x}$ of plant sensor outputs x
3. **If** ($\hat{x}$ is not in the `Controllable_Region`)
4. **Then** // *Exception (Fault Detected):*
5. Start Fault Isolation and Recovery (`FDIR`)
6. **Else** // *Nominal case:*
7. Compute (`Control_Law`) command $\hat{u}$ from $\hat{x}$
8. **Send** DA conversion u of $\hat{u}$ to plant actuators

Fig. 1. A typical control loop skeleton.

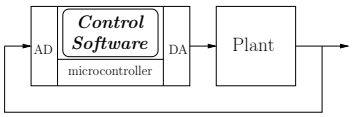


Fig. 2. Software Based Control System.

1.1. The Separation-of-Concerns Approach

For SBCS system level specifications are typically given with respect to the desired behavior of the closed loop system. The *control software* (that is, `Control_Law` and `Controllable_Region`) is designed using a *separation-of-concerns* approach. That is, *Control Engineering* techniques (e.g., see [Brogan 1991]) are used to design, from the closed loop system level specifications, *functional specifications* (*control law*) for the *control software* whereas *Software Engineering* techniques are used to design control software implementing the given functional specifications.

Such a separation-of-concerns approach has several drawbacks.

First, usually control engineering techniques do not yield a formally verified specification for the control law or controllable region when quantization is taken into account. This is particularly the case when the plant has to be modelled as a *Hybrid System* [Alur and Madhusudan 2004; Alur et al. 1995; Henzinger et al. 1997; Alur et al. 1996] (that is a system with continuous as well as discrete state changes). As a result, even if the control software meets its functional specifications there is no formal guarantee that system level specifications are met since quantization effects are not formally accounted for.

Second, issues concerning computational resources, such as control software *Worst Case Execution Time* (WCET), can only be considered very late in the SBCS design activity, namely once the software has been designed. As a result, since the SBCS is a hard real-time system (Fig. 2), the control software may have a WCET greater than the sampling time (line 1 in Fig. 1). This invalidates the schedulability analysis (typically carried out before the control software is completed) and may trigger redesign of the software or even of its functional specifications (in order to simplify its design).

Last, but not least, the classical separation-of-concerns approach does not effectively support design space exploration for the control software. In fact, although in general there will be many functional specifications for the control software that will allow meeting the given system level specifications, the software engineer only gets one to play with. This overconstrains a priori the design space for the control software implementation preventing, for example, effective performance trading (e.g., between number of bits in AD conversion, WCET, RAM usage, CPU power consumption, etc.).

We note that the above considerations also apply to the typical situation where Control Engineering techniques are used to design a control law and then tools like Berkeley's Ptolemy [Eker et al. 2003], Esterel's SCADE [SCADE Web Page 2012] or MathWorks Simulink [Simulink Web Page 2012] are used to generate the control software. Even when the control law is automatically generated and proved correct (for example, as in [Mazo et al. 2010]) such an approach does not yield any formal guarantee about the software correctness since quantization of the state measurements is not taken into account in the computation of the control law. Thus such an approach cannot answer questions like: 1) Can 8 bit AD be used or instead we need, say, 12 bit AD? 2) Will the control software code run *fast enough* on a, say, 1 MIPS microcontroller (that is, is the control software WCET less than the sampling time)? 3) What is the controllable region?

The previous considerations motivate research on *Software Engineering* methods and tools focusing on control software synthesis (rather than on control law synthesis as in *Control Engineering*). The objective is that from the plant model (as a hybrid system), from formal specifications for the closed loop system behavior (*System Level Formal Specifications*) and from *Implementation Specifications* (that is, number of bits used in the quantization process) such methods and tools can generate correct-by-construction control software satisfying the given specifications. This is the focus of the present paper.

For a more in-depth discussion of the literature related to the present paper, we refer the reader to Sect. 9 and Tab. VI.

1.2. Our Main Contributions

We model the controlled system (plant) as a *Discrete Time Linear Hybrid System* (DTLHS) (see Sect. 3), that is a discrete time hybrid system whose dynamics is defined as a *linear predicate* (i.e., a boolean combination of linear constraints, see Sect. 2) on its variables. We model system level safety as well as liveness specifications as sets of states defined, in turn, as linear predicates. In our setting, as always in control problems, liveness constraints define the set of states that any evolution of the closed loop system should eventually reach (*goal states*). Using an approach similar to the one in [Henzinger and Kopke 1997; Henzinger et al. 1998; Agrawal et al. 2006], in [Mari et al. 2012c] we prove that both existence of a controller for a DTLHS and existence of a *quantized* controller for a DTLHS are undecidable problems. Accordingly, we can only hope for semi- or incomplete algorithms.

We present an algorithm computing a sufficient condition and a necessary condition for existence of a solution to our control software synthesis problem (see Sects. 4 and 5). Given a DTLHS model $\mathcal{H}$ for the plant, a quantization schema (i.e. how many bits we use for AD conversion) and system level formal specifications, our algorithm (see Sect. 6) will return 1 if they are able to decide if a solution exists or not, and 0 otherwise (unavoidable case since our problem is undecidable). Furthermore, when our sufficient condition is satisfied, we return a pair of C functions (see Sect. 7) `Control_Law`, `Controllable_Region` such that: function `Control_Law` implements a *Quantized Feedback Controller* (QFC) for $\mathcal{H}$ meeting the given system level formal specifications and function `Controllable_Region` computes the set of states on which `Control_Law` is guaranteed to work correctly (*controllable region*). While *WCET analysis* is actually performed after control software generation, our contribution is to supply both functions with a *Worst Case Execution Time* (WCET) guaranteed to be linear in the number of bits of the state quantization schema (see Sect. 7.1). Furthermore, function `Control_Law` is *robust*, that is, it meets the given closed loop requirements notwithstanding (nondeterministic) *disturbances* such as variations in the plant parameters.

We implemented our algorithm on top of the CUDD package and of the GLPK *Mixed Integer Linear Programming* (MILP) solver, thus obtaining tool *Quantized feedback Kontrol Synthesizer* (QKS) (publicly available at [QKS Web Page 2011]). This allows us to present experimental results on using QKS to synthesize robust control software for a widely used mixed-mode analog circuit: the buck DC-DC converter (e.g. see [So et al. 1996]). This is an interesting and challenging example (e.g., see [Dominguez-Garcia and Krein 2008], [Yousefzadeh et al. 2008]) for automatic synthesis of correct-by-construction control software from system level formal specifications. Moreover, in order to show effectiveness of our approach, we also present experimental results on using QKS for the inverted pendulum [Kreisselmeier and Birkhölzer 1994].

Our experimental results address both computational feasibility and closed loop performances. As for computational feasibility, we show that within about 40 hours of CPU time and within 100MB of RAM we can synthesize control software for a 10-bit quantized buck DC-DC converter. As for closed loop performances, our synthesized control software set-up time (i.e., the time needed to reach the steady state) and ripple (i.e., the wideness of the oscillations around the steady state once this has been reached) compares well with those

available from the Power Electronics community [So et al. 1996; Yousefzadeh et al. 2008] and from commercial products [Texas Instruments 2001].

2. BACKGROUND

We denote with $[n]$ an initial segment $\{1, \ldots, n\}$ of the natural numbers. We denote with $X = [x_1, \ldots, x_n]$ a finite sequence (list) of variables. By abuse of language we may regard sequences as sets and we use $\cup$ to denote list concatenation. Each variable x ranges on a known (bounded or unbounded) interval $\mathcal{D}_x$ either of the reals or of the integers (discrete variables). We denote with $\mathcal{D}_X$ the set $\prod_{x \in X} \mathcal{D}_x$. To clarify that a variable x is *continuous* (i.e. real valued) we may write x^r. Similarly, to clarify that a variable x is *discrete* (i.e. integer valued) we may write x^d. Boolean variables are discrete variables ranging on the set $\mathbb{B} = \{0, 1\}$. We may write x^b to denote a boolean variable. Analogously X^r (X^d, X^b) denotes the sequence of real (integer, boolean) variables in X. Unless otherwise stated, we suppose $\mathcal{D}_{X^r} = \mathbb{R}^{|X^r|}$ and $\mathcal{D}_{X^d} = \mathbb{Z}^{|X^d|}$. Finally, if x is a boolean variable we write $\bar{x}$ for $(1 - x)$.

2.1. Predicates

A *linear expression* $L(X)$ over a list of variables X is a linear combination of variables in X with rational coefficients, $\sum_{x_i \in X} a_i x_i$. A *linear constraint* over X (or simply a *constraint*) is an expression of the form $L(X) \leq b$, where $L(X)$ is a linear expression over X and b is a rational constant. In the following, we also write $L(X) \geq b$ for $-L(X) \leq -b$.

Predicates are inductively defined as follows. A constraint $C(X)$ over a list of variables X is a predicate over X. If $A(X)$ and $B(X)$ are predicates over X, then $(A(X) \wedge B(X))$ and $(A(X) \vee B(X))$ are predicates over X. Parentheses may be omitted, assuming usual associativity and precedence rules of logical operators. A *conjunctive predicate* is a conjunction of constraints. For conjunctive predicates we will also write: $L(X) = b$ for $((L(X) \leq b) \wedge (L(X) \geq b))$ and $a \leq x \leq b$ for $x \geq a \ \wedge \ x \leq b$, where $x \in X$.

A *valuation* over a list of variables X is a function v that maps each variable $x \in X$ to a value $v(x) \in \mathcal{D}_x$. Given a valuation v, we denote with $X^* \in \mathcal{D}_X$ the sequence of values $[v(x_1), \ldots, v(x_n)]$. By abuse of language, we call valuation also the sequence of values X^*. A *satisfying assignment* to a predicate P over X is a valuation X^* such that $P(X^*)$ holds. If a satisfying assignment to a predicate P over X exists, we say that P is *feasible*. Abusing notation, we may denote with P the set of satisfying assignments to the predicate $P(X)$. Two predicates P and Q over X are *equivalent*, denoted by $P \equiv Q$, if they have the same set of satisfying assignments.

A variable $x \in X$ is said to be *bounded* in P if there exist a, $b \in \mathcal{D}_x$ such that $P(X)$ implies $a \leq x \leq b$. A predicate P is bounded if all its variables are bounded.

Given a constraint $C(X)$ and a fresh boolean variable (*guard*) $y \notin X$, the *guarded constraint* $y \rightarrow C(X)$ (if y then $C(X)$) denotes the predicate $((y = 0) \vee C(X))$. Similarly, we use $\bar{y} \rightarrow C(X)$ (if not y then $C(X)$) to denote the predicate $((y = 1) \vee C(X))$. A *guarded predicate* is a conjunction of either constraints or guarded constraints. It is possible to show that, if a guarded predicate P is bounded, then P can be transformed into a (bounded) conjunctive predicate, see [Mari et al. 2012b].

2.2. Mixed Integer Linear Programming

A *Mixed Integer Linear Programming* (MILP) problem with *decision variables* X is a tuple (max, $J(X)$, $A(X)$) where: X is a list of variables, $J(X)$ (*objective function*) is a linear expression on X, and $A(X)$ (*constraints*) is a conjunctive predicate on X. A *solution* to $(\max, J(X), A(X))$ is a valuation X^* such that $A(X^*)$ and $\forall Z \ (A(Z) \ \rightarrow \ (J(Z) \leq J(X^*)))$. $J(X^*)$ is the *optimal value* of the MILP problem. A *feasibility* problem is a MILP

problem of the form $(\max, 0, A(X))$. We write also $A(X)$ for $(\max, 0, A(X))$. We write $(\min, J(X), A(X))$ for $(\max, -J(X), A(X))$.

In algorithm outlines, MILP solver invocations are denoted by function *feasible*$(A(X))$ that returns TRUE if $A(X)$ is feasible and FALSE otherwise, and function *optimalValue*(max, $J(X)$, $A(X)$) that returns either the optimal value of the MILP problem (max, $J(X)$, $A(X)$) or $+\infty$ if such MILP problem is unbounded or unfeasible.

2.3. 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 $T : S \times A \times S \to \mathbb{B}$ is the *transition relation* of $\mathcal{S}$. We say that T (and $\mathcal{S}$) is *deterministic* if $T(s, a, s') \wedge T(s, a, s'')$ implies $s' = s''$, and *nondeterministic* otherwise. Let $s \in S$ and $a \in A$. We denote with $\mathrm{Adm}(\mathcal{S}, s)$ the set of actions admissible in s, that is $\mathrm{Adm}(\mathcal{S}, s) = \{a \in A \mid \exists s' : T(s, a, s')\}$ and with $\mathrm{Img}(\mathcal{S}, s, a)$ the set of next states from s via a, that is $\mathrm{Img}(\mathcal{S}, s, a) = \{s' \in S \mid T(s, a, s')\}$. We call *transition* a triple $(s, a, s') \in S \times A \times S$, and *self loop* a transition (s, a, s). A transition (s, a, s') [self loop (s, a, s)] is a *transition [self loop] of* $\mathcal{S}$ iff $T(s, a, s')$ $[T(s, a, s)]$. A *run* or *path* for an LTS $\mathcal{S}$ is a sequence $\pi = s_0, a_0, s_1, a_1, s_2, a_2, \ldots$ of states s_t and actions a_t such that $\forall t \geq 0\ T(s_t, a_t, s_{t+1})$. The length $|\pi|$ of a finite run π is the number of actions in π. We denote with $\pi^{(S)}(t)$ the $(t + 1)$-th state element of π, and with $\pi^{(A)}(t)$ the $(t + 1)$-th action element of π. That is $\pi^{(S)}(t) = s_t$, and $\pi^{(A)}(t) = a_t$.

Given two LTSs $\mathcal{S}_1 = (S, A, T_1)$ and $\mathcal{S}_2 = (S, A, T_2)$, we say that $\mathcal{S}_1$ *refines* $\mathcal{S}_2$ (denoted by $\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$) iff $T_1(s, a, s')$ implies $T_2(s, a, s')$ for each state $s, s' \in S$ and action $a \in A$. The refinement relation is a partial order on LTSs.

3. DISCRETE TIME LINEAR HYBRID SYSTEMS

In this section we introduce our class of *Discrete Time Linear Hybrid System* (DTLHS), together with the DTLHS representing the buck DC-DC converter on which our experiments will focus.

Definition 3.1 (*DTLHS*). A Discrete Time Linear Hybrid System is a tuple $\mathcal{H} = (X, U, Y, N)$ where:

- $X = X^r \cup X^d$ is a finite sequence of real (X^r) and discrete (X^d) *present state* variables. We denote with X' the sequence of *next state* variables obtained by decorating with $'$ all variables in X.
- $U = U^r \cup U^d$ is a finite sequence of *input* variables.
- $Y = Y^r \cup Y^d$ is a finite sequence of *auxiliary* variables. Auxiliary variables are typically used to model *modes* (e.g., from switching elements such as diodes) or "local" variables.
- $N(X, U, Y, X')$ is a conjunctive predicate over $X \cup U \cup Y \cup X'$ defining the *transition relation* (*next state*) of the system. N is *deterministic* if $N(x, u, y_1, x') \wedge N(x, u, y_2, x'')$ implies $x' = x''$, and *nondeterministic* otherwise.

A DTLHS is *bounded* if predicate N is bounded. A DTLHS is *deterministic* if N is deterministic.

Since any bounded guarded predicate can be transformed into a conjunctive predicate (see Sect. 2.1), for the sake of readability we will use bounded guarded predicates to describe the transition relation of bounded DTLHSs. To this aim, we will also clarify which variables are boolean, and thus may be used as guards in guarded constraints.

Example 3.2. Let x be a continuous variable, u be a boolean variable, and $N(x, u, x') \equiv [\overline{u} \to x' = \alpha x] \wedge [u \to x' = \beta x] \wedge -4 \leq x \leq 4$ be a guarded predicate with $\alpha = \frac{1}{2}$ and $\beta = \frac{3}{2}$. Then $\mathcal{H} = (\{x\}, \{u\}, \varnothing, N)$ is a bounded DTLHS. Note that $\mathcal{H}$ is deterministic. Adding nondeterminism to $\mathcal{H}$ allows us to address the problem of (bounded) variations

in the DTLHS parameters. For example, variations in the parameter α can be modelled with a tolerance $\rho \in [0, 1]$ for α. This replaces N with: $N^{(\rho)} \equiv [\overline{u} \rightarrow x' \leq (1+\rho)\alpha x] \wedge [\overline{u} \rightarrow x' \geq (1-\rho)\alpha x] \wedge [u \rightarrow x' = \beta x]$. We have that $\mathcal{H}^{(\rho)} = (\{x\}, \{u\}, \varnothing, N^{(\rho)})$, for $\rho \in (0, 1]$, is a nondeterministic DTLHS. Note that, as expected, $\mathcal{H}^{(0)} = \mathcal{H}$.

In the following definition, we give the semantics of DTLHSs in terms of LTSs.

Definition 3.3 (*DTLHS dynamics*). Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS. The dynamics of $\mathcal{H}$ is defined by the Labeled Transition System $\text{LTS}(\mathcal{H}) = (\mathcal{D}_X, \mathcal{D}_U, \tilde{N})$ where: $\tilde{N} : \mathcal{D}_X \times \mathcal{D}_U \times \mathcal{D}_X \rightarrow \mathbb{B}$ is a function s.t. $\tilde{N}(x, u, x') \equiv \exists y \in \mathcal{D}_Y : N(x, u, y, x')$. A *state* x for $\mathcal{H}$ is a state x for $\text{LTS}(\mathcal{H})$ and a *run* (or *path*) for $\mathcal{H}$ is a run for $\text{LTS}(\mathcal{H})$ (Sect. 2.3).

Example 3.4. Let $\mathcal{H}$ be the DTLHS of Ex. 3.2. Then a sequence π is a run for $\mathcal{H}$ iff state $\pi^{(S)}(i+1)$ is obtained by multiplying $\pi^{(S)}(i)$ by $\frac{3}{2}$ when $\pi^{(A)}(i) = 1$, and by $\frac{1}{2}$ when $\pi^{(A)}(i) = 0$.

3.1. Buck DC-DC Converter as a DTLHS

The buck DC-DC converter (Fig. 3) is a mixed-mode analog circuit converting the DC input voltage (V_i in Fig. 3) to a desired DC output voltage (v_O in Fig. 3). As an example, buck DC-DC converters are used off-chip to scale down the typical laptop battery voltage (12-24) to the just few volts needed by the laptop processor (e.g. [So et al. 1996]) as well as on-chip to support *Dynamic Voltage and Frequency Scaling* (DVFS) in multicore processors (e.g. [Kim et al. 2007; Schrom et al. 2004]). Because of its widespread use, control schemas for buck DC-DC converters have been widely studied (e.g. see [Kim et al. 2007; Schrom et al. 2004; So et al. 1996; Yousefzadeh et al. 2008]). The typical software based approach (e.g. see [So et al. 1996]) is to control the switch u in Fig. 3 (typically implemented with a MOSFET) with a microcontroller.

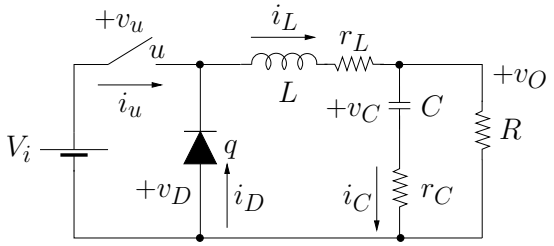


Fig. 3. Buck DC-DC converter.

Designing the software to run on the microcontroller to properly actuate the switch is the control software design problem for the buck DC-DC converter in our context.

The circuit in Fig. 3 can be modeled as a DTLHS $\mathcal{H} = (X, U, Y, N)$ in the following way. As for the sets of variables, we have $X = X^r = [i_L, v_O]$, $U = U^b = [u]$, $Y = Y^r \cup Y^b$ with $Y^r = [i_u, v_u, i_D, v_D]$ and $Y^b = [q]$. As for N, it is given by the conjunction of the following (guarded) constraints:

$$i_L{}' = (1 + Ta_{1,1})i_L + Ta_{1,2}v_O + Ta_{1,3}v_D \quad (1)$$

$$v_O{}' = Ta_{2,1}i_L + (1 + Ta_{2,2})v_O + Ta_{2,3}v_D. \quad (2)$$

$$q \rightarrow v_D = 0 \quad (3) \qquad i_D = i_L - i_u \quad (6) \qquad \bar{u} \rightarrow v_u = R_{off} i_u \quad (9)$$
$$q \rightarrow i_D \geq 0 \quad (4) \qquad \bar{q} \rightarrow v_D \leq 0 \quad (7) \qquad v_D = v_u - V_i \quad (10)$$
$$u \rightarrow v_u = 0 \quad (5) \qquad \bar{q} \rightarrow v_D = R_{off} i_D \quad (8)$$

where the coefficients $a_{i,j}$ depend on the circuit parameters R, r_L, r_C, L and C in the following way: $a_{1,1} = -\frac{r_L}{L}$, $a_{1,2} = -\frac{1}{L}$, $a_{1,3} = -\frac{1}{L}$, $a_{2,1} = \frac{R}{r_c+R}[-\frac{r_c r_L}{L} + \frac{1}{C}]$, $a_{2,2} = \frac{-1}{r_c+R}[\frac{r_c R}{L} + \frac{1}{C}]$, $a_{2,3} = -\frac{1}{L}\frac{r_c R}{r_c+R}$.

4. QUANTIZED FEEDBACK CONTROL

In this section, we formally define the Quantized Feedback Control Problem for DTLHSs (Sect. 4.3). To this end, first we give the definition of Feedback Control Problem for LTSs (Sect. 4.1), and then for DTLHSs (Sect. 4.2). Finally, we show that our definitions are well founded (Sect. 4.4).

4.1. Feedback Control Problem for LTSs

We begin by extending to possibly infinite LTSs the definitions in [Tronci 1998; Cimatti et al. 1998] for finite LTSs. In what follows, let $\mathcal{S} = (S, A, T)$ be an LTS, and $I, G \subseteq S$ be, respectively, the *initial* and *goal* regions.

Definition 4.1 (*LTS control problem*). A *controller* for an LTS $\mathcal{S}$ is a function $K : S \times A \rightarrow \mathbb{B}$ such that $\forall s \in S$, $\forall a \in A$, if $K(s,a)$ then $a \in \mathrm{Adm}(\mathcal{S}, s)$. We denote with $\mathrm{Dom}(K)$ the set of states for which a control action is defined. Formally, $\mathrm{Dom}(K) = \{s \in S \mid \exists a : K(s,a)\}$. $\mathcal{S}^{(K)}$ denotes the *closed loop system*, that is the LTS $(S, A, T^{(K)})$, where $T^{(K)}(s,a,s') = T(s,a,s') \wedge K(s,a)$. A *control law* for a controller K is a (partial) function $k : S \rightarrow A$ s.t. for all $s \in \mathrm{Dom}(K)$ we have that $K(s, k(s))$ holds. By abuse of language we say that a controller is a control law if for all $s \in S$, $a, b \in A$ it holds that $(K(x,a) \wedge K(x,b)) \rightarrow (a = b)$. An *LTS control problem* is a triple $(\mathcal{S}, I, G)$.

Example 4.2. Let $S = \{-1, 0, 1\}$ and $A = \{0, 1\}$. Let $\mathcal{S}_0$ be the LTS (S, A, T_0), where the transition relation T_0 consists of the continuous arrows in Fig. 4. A function K is a controller for $\mathcal{S}_0$ iff $(s \neq 0) \rightarrow (K(s,1) = 0)$. As an example, we have that K defined as $K(s,a) = ((s \neq 0) \rightarrow (a \neq 1))$ is a controller but not a control law, and that $k(s) = 0$ is a control law for K (note that $K(s,a) = (a = 0)$ is a control law).

Def. 4.1 also introduces the formal definition of *control law*, as our model of control software, i.e. of how function `Control_Law` in Fig. 1 must behave. Namely, while a controller may enable many actions in a given state, a control law (i.e. the final software implementation) must provide only one action. Note that the notion of controller is important because it contains all possible control laws.

In the following we give formal definitions of strong and weak solutions to a control problem for an LTS.

We call a path π *fullpath* if either it is infinite or its last state $\pi^{(S)}(|\pi|)$ has no successors (i.e. $\mathrm{Adm}(\mathcal{S}, \pi^{(S)}(|\pi|)) = \varnothing$). We denote with $\mathrm{Path}(\mathcal{S}, s, a)$ the set of fullpaths of $\mathcal{S}$ starting in state s with action a, i.e. the set of fullpaths π such that $\pi^{(S)}(0) = s$ and $\pi^{(A)}(0) = a$.

Given a path π in $\mathcal{S}$, we define the measure $J(\mathcal{S}, G, \pi)$ on paths as the distance of $\pi^{(S)}(0)$ to the goal on π. That is, if there exists $n > 0$ s.t. $\pi^{(S)}(n) \in G$, then $J(\mathcal{S}, G, \pi) = \min\{n \mid n > 0 \wedge \pi^{(S)}(n) \in G\}$. Otherwise, $J(\mathcal{S}, G, \pi) = +\infty$. We require $n > 0$ since our systems are nonterminating and each controllable state (including a goal state) must have a path of positive length to a goal state. Taking $\sup \varnothing = +\infty$ and $\inf \varnothing = -\infty$, the *worst case distance* (pessimistic view) of a state s from the goal region G is $J_{\mathrm{strong}}(\mathcal{S}, G, s) = \sup\{J^{(S)}(\mathcal{S}, G, s, a) \mid a \in \mathrm{Adm}(\mathcal{S}, s)\}$, where: $J^{(S)}(\mathcal{S}, G, s, a) =$

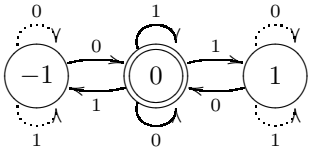


Fig. 4. LTSs $\mathcal{S}_0$ (continuous arrows) and $\mathcal{S}_1$ (all arrows). Double circle represents the goal state.

$\sup\{J(\mathcal{S}, G, \pi) \mid \pi \in \mathrm{Path}(\mathcal{S}, s, a)\}$. The *best case distance* (optimistic view) of a state s from the goal region G is $J_{\mathrm{weak}}(\mathcal{S}, G, s) = \sup\{J^{(W)}(\mathcal{S}, G, s, a) \mid a \in \mathrm{Adm}(\mathcal{S}, s)\}$, where: $J^{(W)}(\mathcal{S}, G, s, a) = \inf\{J(\mathcal{S}, G, \pi) \mid \pi \in \mathrm{Path}(\mathcal{S}, s, a)\}$.

Definition 4.3 (*Solution to LTS control problem*). Let $\mathcal{P} = (\mathcal{S}, I, G)$ be an LTS control problem and K be a controller for $\mathcal{S}$ such that $I \subseteq \mathrm{Dom}(K)$. K is a *strong [weak] solution* to $\mathcal{P}$ if for all $s \in \mathrm{Dom}(K)$, $J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, s)$ $[J_{\mathrm{weak}}(\mathcal{S}^{(K)}, G, s)]$ is finite. An *optimal strong [weak] solution* to $\mathcal{P}$ is a strong [weak] solution K^* to $\mathcal{P}$ such that for all strong [weak] solutions K to $\mathcal{P}$, for all $s \in S$ we have that $J_{\mathrm{strong}}(\mathcal{S}^{(K^*)}, G, s) \leq J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, s)$ $[J_{\mathrm{weak}}(\mathcal{S}^{(K^*)}, G, s) \leq J_{\mathrm{weak}}(\mathcal{S}^{(K)}, G, s)]$.

Intuitively, a strong solution K takes a *pessimistic* view by requiring that for each initial state, *all* runs in the closed loop system $\mathcal{S}^{(K)}$ reach the goal, no matter nondeterministic outcomes. A weak solution K takes an *optimistic* view about nondeterminism: it just asks that for each action a enabled in a given state s, there exists at least a path in $\mathrm{Path}(\mathcal{S}^{(K)}, s, a)$ leading to the goal. Unless otherwise stated, we say *solution* for *strong solution*.

Finally, we define the *most general optimal strong [weak] solution* to $\mathcal{P}$ (*strong [weak] mgo* in the following) as the unique strong [weak] optimal solution to $\mathcal{P}$ enabling as many actions as possible (i.e., the most liberal one). In Sect. 4.4 we show that the definition of mgo is well posed.

Example 4.4. Let $\mathcal{S}_0, \mathcal{S}_1$ be the LTSs in Fig. 4 (see also Ex. 4.2). Let $\mathcal{P}_0 = (\mathcal{S}_0, I, G)$ and $\mathcal{P}_1 = (\mathcal{S}_1, I, G)$ be two control problems, where $I = \{-1, 0, 1\}$ and $G = \{0\}$. The controller $K(s, a) \equiv [s \neq 0 \rightarrow a = 0]$ is a strong solution to the control problem $\mathcal{P}_0$. Observe that K is not optimal. Indeed, the controller $\tilde{K}(s, a) \equiv a = 0$ is such that $J_{\mathrm{strong}}(\mathcal{S}_0^{(\tilde{K})}, G, 0) = 1 < 2 = J_{\mathrm{strong}}(\mathcal{S}_0^{(K)}, G, 0)$. The control problem $\mathcal{P}_1$ has no strong solution. As a matter of fact, to drive the system to the goal region $\{0\}$, any solution K must enable action 0 in states -1 and 1: in such a case, however, we have that $J_{\mathrm{strong}}(\mathcal{S}_1^{(K)}, \hat{G}, 1) = J_{\mathrm{strong}}(\mathcal{S}_1^{(K)}, \hat{G}, -1) = \infty$ because of the self loops $(1, 0, 1)$ and $(-1, 0, -1)$ of T_1. Finally, note that K is the weak mgo for $\mathcal{P}_1$ and $\tilde{K}$ is the strong mgo for $\mathcal{P}_0$.

Remark 4.5. Note that if K is a strong solution to $(\mathcal{S}, I, G)$ and $G \subseteq I$ (as is usually the case in control problems) then $\mathcal{S}^{(K)}$ is *stable* from I to G, that is each run in $\mathcal{S}^{(K)}$ starting from a state in I leads to a state in G. In fact, from Def. 4.3 we have that each state $s \in I$ reaches a state $s' \in G$ in a finite number of steps. Moreover, since $G \subseteq I$, we have that any state $s \in G$ reaches a state $s' \in G$ in a finite number of steps. Thus, any path starting in I in the closed loop system $\mathcal{S}^{(K)}$ *touches* G an infinite number of times (*liveness*).

4.2. Feedback Control Problem for DTLHSs

A control problem for a DTLHS $\mathcal{H}$ is the LTS control problem induced by the dynamics of $\mathcal{H}$. For DTLHSs, we only consider control problems where I and G can be represented as predicates over present state variables of $\mathcal{H}$.

Definition 4.6 (*DTLHS control problem*). Given a DTLHS $\mathcal{H} = (X, U, Y, N)$ and predicates I and G over X, the *DTLHS (feedback) control problem* $(\mathcal{H}, I, G)$ is the LTS control problem $(LTS(\mathcal{H}), I, G)$. Thus, a controller $K : \mathcal{D}_X \times \mathcal{D}_U \to \mathbb{B}$ is a *strong [weak] solution* to $(\mathcal{H}, I, G)$ iff it is a strong [weak] solution to $(LTS(\mathcal{H}), I, G)$.

For DTLHS control problems, usually *robust* controllers are desired. That is, controllers that, notwithstanding nondeterminism in the plant (e.g. due to parameter variations, see Ex. 3.2), drive the plant state to the goal region. For this reason we focus on strong solutions.

Observe that the feedback controller for a DTLHS will only measure present state variables (e.g., output voltage and inductor current in Sect. 3.1) and will not measure auxiliary variables (e.g. diode state in Sect. 3.1).

Example 4.7. The typical goal of a controller for the buck DC-DC converter in Sect. 3.1 is keeping the output voltage v_O *close enough* to a given reference value V_{ref}. This leads to the DTLHS control problem $\mathcal{P} = (\mathcal{H}, I, G)$ where $\mathcal{H}$ is defined in Sect. 3.1, $I \equiv (|i_L| \leq 2) \wedge (0 \leq v_O \leq 6.5)$, $G \equiv (|v_O - V_{\text{ref}}| \leq \theta) \wedge (|i_L| \leq 2)$, and $\theta = 0.01$ is the desired buck precision.

4.3. Quantized Feedback Control Problem

Software running on a microcontroller (*control software* in the following) cannot handle real values. For this reason real valued state feedback from plant sensors undergoes an *Analog-to-Digital* (AD) conversion before being sent to the control software. This process is called *quantization* (e.g. see [Fu and Xie 2005] and citations thereof). A *Digital-to-Analog* (DA) conversion is needed to transform the control software digital output into real values to be sent to plant actuators. In the following, we formally define quantized solutions to a DTLHS feedback control problem.

Definition 4.8 (*Quantization function*). A *quantization function* γ for a real interval $I = [a, b]$ is a non-decreasing function $\gamma : [a, b] \to \hat{I}$, where $\hat{I}$ is a bounded integer interval $[\gamma(a), \gamma(b)] \subseteq \mathbb{Z}$. The *quantization step* of γ, denoted by $\|\gamma\|$, is defined as $\sup\{\, |w-z| \mid w, z \in I \wedge \gamma(w) = \gamma(z)\}$.

For ease of notation, we extend quantizations to integer intervals, by stipulating that in such a case the quantization function is the identity function (i.e. $\gamma(x) = x$). Note that, with this convention, the quantization step on an integer interval is always 0.

Definition 4.9 (*Quantization for DTLHSs*). Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS, and let $W = X \cup U$. A *quantization* $\mathcal{Q}$ for $\mathcal{H}$ is a pair (A, Γ), where:

— A is a predicate of form $\wedge_{w \in W}(a_w \leq w \leq b_w)$ with $a_w, b_w \in \mathcal{D}_w$. For each $w \in W$, we define $A_w = \{v \in \mathcal{D}_w \mid a_w \leq v \leq b_w\}$ as the *admissible region* for variable w. Moreover, we define $A_V = \prod_{v \in V} A_v$, with $V \subseteq W$, as the admissible region for variables in V.
— Γ is a set of maps $\Gamma = \{\gamma_w \mid w \in W$ and γ_w is a quantization function for $A_w\}$.

Let $V = [w_1, \ldots, w_k]$ and $v = [v_1, \ldots, v_k] \in A_V$, where $V \subseteq W$. We write $\Gamma(v)$ (or $\hat{v}$) for the tuple $[\gamma_{w_1}(v_1), \ldots, \gamma_{w_k}(v_k)]$ and $\Gamma^{-1}(\hat{v})$ for the set $\{v \in A_V \mid \Gamma(v) = \hat{v}\}$. Finally, the *quantization step* $\|\Gamma\|$ for Γ is defined as $\sup\{\, \|\gamma\| \mid \gamma \in \Gamma\}$.

For ease of notation, in the following we will also consider quantizations for primed variables $x' \in X'$, by stipulating that $\gamma_{x'} \equiv \gamma_x$.

Example 4.10. Let $\mathcal{H}$ be the DTLHS described in Ex. 3.2. Let us consider the quantization $\mathcal{Q} = (A, \Gamma)$, where $A \equiv -2.5 \leq x \leq 2.5 \wedge 0 \leq u \leq 1$. A defines the admissible region $A_x = A_X = [-2.5, 2.5]$. Let $\Gamma = \{\gamma_x, \gamma_u\}$, with $\gamma_x(x) = round(x/2)$ (where $round(x) = \lfloor x \rfloor + \lfloor 2(x - \lfloor x \rfloor) \rfloor$ is the usual rounding function) and $\gamma_u(u) = u$. Note that

$\gamma_x(x) = -1$ for all $x \in [-2.5, -1]$, $\gamma_x(x) = 0$ for all $x \in (-1, 1)$ and $\gamma_x(x) = 1$ for all $x \in [1, 2.5]$. Thus, we have that $\Gamma(A_x) = \{-1, 0, 1\}$, $\Gamma(A_u) = \{0, 1\}$ and $\|\Gamma\| = 1$.

Quantization, i.e. representing reals with integers, unavoidably introduces errors in reading real-valued plant sensors in the control software. We address this problem in the following way. First, we introduce the definition of ε-solution. Essentially, we require that the controller drives the plant "near enough" (up to a given error ε) to the goal region G.

Definition 4.11 (*ε-relaxation of a set*). Let $\varepsilon \geq 0$ be a real number and $W \subseteq \mathbb{R}^n \times \mathbb{Z}^m$. The *$\varepsilon$-relaxation* of W is the set (*ball* of radius ε) $\mathcal{B}_\varepsilon(W) = \{(z_1, \ldots z_n, q_1, \ldots q_m) \mid \exists (x_1, \ldots, x_n, q_1, \ldots, q_m) : (x_1, \ldots, x_n, q_1, \ldots, q_m) \in W$ and $\forall i \in \{1, \ldots n\}\ |z_i - x_i| \leq \varepsilon\}$.

Definition 4.12 (*ε-solution to DTLHS control problem*). Let $\mathcal{P} = (\mathcal{H}, I, G)$ be a DTLHS control problem and let $\varepsilon > 0$ be a real number. A *strong [weak] ε-solution* to $\mathcal{P}$ is a strong [weak] solution to the LTS control problem $(LTS(\mathcal{H}), I, \mathcal{B}_\varepsilon(G))$.

Example 4.13. Let $\mathcal{H}$ be the DTLHS described in Ex. 3.2. We consider the control problem defined by the initial region $I = [-2.5, 2.5]$ and the goal region $G = \{0\}$ (represented by the predicate $x = 0$). The DTLHS control problem $\mathcal{P} = (\mathcal{H}, I, G)$ has no solution (because of the Zeno phenomenon), but for all $\varepsilon > 0$ it has the ε-solution K such that $\forall x \in I.\ K(x, 0)$.

Second, we introduce the definition of *quantized solution* to a DTLHS control problem for a given quantization $\mathcal{Q} = (A, \Gamma)$. Essentially, a quantized solution models the fact that in an SBCS control decisions are taken by the control software by just looking at quantized state values. Despite this, a quantized solution guarantees that each DTLHS initial state reaches a DTLHS goal state (up to an error at most $\|\Gamma\|$).

Definition 4.14 (*Quantized Feedback Control solution to DTLHS control problem*). Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$ and $\mathcal{P} = (\mathcal{H}, I, G)$ be a DTLHS control problem. A $\mathcal{Q}$ *Quantized Feedback Control* (QFC) *strong [weak] solution* to $\mathcal{P}$ is a strong [weak] $\|\Gamma\|$-solution $K : \mathcal{D}_X \times \mathcal{D}_U \to \mathbb{B}$ to $\mathcal{P}$ such that $K(x, u) = 0$ if $(x, u) \notin A_X \times A_U$, and otherwise $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ where $\hat{K} : \Gamma(A_X) \times \Gamma(A_U) \to \mathbb{B}$.

Note that a $\mathcal{Q}$ QFC solution to a DTLHS control problem does not work outside the admissible region defined by $\mathcal{Q}$. This models the fact that controllers for real-world systems must maintain the plant inside given bounds (such requirements are part of the safety specifications). In the following, we will define $\mathcal{Q}$ QFC solutions by only specifying their behavior inside the admissible region.

Example 4.15. Let $\mathcal{P}$ be the DTLHS control problem defined in Ex. 4.13 and $\mathcal{Q} = (A, \Gamma)$ be the quantization defined in Ex. 4.10. Let $\hat{K}$ be defined by $\hat{K}(\hat{x}, \hat{u}) \equiv [\hat{x} \neq 0 \to \hat{u} = 0]$. For any $\varepsilon > 0$, the quantized controller $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is an ε-solution to $\mathcal{P}$, and hence it is a $\mathcal{Q}$ QFC solution.

Along the same lines of similar undecidability proofs [Henzinger et al. 1998; Agrawal et al. 2006], it is possible to show that existence of a $\mathcal{Q}$ QFC solution to a DTLHS control problem (*DTLHS quantized control problem*) is undecidable, as shown in [Mari et al. 2012c].

THEOREM 4.16. *The DTLHS quantized control problem is undecidable.*

4.4. Proof of Uniqueness of the Most General Optimal Controller

In this section, we prove properties on mgo (see Sect. 4.1). This section can be skipped at a first reading. We begin by giving the formal definition of strong and weak mgo.

Definition 4.17 (*Most general optimal solution to control problem*). The *most general optimal strong [weak] solution* to $\mathcal{P}$ is an optimal strong [weak] solution $\tilde{K}$ to $\mathcal{P}$ such that for all other optimal strong [weak] solutions K to $\mathcal{P}$, for all $s \in S$, for all $a \in A$ we have that $K(s,a) \to \tilde{K}(s,a)$.

PROPOSITION 4.18. *An LTS control problem* $(\mathcal{S}, \varnothing, G)$ *has always an* unique *strong mgo* K^*. *Moreover, for all* $I \subseteq S$, *we have:*

— *if* $I \subseteq \mathrm{Dom}(K^*)$, *then* K^* *is the unique strong mgo for the control problem* $(\mathcal{S}, I, G)$;
— *if* $I \not\subseteq \mathrm{Dom}(K^*)$, *then the control problem* $(\mathcal{S}, I, G)$ *has no strong solution.*

PROOF. Let $\mathcal{S} = (S, A, T)$ be an LTS, and let $(\mathcal{S}, I, G)$ be an LTS control problem. We define the sequences of sets D_n and F_n as follows:

— $D_0 = \varnothing$
— $F_1 = \{s \in S \mid \exists a \in A : a \in \mathrm{Adm}(\mathcal{S}, s) \wedge \mathrm{Img}(\mathcal{S}, s, a) \subseteq G\}$
— $F_{n+1} = \{s \in S \setminus D_n \mid \exists a \in A : a \in \mathrm{Adm}(\mathcal{S}, s) \wedge \mathrm{Img}(\mathcal{S}, s, a) \subseteq D_n\}$
— $D_{n+1} = D_n \cup F_{n+1}$

Intuitively, D_n is the set of states which can be driven inside G in at most n steps, notwithstanding nondeterminism. F_n is the subset of D_n containing only those states for which at least a path to G of length exactly n exists.

The following properties hold for D_n and F_n:

(1) If $F_n = \varnothing$ for some $n \geq 1$, then for all $m \geq n$, $F_m = \varnothing$. In fact, if $F_n = \varnothing$, then $D_n = D_{n-1}$, and hence $F_{n+1} = F_n = \varnothing$.
(2) If $D_{n+1} = D_n$ for some $n \geq 0$, then for all $m \geq n$, $D_m = D_n$. This immediately follows from the previous point 1.
(3) $D_n = \bigcup_{1 \leq j \leq n} F_j$ for $n \geq 1$ (also for $n \geq 0$ if we take the union of no sets to be $\varnothing$). We prove this property by induction on n. As for the induction base, we have that $D_1 = F_1$. As for the inductive step, $D_{n+1} = D_n \cup F_{n+1} = \bigcup_{1 \leq j \leq n} F_j \cup F_{n+1} = \bigcup_{1 \leq j \leq n+1} F_j$.
(4) $F_i \cap F_j = \varnothing$ for all $i \neq j$. We have that if $s \in F_{n+1}$ then $s \notin D_n$. By previous point 3, we have that $s \notin D_n$ implies $s \notin F_j$ for $1 \leq j \leq n$. Hence, $s \in F_{n+1}$ implies that $s \notin F_j$ for all $1 \leq j \leq n$. If by absurd a state s exists s.t. $s \in F_i \cap F_j$ for some $i > j$, then $s \in F_i$ would imply $s \notin F_j$.

For all $s \in S$ and $a \in A$, we define the controller $\tilde{K} : S \times A \to \mathbb{B}$ as follows: $\tilde{K}(s,a) \Leftrightarrow (\exists n > 1 : s \in F_n \wedge a \in \mathrm{Adm}(\mathcal{S}, s) \wedge \mathrm{Img}(\mathcal{S}, s, a) \subseteq D_{n-1}) \vee (s \in F_1 \wedge a \in \mathrm{Adm}(\mathcal{S}, s) \wedge \mathrm{Img}(\mathcal{S}, s, a) \subseteq G)$.

Note that $\mathrm{Dom}(\tilde{K}) = \overline{D} = \bigsqcup_{n \in \mathbb{N}} D_n$, i.e. the domain of $\tilde{K}$ is the least upper bound for sets D_n (we are not supposing S to be finite, thus there may be a nonempty D_n for any $n \in \mathbb{N}$).

$\tilde{K}$ is a strong solution to $(\mathcal{S}, \varnothing, G)$. To prove this, we show that, if $t \in F_n$, then $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, t) = n$ (note that $t \in \mathrm{Dom}(\tilde{K})$ implies $t \in F_n$ for some $n \geq 1$). In fact, if $t \in F_1$ then $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, t) = \sup\{J^{(S)}(\mathcal{S}^{(\tilde{K})}, G, t, a) \mid a \in \mathrm{Adm}(\mathcal{S}^{(\tilde{K})}, t)\} = \sup\{J^{(S)}(\mathcal{S}^{(\tilde{K})}, G, t, a) \mid a \text{ is s.t. } \varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq G\} = \sup\{\sup\{J(\mathcal{S}^{(\tilde{K})}, G, \pi) \mid \pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a)\} \mid a \text{ is s.t. } \varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq G\} = \sup\{J(\mathcal{S}^{(\tilde{K})}, G, \pi) \mid \pi \in \{\pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a) \mid a \text{ is s.t. } \varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq G\}\} = \sup\{\min\{n \mid n > 0 \wedge \pi^{(S)}(n) \in G\} \mid \pi \in \{\pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a) \mid a \text{ is s.t. } \varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq G\}\}$. Since for all $\pi \in \{\pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a) \mid a \text{ is s.t. } \varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq G\}$ we have that $\pi^{(S)}(1) \in G$, we finally have that $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, t) = \sup\{1\} = 1$. On the other hand, if $t \in F_n$ then $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, t) = \sup\{\min\{n \mid n > 0 \wedge \pi^{(S)}(n) \in G\} \mid \pi \in \{\pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a) \mid a$

is s.t. $\varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq D_{n-1}\}\} = \sup\{n_1, \ldots, n_j, \ldots\}$. We have that, for all j, $n_j \leq n$. In fact, being $t \in F_n$ and a s.t. $\varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq D_{n-1}$, we have that $\pi^{(S)}(1) \in D_{n-1}$ for all paths $\pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a)$. This implies that $\pi^{(S)}(1) \in D_{n-2} \vee \pi^{(S)}(1) \in F_{n-1}$. By property 3 above, this implies that there exists $1 \leq i \leq n-1$ s.t. $\pi^{(S)}(1) \in F_i$. By iterating $n-1$ times such a reasoning, we obtain that there exists $1 \leq i \leq n$ s.t. $\pi^{(S)}(i) \in G$, which implies $n_j \leq n$ for all j. Moreover, there exists a path $\pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a)$ s.t. $\pi^{(S)}(n) \in G$ and for all $0 < i < n$ we have that $\pi^{(S)}(i) \notin G$. Suppose by absurd that for all paths $\pi \in \mathrm{Path}(\mathcal{S}^{(\tilde{K})}, t, a)$ we have that, if for all $0 < i < n$ $\pi^{(S)}(i) \notin G$, then $\pi^{(S)}(n) \notin G$. By using an iterative reasoning as above, it is possible to show that this contradicts t being in F_n and a being s.t. $\varnothing \neq \mathrm{Img}(\mathcal{S}, t, a) \subseteq D_{n-1}$. Thus, being $n_j \leq n$ for all j and existing a j s.t. $n_j = n$, we have that $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, t) = \sup\{n_1, \ldots, n_j, \ldots\} = n$.

Note that also the converse holds, i.e. $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, t) = n$ implies $t \in F_n$. This can be proved analogously to the reasoning above.

To prove that $\tilde{K}$ is optimal, let us suppose that there exists another solution K and that there exists a nonempty set Z of states, such that for all $z \in Z$, $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, z) > J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, z)$. Let $z_0 \in Z$ be a state for which $J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, z_0) = n$ is minimal in Z, and let $a \in A$ be such that $K(z_0, a)$.

We have that $n = 1$ implies that $\mathrm{Img}(\mathcal{S}, z_0, a) \subseteq G$. But in such a case, z_0 would belong to F_1, and hence $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, z_0) = 1 = J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, z_0)$.

If $n > 1$, for all $s \in \mathrm{Img}(\mathcal{S}, z_0, a)$, we have that $J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, s) \leq n-1$. Since n is the minimal distance for which $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, z) > J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, z) = n$, we have that for all $s \in \mathrm{Img}(\mathcal{S}, z_0, a)$, $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, s) \leq J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, s) \leq n-1$. This implies that, $J_{\mathrm{strong}}(\mathcal{S}^{(\tilde{K})}, G, z_0) \leq n$, which is absurd.

To prove that $\tilde{K}$ is the most general optimal solution, we proceed in a similar way. Let us suppose that there exists another optimal solution K and that there exists a nonempty set Z of states, such that for all $z \in Z$ there exists an action a s.t. $K(z, a)$ and $\neg\tilde{K}(z, a)$ holds. Let $z_0 \in Z$ be a state for which $J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, z_0) = n$ is minimal in Z.

If $n = 1$ we have that $\mathrm{Img}(\mathcal{S}, z_0, a) \subseteq G$ and thus $z_0 \in F_1$ and $\tilde{K}(z_0, a)$, which leads to a contradiction.

If $n > 1$, by minimality of $J_{\mathrm{strong}}(\mathcal{S}^{(K)}, G, z_0)$ in Z we have that, for all $s \in \mathrm{Img}(\mathcal{S}, z_0, a)$, $K(s, u)$ implies $\tilde{K}(s, u)$. This implies that $\mathrm{Img}(\mathcal{S}, z_0, a) \in D_{n-1}$ and thus $\tilde{K}(z_0, a)$ holds. □

5. CONTROL ABSTRACTION

A quantization naturally induces an abstraction of a DTLHS. Motivated by finding QFC solutions in the abstract model, in this paper we introduce a novel notion of abstraction, namely *control abstraction*. In what follows we introduce the notion of control abstraction. In Sect. 5.1 we discuss on minimum and maximum control abstractions. In Sect. 5.2 we give some properties on control abstractions.

Control abstraction (Def. 5.3) models how a DTLHS $\mathcal{H}$ is *seen* from the control software after AD conversions. Since QFC control rests on AD conversion we must be careful not to drive the plant outside the bounds in which AD conversion works correctly. This leads to the definition of *admissible action* (Def. 5.1). Intuitively, an action is admissible in a state if it never drives the system outside of its admissible region.

Definition 5.1 (*Admissible actions*). Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS and $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$. An action $u \in A_U$ is *A-admissible* in $s \in A_X$ if for all s', $(\exists y \in A_Y : N(s, u, y, s'))$ implies $s' \in A_X$. An action $\hat{u} \in \Gamma(A_U)$ is *$\mathcal{Q}$-admissible* in $\hat{s} \in \Gamma(A_X)$ if for all $s \in \Gamma^{-1}(\hat{s})$, $u \in \Gamma^{-1}(\hat{u})$, u is A-admissible for s in $\mathcal{H}$.

Example 5.2. Let $\mathcal{H}$ be as in Ex. 3.2 and $\mathcal{Q}$ as in Ex. 4.10. We have that action $u = 1$ is not A-admissible in the state $s = 2$, thus $\hat{u} = 1$ is not $\mathcal{Q}$-admissible in the state $\hat{s} = 1$. Analogously, $\hat{u} = 1$ is not $\mathcal{Q}$-admissible in $\hat{s} = -1$. It is easy to see that no other $\hat{u}, \hat{s}$ exist s.t. $\hat{u}$ is not $\mathcal{Q}$-admissible in $\hat{s}$.

Definition 5.3 (*Control abstraction*). Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS and $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$. We say that the LTS $\hat{\mathcal{H}} = (\Gamma(A_X),\ \Gamma(A_U),\ \hat{N})$ is a $\mathcal{Q}$ *control abstraction* of $\mathcal{H}$ if its transition relation $\hat{N}$ satisfies the following conditions:

(1) Each abstract transition stems from a concrete transition. Formally: for all $\hat{s}, \hat{s}' \in \Gamma(A_X)$, $\hat{u} \in \Gamma(A_U)$, if $\hat{N}(\hat{s}, \hat{u}, \hat{s}')$ then there exist $s \in \Gamma^{-1}(\hat{s})$, $u \in \Gamma^{-1}(\hat{u})$, $s' \in \Gamma^{-1}(\hat{s}')$, $y \in A_Y$ such that $N(s, u, y, s')$.
(2) Each concrete transition is faithfully represented by an abstract transition, whenever it is not a self loop and its corresponding abstract action is $\mathcal{Q}$-admissible. Formally: for all $s, s' \in A_X$, $u \in A_U$ such that $\exists y : N(s, u, y, s')$, if $\Gamma(u)$ is $\mathcal{Q}$-admissible in $\Gamma(s)$ and $\Gamma(s) \neq \Gamma(s')$ then $\hat{N}(\Gamma(s), \Gamma(u), \Gamma(s'))$.
(3) If there is no upper bound to the length of concrete paths inside the counter-image of an abstract state then there is an abstract self loop. Formally: for all $\hat{s} \in \Gamma(A_X)$, $\hat{u} \in \Gamma(A_U)$, if it exists an infinite run π in $\mathcal{H}$ such that $\forall t \in \mathbb{N}\ \pi^{(S)}(t) \in \Gamma^{-1}(\hat{s})$ and $\pi^{(A)}(t) \in \Gamma^{-1}(\hat{u})$ then $\hat{N}(\hat{s}, \hat{u}, \hat{s})$. A self loop $(\hat{s}, \hat{u}, \hat{s})$ of $\hat{N}$ satisfying the above property is said to be a *non-eliminable self loop*, and *eliminable self loop* otherwise.

Example 5.4. Let $\mathcal{H}$ be as in Ex. 3.2 and $\mathcal{Q}$ be as in Ex. 4.10. Any $\mathcal{Q}$ control abstraction $\hat{\mathcal{H}}$ of $\mathcal{H}$ has the form $(\{-1, 0, 1\}, \{0, 1\}, \hat{N})$ where $\hat{N}$ always contains at least all continuous arrows in the automaton depicted in Fig. 4 and some dotted arrows. Note that the only non-eliminable self loops are $(0, 0, 0)$ and $(0, 1, 0)$.

Along the same lines of the proof for Theor. 4.16, in [Mari et al. 2012c] we proved that we cannot algorithmically decide if a self loop is eliminable or non-eliminable.

Proposition 5.5. *Given a DTLHS $\mathcal{H}$ and a quantization $\mathcal{Q}$, it is undecidable to determine if a self loop is non-eliminable.*

Note that if in Def. 5.3 we drop condition 3 and the guard $\Gamma(s) \neq \Gamma(s')$ in condition 2, then we essentially get the usual definition of *abstraction* (e.g. see [Alur et al. 2006] and citations thereof). As a result, any abstraction is also a control abstraction whereas a control abstraction in general is not an abstraction since some self loops or some non admissible actions may be missing.

In the following, we will deal with two types of control abstractions, namely *full* and *admissible* control abstractions, which are defined as follows.

Definition 5.6 (*Admissible and full control abstractions*). Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS and $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$. A $\mathcal{Q}$ control abstraction $\hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$ of $\mathcal{H}$ is an *admissible* $\mathcal{Q}$ *control abstraction* iff, for all $\hat{s} \in \Gamma(A_X), \hat{u} \in \Gamma(A_U)$ s.t. $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}, \hat{s})$: i) $\hat{u}$ is $\mathcal{Q}$-admissible in $\hat{s}$; ii) $\forall s \in \Gamma^{-1}(\hat{s})\ \forall u \in \Gamma^{-1}(\hat{u})\ \exists s' \in \mathcal{D}_X\ \exists y \in \mathcal{D}_Y : N(s, u, y, s')$, i.e. each concrete state in $\Gamma^{-1}(\hat{s})$ has a successor for all concrete actions in $\Gamma^{-1}(\hat{u})$.

We say that $\hat{\mathcal{H}}$ is a *full* $\mathcal{Q}$ *control abstraction* if it satisfies properties 1 and 3 of Def. 5.3, plus the following property (derived from property 2 of Def. 5.3): for all $s, s' \in A_X$, $u \in A_U$ such that $\exists y : N(s, u, y, s')$, if $\Gamma(s) \neq \Gamma(s')$ then $\hat{N}(\Gamma(s), \Gamma(u), \Gamma(s'))$.

Example 5.7. Let $\mathcal{H}$ be as in Ex. 3.2, $\mathcal{Q}$ be as in Ex. 4.10. For all $\mathcal{Q}$ admissible control abstractions of $\mathcal{H}$, $\hat{N}(1, 1, 1) = \hat{N}(-1, 1, -1) = 0$, since action 1 is not $\mathcal{Q}$-admissible either

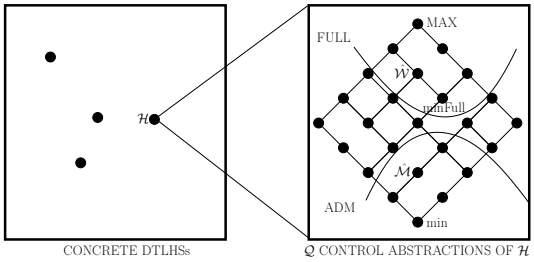


Fig. 5. Lattices on $\mathcal{Q}$ control abstractions.

in -1 or in 1 (see Ex. 5.2). On the contrary, for all full $\mathcal{Q}$ control abstractions of $\mathcal{H}$, $\hat{N}(1,1,1) = \hat{N}(-1,1,-1) = 1$. Thus, a control abstraction s.t. $\hat{N}(1,1,1) \oplus \hat{N}(-1,1,-1)$ (where $\oplus$ is the logical XOR) is neither full nor admissible.

By the definition of quantization, a control abstraction is a finite LTS. It is possible to show that two different admissible [full] $\mathcal{Q}$ control abstractions only differ in the number of self loops. Moreover, the set of admissible [full] $\mathcal{Q}$ control abstraction is a finite lattice with respect to the LTS refinement relation (Sect. 5.2). This implies that such lattices have minimum (and maximum). Thus, it is easy to prove that the minimum admissible [full] $\mathcal{Q}$ control abstraction is the admissible [full] $\mathcal{Q}$ control abstraction with non-eliminable self loops only. Thus, the following proposition is a corollary of Prop. 5.5.

PROPOSITION 5.8. *Given a DTLHS $\mathcal{H}$ and a quantization $\mathcal{Q}$, it is undecidable to state if an admissible [full] $\mathcal{Q}$ control abstraction for $\mathcal{H}$ is the minimum admissible [full] $\mathcal{Q}$ control abstraction for $\mathcal{H}$.*

5.1. Maximum and Minimum Control Abstractions

By Theor. 4.16, we cannot hope for a constructive sufficient and necessary condition for the existence of a $\mathcal{Q}$ QFC solution to a DTLHS control problem, for a given $\mathcal{Q}$. Accordingly, our approach is able to determine (via a sufficient condition) if a $\mathcal{Q}$ QFC solution exists, and otherwise to state (via a necessary condition) if a $\mathcal{Q}$ QFC solution cannot exist. If both conditions are false, then our approach is not able to decide if a $\mathcal{Q}$ QFC solution exists or not. We base our sufficient [necessary] condition on computing a *(close to) minimum* admissible [full] $\mathcal{Q}$ control abstraction. Theor. 5.9 gives the foundations for such an approach. The proof of Theor. 5.9 follows from the definitions of admissible and full control abstractions and properties of strong and weak solutions (Sect. 5.2). In the following theorem we use the *refinement* order relation (denoted by $\sqsubseteq$) defined in Sect. 2.3.

THEOREM 5.9. *Let $\mathcal{H}$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$, and $(\mathcal{H}, I, G)$ be a control problem.*

(1) If $\hat{\mathcal{H}}$ is an admissible $\mathcal{Q}$ control abstraction and $\hat{K}$ is a strong solution to $(\hat{\mathcal{H}}, \Gamma(I), \Gamma(G))$ then, for any control law k for $\hat{K}$, $K(x,u) = (k(\Gamma(x)) = \Gamma(u))$ is a $\mathcal{Q}$ QFC strong solution to $(\mathcal{H}, I, G)$.

(2) If $\hat{\mathcal{H}}_1 \sqsubseteq \hat{\mathcal{H}}_2$ are two admissible $\mathcal{Q}$ control abstractions of $\mathcal{H}$ and $\hat{K}$ is a strong solution to $(\hat{\mathcal{H}}_2, \Gamma(I), \Gamma(G))$, then $\hat{K}$ is a strong solution to $(\hat{\mathcal{H}}_1, \Gamma(I), \Gamma(G))$.

(3) If $\hat{\mathcal{H}}$ is a full $\mathcal{Q}$ control abstraction and $(\hat{\mathcal{H}}, \Gamma(I), \Gamma(G))$ does not have a weak solution then there exists no $\mathcal{Q}$ QFC (weak as well as strong) solution to $(\mathcal{H}, I, G)$.

(4) If $\hat{\mathcal{H}}_1 \sqsubseteq \hat{\mathcal{H}}_2$ are two full $\mathcal{Q}$ control abstractions of $\mathcal{H}$ and $\hat{K}$ is a weak solution to $(\hat{\mathcal{H}}_1, \Gamma(I), \Gamma(G))$, then $\hat{K}$ is a weak solution to $(\hat{\mathcal{H}}_2, \Gamma(I), \Gamma(G))$.

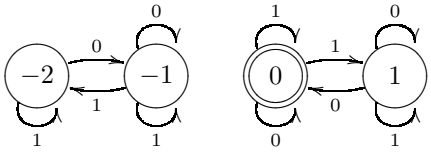


Fig. 6. $\mathcal{Q}$ control abstraction without weak solutions.

Fig. 5 graphically represents a sketch of the correspondence between a concrete DTLHS $\mathcal{H}$ and its control abstractions $\hat{\mathcal{H}}$ lattices.

Example 5.10. Let $\mathcal{P} = (\mathcal{H}, I, G)$ be as in Ex. 4.13 and $\mathcal{Q} = (A, \Gamma)$ be as in Ex. 4.10. For all admissible $\mathcal{Q}$ control abstractions $\hat{\mathcal{H}}$ (see Ex. 5.7) not containing the eliminable self loops $(-1, 0, -1)$ and $(1, 0, 1)$, $\hat{K}(\hat{x}, \hat{u}) \equiv [\hat{x} \neq 0 \rightarrow \hat{u} = 0]$ (see Ex. 4.15) is the strong mgo for $(\hat{\mathcal{H}}, \Gamma(I), \Gamma(G))$. Thus, $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $\mathcal{Q}$ QFC solution to $\mathcal{P}$. Let us consider the quantization $\mathcal{Q}' = (A, \Gamma')$, where $\Gamma'(w) = \lfloor w/2 \rfloor$. A full $\mathcal{Q}'$ control abstraction of $\mathcal{H}$ is $\mathcal{L} = (\{-2, -1, 0, 1\}, \{0, 1\}, \hat{N})$, where the transition $\hat{N}$ is depicted in Fig. 6. $(\mathcal{L}, \Gamma'(I), \Gamma'(G))$ has no weak solution, thus $\mathcal{P}$ has no $\mathcal{Q}'$ QFC solution.

5.2. Proof of Control Abstraction Properties

In this section we give proofs about control abstraction properties. This section can be skipped at a first reading. In the following, we denote with $\mathsf{C}(\mathcal{H}, \mathcal{Q})$ the set of all $\mathcal{Q}$ control abstractions of a DTLHS $\mathcal{H}$.

FACT 5.11. *Let $\mathcal{M}_1 = (S, B, T_1)$ and $\mathcal{M}_2 = (S, B, T_2)$ be two admissible $\mathcal{Q}$ control abstractions of a DTLHS $\mathcal{H}$, with $\mathcal{Q} = (A, \Gamma)$ quantization for $\mathcal{H}$. Then $\forall \hat{x}, \hat{x}' \in S$ s. t. $\hat{x} \neq \hat{x}'$, $\forall \hat{a} \in B\, [T_1(\hat{x}, \hat{a}, \hat{x}') \Leftrightarrow T_2(\hat{x}, \hat{a}, \hat{x}')]$. The same holds if $\mathcal{M}_1, \mathcal{M}_2$ are full $\mathcal{Q}$ control abstractions.*

PROOF. Let $\hat{x} \neq \hat{x}' \in S, \hat{a} \in B$ be such that $T_1(\hat{x}, \hat{a}, \hat{x}')$ holds. If $\mathcal{M}_1$ is an admissible $\mathcal{Q}$ control abstraction, this implies, by Def. 5.6, that $\hat{a}$ is A-admissible in $\hat{x}$. From point 1 of Def. 5.3 (for the admissible control abstraction case) or Def. 5.6 of full control abstraction (for the full control abstraction case), and from $T_1(\hat{x}, \hat{a}, \hat{x}')$ follows that $\exists x \in \Gamma^{-1}(\hat{x}) \exists x' \in \Gamma^{-1}(\hat{x}') : \exists a \in \Gamma^{-1}(\hat{a}) \exists y : N(x, a, y, x')$. By point 2 of Def. 5.3 this implies that $T_2(\hat{x}, \hat{a}, \hat{x}')$ holds.

The same reasoning may be applied to prove the other implication. □

FACT 5.12. *Given a DTLHS $\mathcal{H}$ and a quantization $\mathcal{Q}$, the set $(\mathsf{C}(\mathcal{H}, \mathcal{Q}), \sqsubseteq)$ of $\mathcal{Q}$ control abstractions of $\mathcal{H}$ is a* lattice. *Moreover, the set of full $\mathcal{Q}$ control abstractions of $\mathcal{H}$ is a* lattice.

PROOF. By conditions 2 and 3 of Def. 5.3 all control abstractions do contain all admissible actions that have a concrete witness and all non-eliminable self-loops.

As a consequence, if S is the set of eliminable self-loops and U is the set of non admissible actions, then $(\mathsf{C}(\mathcal{H}, \mathcal{Q}), \sqsubseteq)$ is isomorphic to the complete lattice $(2^{S \times U}, \subseteq)$.

Analogously, the set of full $\mathcal{Q}$ control abstractions of $\mathcal{H}$ is isomorphic to the complete lattice $(2^S, \subseteq)$. □

PROOF THEOREM 5.9. The idea underlying the proof is that two different admissible (as well as full) control abstractions, with the same quantization, have the same loop free structure, i.e. the same arcs except from self loops, as proved by Prop. 5.11. For ease of notation, given a state x (resp. an action u) we will often denote the corresponding abstract state $\Gamma(x)$ (resp. action $\Gamma(u)$) with $\hat{x}$ (resp. $\hat{u}$). Analogously, we will often write $\hat{I}$

(resp. $\hat{G}$) for $\Gamma(I)$ (resp. $\Gamma(G)$). In the following, $\mathcal{P} = (\mathcal{H}, I, G)$, $\hat{\mathcal{P}} = (\hat{\mathcal{H}}, \Gamma(I), \Gamma(G))$, and $\hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$.

Proof of point 1. Applying the definition of solution to a DTLHS control problem (Def. 4.12), we have to show that if $\hat{K}$ is a strong solution to the LTS control problem $(\hat{\mathcal{H}}, \hat{I}, \hat{G})$, then K defined by $K(x,u) = (k(\hat{x}) = \hat{u})$ is a strong solution to the LTS control problem (LTS($\mathcal{H}$), I, $\mathcal{B}_{\|\Gamma\|}(G)$), being k a control law for $\hat{K}$.

Note that, since $\hat{\mathcal{H}}$ is an admissible control abstraction, it contains admissible actions only. This implies that all actions enabled by $\hat{K}$ in $\hat{x}$ are $\mathcal{Q}$-admissible in $\hat{x}$. Hence, we have that all actions enabled by K in x are A-admissible in x. Together with point 2 of Def. 5.3, this implies that, for any transition (x,u,x') of LTS$(\mathcal{H})^{(K)}$ such that $\hat{x} \neq \hat{x}'$, $(\hat{x}, \hat{u}, \hat{x}')$ is a (abstract) transition of $\hat{\mathcal{H}}^{(\hat{K})}$.

First of all, we prove that $I \subseteq \mathrm{Dom}(K)$. Given a state $x \in I$, we have that $\hat{x} \in \hat{I}$. Since $\hat{K}$ is a strong solution to $\hat{\mathcal{P}}$, we have that $\hat{I} \subseteq \mathrm{Dom}(\hat{K})$, thus $\hat{x} \in \mathrm{Dom}(\hat{K})$. Hence, there exists $\hat{u} \in \Gamma(A_U)$ such that $\hat{K}(\hat{x}, \hat{u})$ holds, which implies that $k(\hat{x})$ is defined. By definition of K, we have that for all $u \in \Gamma^{-1}(k(\hat{x}))$ and for all $x \in \Gamma^{-1}(\hat{x})$ $K(x,u)$ holds, which means that $x \in \mathrm{Dom}(K)$.

Now, we prove that for all $x \in \mathrm{Dom}(K)$, $J_{\mathrm{strong}}(\mathrm{LTS}(\mathcal{H})^{(K)}, \mathcal{B}_{\|\Gamma\|}(G), x)$ is finite. Let us suppose by absurd that $J_{\mathrm{strong}}(\mathrm{LTS}(\mathcal{H})^{(K)}, \mathcal{B}_{\|\Gamma\|}(G), x) = \infty$. This implies that one of the two following holds:

(1) there exists a finite fullpath $\pi = x_0 u_0 x_1 u_1 \dots x_n u_n$ in LTS$(\mathcal{H})^{(K)}$ such that $x_0 = x$, $\mathrm{Adm}(\mathrm{LTS}(\mathcal{H})^{(K)}, x_n) = \varnothing$ and, for all $i \in [n]$, $x_i \notin \mathcal{B}_{\|\Gamma\|}(G)$;
(2) there exists an infinite fullpath $\pi = x_0 u_0 x_1 u_1 \dots x_n u_n \dots$ in LTS$(\mathcal{H})^{(K)}$ such that $x_0 = x$ and, for all $i \in \mathbb{N}$, $x_i \notin \mathcal{B}_{\|\Gamma\|}(G)$.

Let us deal with the finite fullpath case first (point 1 above). Let $\hat{\pi} = \hat{x}_0 \hat{u}_0 \dots \hat{u}_{n-1} \hat{x}_n$, and let ρ be defined from $\hat{\pi}$ by collapsing all consecutive equal (abstract) states into one (abstract) state. Formally, $|\rho| = \max_{i \in [n]} \alpha(i)$ and $\rho(i) = \hat{\pi}^{(S)}(\alpha(i)) = \Gamma(\pi^{(S)}(\alpha(i)))$, where the function $\alpha : \mathbb{N} \to \mathbb{N}$ is recursively defined as follows:

— let $Z_z = \{j \mid z < j \leq n \wedge \Gamma(x_j) \neq \Gamma(x_z)\}$
— $\alpha(0) = 0$
— $\alpha(i+1) = \begin{cases} \alpha(i) & \text{if } Z_{\alpha(i)} = \varnothing \\ \min Z_{\alpha(i)} & \text{otherwise} \end{cases}$

By the fact (proved above) that if (x,u,x') is a transition of LTS$(\mathcal{H})^{(K)}$ with $\hat{x} \neq \hat{x}'$, then $(\hat{x}, \hat{u}, \hat{x}')$ is a transition of $\hat{\mathcal{H}}^{(\hat{K})}$, we have that ρ is a run of $\hat{\mathcal{H}}^{(\hat{K})}$. Let $m = |\rho| = \max_{i \in [n]} \alpha(i)$. Since $\hat{K}$ is a strong solution to $\hat{P}$ and $\hat{x} \in \mathrm{Dom}(\hat{K})$, we have that $\hat{x}_m \in \mathrm{Dom}(\hat{K})$. This implies that there exists $\hat{u} \in \Gamma(A_U)$ s.t. $\hat{K}(\hat{x}_m, \hat{u})$ and $k(\hat{x}_m) = \hat{u}$, thus that there exists $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}^{(\hat{K})}, \hat{x}_m)$. Thus by Def. 5.6 (and since $x_n \in \Gamma^{-1}(\hat{x}_m)$) we have that $\mathrm{Adm}(\mathrm{LTS}(\mathcal{H})^{(K)}, x_n) \supseteq \Gamma^{-1}(\hat{u}) \neq \varnothing$, which implies that π cannot be a finite fullpath.

As for the infinite fullpath case (point 2 above), we observe that in π we cannot have an infinite sequence $x_m u_m x_{m+1} u_{m+1} \dots$ such that for all $j \geq m$, $\Gamma(x_j) = \Gamma(x_m)$ and $\Gamma(u_j) = \Gamma(u_m)$. In fact, suppose by absurd that this is true, and let $\tilde{m}$ be the least m for which this happens. Then $(\hat{x}_m, \hat{u}_m, \hat{x}_m)$ is a non-eliminable self loop. Since $x_j \notin \mathcal{B}_{\|\Gamma\|}(G)$ for all $j \geq m$, and thus $\hat{x}_j \notin \hat{G}$ for all $j \geq m$, we also have that $J_{\mathrm{strong}}(\hat{\mathcal{H}}^{(\hat{K})}, \hat{G}, \hat{x}_m) = \infty$. By applying the same reasoning used for the finite fullpath case, we have that there is a path in $\hat{\mathcal{H}}^{(\hat{K})}$ leading from $\hat{x}$ to $\hat{x}_m$, which implies that $J_{\mathrm{strong}}(\hat{\mathcal{H}}^{(\hat{K})}, \hat{G}, \hat{x}) = \infty$. Finally, this

contradicts the fact that $\hat{K}$ is a strong solution to $\hat{P}$ and $\hat{x} \in \mathrm{Dom}(\hat{K})$. Since the control law k for $\hat{K}$ (and thus K, which is defined on k) only enables one action $\hat{u}$ for each abstract state, we may conclude that we cannot have an infinite sequence $x_m u_m x_{m+1} u_{m+1} \dots$ such that for all $j \geq m$, $\Gamma(x_j) = \Gamma(x_m)$.

Thanks to this fact, from a given infinite fullpath $\pi = x_0 u_0 x_1 u_1 \dots x_n u_n \dots$ of $\mathrm{LTS}(\mathcal{H})^{(\alpha)}$ with $x_0 = x$, we can extract an infinite abstract fullpath ρ s.t. $\rho(i) = \Gamma(\pi^{(S)}(\alpha(i)))$, where the function $\alpha : \mathbb{N} \to \mathbb{N}$ is recursively defined as follows:

— $\alpha(0) = 0$
— $\alpha(i+1) = \min\{j \mid \alpha(i) < j \wedge \Gamma(x_j) \neq \Gamma(x_{\alpha(i)})\}$.

By the fact (proved above) that if (x, u, x') is a transition of $\mathrm{LTS}(\mathcal{H})^{(K)}$ with $\hat{x} \neq \hat{x}'$, then $(\hat{x}, \hat{u}, \hat{x}')$ is a transition of $\hat{\mathcal{H}}^{(\hat{K})}$, we have that ρ is a run of $\hat{\mathcal{H}}^{(\hat{K})}$. Moreover, since for all $i \in \mathbb{N}$ $x_i \notin \mathcal{B}_{\|\Gamma\|}(G)$, then we have that for all $i \in \mathbb{N}$ $\hat{x}_{\alpha(i)} \notin \hat{G}$. This contradicts the fact that $\hat{K}$ is a strong solution to $\hat{P}$ and $\hat{x} \in \mathrm{Dom}(\hat{K})$.

Proof of point 2. Let $\hat{\mathcal{H}}_1 = (\Gamma(A_X), \Gamma(A_U), T_1)$ and $\hat{\mathcal{H}}_2 = (\Gamma(A_X), \Gamma(A_U), T_2)$ be two admissible $\mathcal{Q}$ control abstractions of $\mathcal{H}$, with $\hat{\mathcal{H}}_1 \sqsubseteq \hat{\mathcal{H}}_2$. If $\hat{\mathcal{H}}_1 = \hat{\mathcal{H}}_2$ the thesis is proved, thus let us suppose that $\hat{\mathcal{H}}_1 \neq \hat{\mathcal{H}}_2$. By Fact 5.11, the only difference between $\hat{\mathcal{H}}_1$ and $\hat{\mathcal{H}}_2$ may be in a finite number of (eliminable) self loops which are in $\hat{\mathcal{H}}_2$ only. That is, there exists a transitions set $B = \{(\hat{x}_1, \hat{u}_1, \hat{x}_1), \dots, (\hat{x}_m, \hat{u}_m, \hat{x}_m)\}$ s.t. for all $(\hat{x}_i, \hat{u}_i, \hat{x}_i) \in B$ we have that $T_1(\hat{x}_i, \hat{u}_i, \hat{x}_i) = 0 \wedge T_2(\hat{x}_i, \hat{u}_i, \hat{x}_i) = 1$, and for all $(\hat{x}, \hat{u}, \hat{x}') \in \Gamma(A_X) \times \Gamma(A_U) \times \Gamma(A_X)$ we have that if $(\hat{x}, \hat{u}, \hat{x}') \notin B$ then $T_1(\hat{x}, \hat{u}, \hat{x}) = T_2(\hat{x}, \hat{u}, \hat{x})$. Let $\hat{K}$ be the strong mgo to the LTS control problem $(\hat{\mathcal{H}}_2, \hat{I}, \hat{G})$ and let $(\hat{x}_i, \hat{u}_i, \hat{x}_i) \in B$.

Note that if $\hat{x}_i \notin \hat{G}$ and $\hat{K}(\hat{x}_i, \hat{u}_i)$ then $J_{\mathrm{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}_i) = \infty$ since there exists a $\pi \in \mathrm{Path}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{x}_i, \hat{u}_i)$ s.t. $\pi^{(S)}(t) = \hat{x}_i$ and $\pi^{(A)}(t) = \hat{u}_i$ for all $t \in \mathbb{N}$. As a consequence, if $\hat{x}_i \notin \hat{G}$ then $\hat{K}(\hat{x}_i, \hat{u}_i)$ does not hold. Moreover, suppose that $\hat{x}_i \in \hat{G}$. Since $(\hat{x}_i, \hat{u}_i, \hat{x}_i)$ is an eliminable self loop of $\hat{\mathcal{H}}_2$ and $\hat{\mathcal{H}}_2$ is an admissible $\mathcal{Q}$ control abstraction, there exists a state $\hat{x}' \neq \hat{x}_i$ such that $T_2(\hat{x}_i, \hat{u}_i, \hat{x}')$.

We are now ready to prove the thesis. Since we already know that $\hat{I} \subseteq \mathrm{Dom}(\hat{K})$, we only have to prove that i) $\hat{K}$ is a controller for $\hat{\mathcal{H}}_1$ and that ii) $J_{\mathrm{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) < \infty$ for all $\hat{x} \in \mathrm{Dom}(\hat{K})$.

As for the first point, we have to show that $\hat{K}(\hat{x}, \hat{u})$ implies $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}_1, \hat{x})$ (Def. 4.1). Suppose by absurd that $\hat{u} \notin \mathrm{Adm}(\hat{\mathcal{H}}_1, \hat{x})$ for some $\hat{x}, \hat{u}$. Since $\hat{K}(\hat{x}, \hat{u})$ implies $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}_2, \hat{x})$, we have that $(\hat{x}, \hat{u}, \hat{x}) \in B$. If $\hat{x} \notin \hat{G}$ then $\hat{K}(\hat{x}, \hat{u}) = 0$, which is false by hypothesis. If $\hat{x} \in \hat{G}$, then there exists a state $\hat{x}' \neq \hat{x}$ such that $T_2(\hat{x}, \hat{u}, \hat{x}')$. Thus, $T_1(\hat{x}, \hat{u}, \hat{x}')$ holds by Fact 5.11 and we have $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}_1, \hat{x})$, which is absurd.

As for the second one, it is sufficient to prove that $J_{\mathrm{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = J_{\mathrm{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$. This can be proved by induction on the value of $J_{\mathrm{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$.

Suppose $J_{\mathrm{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) = 1$. Then, $\varnothing \neq \mathrm{Img}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{x}, \hat{u}) \subseteq \hat{G}$ for all $\hat{u}$ s.t. $\hat{K}(\hat{x}, \hat{u})$. If for all $\hat{u}$ s.t. $\hat{K}(\hat{x}, \hat{u})$ there exists a state $\hat{x}' \neq \hat{x}$ s.t. $\hat{x}' \in \mathrm{Img}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{x}, \hat{u})$, then we have that $\hat{x}' \in \mathrm{Img}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{x}, \hat{u})$ by Fact 5.11, and since $\varnothing \neq \mathrm{Img}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{x}, \hat{u}) \subseteq \mathrm{Img}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{x}, \hat{u}) \subseteq \hat{G}$ we have that $J_{\mathrm{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = 1 = J_{\mathrm{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$. Otherwise, let $\hat{u}$ be s.t. $\hat{K}(\hat{x}, \hat{u})$ and $T_2(\hat{x}, \hat{u}, \hat{x}') \to \hat{x}' = \hat{x}$. Note that this implies $\hat{x} \in \hat{G}$. If $(\hat{x}, \hat{u}, \hat{x}) \notin B$, then $T_1(\hat{x}, \hat{u}, \hat{x})$ thus $J_{\mathrm{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = 1 = J_{\mathrm{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$. The other case, i.e. $(\hat{x}, \hat{u}, \hat{x}) \in B$, is

impossible since, by the reasoning above and being $\hat{x} \in \hat{G}$, it would imply that there exists a state $\hat{x}' \neq \hat{x}$ such that $T_2(\hat{x}, \hat{u}, \hat{x}')$.

Suppose now that for all $\hat{x}$ s.t. $J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) = n$, $J_{\text{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$. Let $\hat{x} \in \text{Dom}(\hat{K})$ be s.t. $J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) = n + 1$. If $(\hat{x}, \hat{u}, \hat{x}) \notin B$ for any $\hat{u}$, then $\text{Img}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{x}, \hat{u}) = \text{Img}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{x}, \hat{u})$ for all $\hat{u}$, thus $J_{\text{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$ by induction hypothesis. Otherwise, let $(\hat{x}, \hat{u}, \hat{x}) \in B$ for some $\hat{u}$. By the reasoning above, if $\hat{x} \notin \hat{G}$ then $\hat{K}(\hat{x}, \hat{u}) = 0$, and again $J_{\text{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$ by induction hypothesis. If $\hat{x} \in \hat{G}$, then there exists a state $\hat{x}' \neq \hat{x}$ such that $T_2(\hat{x}, \hat{u}, \hat{x}')$ (and $T_1(\hat{x}, \hat{u}, \hat{x}')$). Since $J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) = n + 1$, we must have $J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}') \leq n$, thus again $J_{\text{strong}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x})$ by inductive hypothesis.

Finally, note that in general $\hat{K}$ is not optimal for $(\mathcal{H}_1, \hat{I}, \hat{G})$. As a counterexample, consider the control abstractions $\hat{\mathcal{H}}_2 = (\{0, 1, 2\}, \{0, 1\}, \{(0, 0, 2), (0, 0, 0), (0, 1, 1), (1, 1, 2), (2, 0, 2)\})$ and $\hat{\mathcal{H}}_1 = (\{0, 1, 2\}, \{0, 1\}, \{(0, 0, 2), (0, 1, 1), (1, 1, 2), (2, 0, 2)\})$, with $\hat{I} = \{0, 1, 2\}$ and $\hat{G} = \{2\}$. We have that the strong mgo for $\hat{\mathcal{H}}_2$ is $\hat{K}_2 = \{(0, 1), (1, 1), (2, 0)\}$, whilst the strong mgo for $\hat{\mathcal{H}}_1$ is $\hat{K}_1 = \{(0, 0), (1, 1), (2, 0)\}$, with $J_{\text{strong}}(\hat{\mathcal{H}}_1^{(\hat{K}_1)}, \hat{G}, 0) = 1$ and $J_{\text{strong}}(\hat{\mathcal{H}}_1^{(\hat{K}_2)}, \hat{G}, 0) = J_{\text{strong}}(\hat{\mathcal{H}}_2^{(\hat{K}_2)}, \hat{G}, 0) = 2$.

Proof of point 3. Applying the definition of DTLHS control problem (Def. 4.12), we will show that if K is a weak solution to the LTS control problem ($\text{LTS}(\mathcal{H})$, I, $\mathcal{B}_{\|\Gamma\|}(G)$), and $\hat{\mathcal{H}}$ is any full $\mathcal{Q}$ control abstraction of $\mathcal{H}$ then there exists a weak solution $\hat{K}$ to the control problem $(\hat{\mathcal{H}}, \hat{I}, \hat{G})$.

Let us define, for $\hat{x} \in \Gamma(A_X)$ and $\hat{u} \in \Gamma(A_U)$, $\hat{K}(\hat{x}, \hat{u}) = \exists x \in \Gamma^{-1}(\hat{x})\ \exists u \in \Gamma^{-1}(\hat{u}) : K(x, u)$. We show that $\hat{K}$ is a weak solution to any full $\mathcal{Q}$ control abstraction of $\mathcal{H}$.

Let $\hat{\mathcal{H}}$ be a full $\mathcal{Q}$ control abstraction of $\mathcal{H}$. First of all, we show that $\hat{K}$ is a controller for $\hat{\mathcal{H}}$ (Def. 4.1), i.e. that $\hat{K}(\hat{x}, \hat{u})$ implies $\hat{u} \in \text{Adm}(\hat{\mathcal{H}}, \hat{x})$. Suppose $\hat{K}(\hat{x}, \hat{u})$ holds: this implies that there exist $x \in \Gamma^{-1}(\hat{x}), u \in \Gamma^{-1}(\hat{u})$ s.t. $K(x, u)$ and $u \in \text{Adm}(\mathcal{H}, x)$. If there exists $x' \in A_X$ s.t. $x' \in \text{Img}(\mathcal{H}, x, u)$ and $\hat{x}' \neq \hat{x}$, then, being $\hat{\mathcal{H}}$ a full $\mathcal{Q}$ control abstraction of $\mathcal{H}$, we have that $(\hat{x}, \hat{u}, \hat{x}')$ is a transition of $\hat{\mathcal{H}}$, thus $\hat{u} \in \text{Adm}(\hat{\mathcal{H}}, \hat{x})$. Otherwise, one of the following must hold:

— $\text{Img}(\mathcal{H}, x, u) = \varnothing$, which is impossible since $K(x, u)$;
— for all $x' \in A_X$ s.t. $x' \in \text{Img}(\mathcal{H}, x, u)$, we have that either $x' \notin A_X$ or $\hat{x}' = \hat{x}$. Being K a weak controller for $\mathcal{H}$ defined only on $A_X \times A_U$ (i.e., $K(x, u)$ implies $x \in A_X$ and $u \in A_U$), and given that $K(x, u)$ holds, we must have that there exists $x' \in A_X$ s.t. $x' \in \text{Img}(\mathcal{H}, x, u)$ and $\hat{x}' = \hat{x}$. If $x = x'$, then there exists an infinite path inside $\Gamma^{-1}(\hat{x})$ with actions in $\Gamma^{-1}(\hat{u})$, i.e. $(\hat{x}, \hat{u}, \hat{x})$ is a non-eliminable self loop. This implies that $\hat{N}(\hat{x}, \hat{u}, \hat{x})$ holds, thus $\hat{u} \in \text{Adm}(\hat{\mathcal{H}}, \hat{x})$. Otherwise, i.e. if $x \neq x'$, then we whole reasoning may be applied to x'. Then, either we arrive to a state $t \notin \Gamma^{-1}(\hat{x})$ starting from a state in $\Gamma^{-1}(\hat{x})$, and $\hat{N}(\hat{x}, \hat{u}, \hat{t})$ implies $\hat{u} \in \text{Adm}(\hat{\mathcal{H}}, \hat{x})$, or we have an infinite path inside $\Gamma^{-1}(\hat{x})$ via $\Gamma^{-1}(\hat{u})$, thus $(\hat{x}, \hat{u}, \hat{x})$ is a non-eliminable self loop and $\hat{N}(\hat{x}, \hat{u}, \hat{x})$ implies $\hat{u} \in \text{Adm}(\hat{\mathcal{H}}, \hat{x})$.

We now have to prove that $\hat{K}$ is a weak solution to $\hat{\mathcal{H}}$, where $\hat{\mathcal{H}}$ is a full $\mathcal{Q}$ control abstraction of $\mathcal{H}$. First of all, we show that $\hat{I} \subseteq \text{Dom}(\hat{K})$. Given $\hat{x} \in \hat{I}$, we have that there

exists $x \in \Gamma^{-1}(\hat{x})$ such that $x \in I$. Since K is a weak solution to $\mathcal{P}$, there exists $u \in A_U$ s.t. $K(x,u)$, thus by definition of $\hat{K}$, $\hat{K}(\hat{x},\hat{u})$ holds, and hence $\hat{x} \in \mathrm{Dom}(\hat{K})$.

Now, we show that for all $\hat{x} \in \mathrm{Dom}(\hat{K})$, $J_{\mathrm{weak}}(\hat{\mathcal{H}}^{(\hat{K})}, \hat{G}, \hat{x})$ is finite. By definition of $\hat{K}$, and since K is a weak solution to $\mathcal{P}$, there exists a finite path $\pi = x_0 u_0 x_1 u_1 \dots u_{n-1} x_n$ such that $x_0 \in \Gamma^{-1}(\hat{x})$, $x_i \in A_X$ for all $0 \leq i \leq n$ and $x_n \in \mathcal{B}_{\|\Gamma\|}(G)$.

Let $\hat{\pi} = \hat{x}_0 \hat{u}_0 \dots \hat{u}_{n-1} \hat{x}_n$, and let ρ be defined from $\hat{\pi}$ by collapsing all consecutive equal (abstract) states into one state. Formally, $|\rho| = \max_{i \in [n]} \alpha(i)$ and $\rho(i) = \hat{\pi}^{(S)}(\alpha(i)) = \Gamma(\pi^{(S)}(\alpha(i)))$, where the function $\alpha : \mathbb{N} \to \mathbb{N}$ is recursively defined as follows:

— let $Z_z = \{j \mid z < j \leq n \wedge \Gamma(x_j) \neq \Gamma(x_z)\}$
— $\alpha(0) = 0$
— $\alpha(i+1) = \begin{cases} \alpha(i) & \text{if } Z_{\alpha(i)} = \varnothing \\ \min Z_{\alpha(i)} & \text{otherwise} \end{cases}$

In a full $\mathcal{Q}$ control abstraction $\hat{\mathcal{H}}$, if (x,u,x') is transition of $\mathrm{LTS}(\mathcal{H})$ and $\hat{x} \neq \hat{x}'$, then $\hat{N}(\hat{x},\hat{u},\hat{x}')$. Then we have that ρ is a finite path in $\hat{\mathcal{H}}^{(\hat{K})}$ that leads from $\hat{x}_0 = \hat{x}$ to the goal. As a consequence, $\hat{K}$ is a weak solution to $\hat{\mathcal{P}}$.

Proof of point 4. Analogously to the proof of point 2, let $\hat{\mathcal{H}}_1 = (\Gamma(A_X), \Gamma(A_U), T_1)$ and $\hat{\mathcal{H}}_2 = (\Gamma(A_X), \Gamma(A_U), T_2)$ be two full $\mathcal{Q}$ control abstractions of $\mathcal{H}$, with $\hat{\mathcal{H}}_1 \sqsubseteq \hat{\mathcal{H}}_2$. If $\hat{\mathcal{H}}_1 = \hat{\mathcal{H}}_2$ the thesis is proved, thus let us suppose that $\hat{\mathcal{H}}_1 \neq \hat{\mathcal{H}}_2$. By Fact 5.11, the only difference between $\hat{\mathcal{H}}_1$ and $\hat{\mathcal{H}}_2$ may be in a finite number of eliminable self loops which are in $\hat{\mathcal{H}}_2$ only. Let $B = \{(\hat{x}_1,\hat{u}_1,\hat{x}_1), \dots, (\hat{x}_m,\hat{u}_m,\hat{x}_m)\}$ be the set of such self loops. Let $\hat{K}$ be the weak mgo to the LTS control problem $(\hat{\mathcal{H}}_1, \hat{I}, \hat{G})$ and let $(\hat{x}_i,\hat{u}_i,\hat{x}_i) \in B$.

Since we already know that $\hat{I} \subseteq \mathrm{Dom}(\hat{K})$, we only have to prove that i) $\hat{K}$ is a controller for $\hat{\mathcal{H}}_2$ and that ii) $J_{\mathrm{weak}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) < \infty$ for all $\hat{x} \in \mathrm{Dom}(\hat{K})$.

As for the first point, we have to show that $\hat{K}(\hat{x},\hat{u})$ implies $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}_2, \hat{x})$ (Def. 4.1). Since $\hat{K}(\hat{x},\hat{u})$ implies $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}_1, \hat{x})$, and since $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}_1, \hat{x})$ implies $\hat{u} \in \mathrm{Adm}(\hat{\mathcal{H}}_2, \hat{x})$, this point is proved.

As for the second one, it is sufficient to prove that $J_{\mathrm{weak}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) \leq J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x})$. This can be proved by induction on the value of $J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x})$.

Suppose $J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = 1$. Then, $\mathrm{Img}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{x}, \hat{u}) \cap \hat{G} \neq \varnothing$ for all $\hat{u}$ s.t. $\hat{K}(\hat{x},\hat{u})$. Since $\hat{\mathcal{H}}_2$ only adds self loops to $\hat{\mathcal{H}}_1$, we have that $\mathrm{Img}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{x}, \hat{u}) \cap \hat{G} \neq \varnothing$ for all $\hat{u}$ s.t. $\hat{K}(\hat{x},\hat{u})$, thus $J_{\mathrm{weak}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) = 1 = J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x})$.

Suppose now that for all $\hat{x}$ s.t. $J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = n$, $J_{\mathrm{weak}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) \leq J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x})$. Let $\hat{x}$ be s.t. $J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}) = n+1$. If $(\hat{x},\hat{u},\hat{x}) \notin B$ for any $\hat{u}$, then $\mathrm{Img}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{x}, \hat{u}) = \mathrm{Img}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{x}, \hat{u})$ for all $\hat{u}$, thus $J_{\mathrm{weak}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) \leq J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x})$ by induction hypothesis. Otherwise, let $(\hat{x},\hat{u},\hat{x}) \in B$ for some $\hat{u}$. If $\hat{x} \notin \hat{G}$ we simply have that $J_{\mathrm{weak}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) \leq J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x})$ by induction hypothesis. Otherwise, if $\hat{x} \in \hat{G}$, let $\hat{K}_1$ be s.t. $\hat{K}_1(\hat{x},\hat{u}) = 0$ and $\hat{K}_1(\hat{s},\hat{a}) = \hat{K}(\hat{s},\hat{a})$ for $(\hat{s},\hat{a}) \neq (\hat{x},\hat{u})$. Then, $J_{\mathrm{weak}}(\hat{\mathcal{H}}_2^{(\hat{K})}, \hat{G}, \hat{x}) = \max\{1, J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K}_1)}, \hat{G}, \hat{x})\} \leq J_{\mathrm{weak}}(\hat{\mathcal{H}}_1^{(\hat{K})}, \hat{G}, \hat{x}')$, thus the thesis is proved. □

6. QUANTIZED CONTROLLER SYNTHESIS

In this section, we present the quantized controller synthesis algorithm (function *qCtrSyn* in Alg. 1). Function *qCtrSyn* takes as input a DTLHS control problem $\mathcal{P} = (\mathcal{H}, I, G)$ and a quantization $\mathcal{Q}$. Then, resting on Theor. 5.9, *qCtrSyn* computes an admissible $\mathcal{Q}$ control abstraction $\hat{\mathcal{M}}$ in order to find a $\mathcal{Q}$ QFC strong solution to $\mathcal{P}$, and a full $\mathcal{Q}$ control abstraction $\hat{\mathcal{W}}$ to determine if such a solution does not exist.

Sects. 6.6, 6.7, 6.8, and 6.9 show theoretical and implementation details that can be skipped at a first reading.

Namely, as for the sufficient condition, we compute the strong mgo $\hat{K}$ for the LTS control problem $(\hat{\mathcal{M}}, \Gamma(I), \Gamma(G))$. If $\hat{K}$ exists, then a $\mathcal{Q}$ QFC strong solution to $\mathcal{P}$ may be built from $\hat{K}$. Note that, if $\hat{K}$ does not exist, a strong solution may exist for some other admissible $\mathcal{Q}$ control abstraction $\hat{\mathcal{H}}$. However, by point 2 of Theor. 5.9, $\hat{\mathcal{H}}$ must be lower than $\hat{\mathcal{M}}$ in the hierarchy lattice (see Fig. 5). This suggests to compute $\hat{\mathcal{M}}$ as the minimum (admissible) $\mathcal{Q}$ control abstraction of $\mathcal{H}$. Since by Prop. 5.8 we are not able to compute the minimum $\mathcal{Q}$ control abstraction, we compute $\hat{\mathcal{M}}$ as a *close to* minimum admissible $\mathcal{Q}$ control abstraction, i.e. an admissible $\mathcal{Q}$ control abstraction containing as few eliminable self loops as possible (see Ex. 4.4).

As for the necessary condition, we compute the weak mgo $\hat{K}$ for the LTS control problem $(\hat{\mathcal{W}}, \Gamma(I), \Gamma(G))$. If $\hat{K}$ does not exists, then a $\mathcal{Q}$ QFC (weak as well as strong) solution to $\mathcal{P}$ cannot exist. Note that, if $\hat{K}$ exists, a weak solution may not exist for some other full $\mathcal{Q}$ control abstraction $\hat{\mathcal{H}}$. However, by point 4 of Theor. 5.9, $\hat{\mathcal{H}}$ must be lower than $\hat{\mathcal{W}}$ in the hierarchy lattice (see Fig. 5). Hence, again by Prop. 5.8, we compute $\hat{\mathcal{W}}$ as the close to minimum full $\mathcal{Q}$ control abstraction.

Algorithm 1 QFC synthesis

Input: DTLHS control problem $(\mathcal{H}, I, G)$, quantization $\mathcal{Q} = (A, \Gamma)$
function *qCtrSyn*($\mathcal{H}$, $\mathcal{Q}$, I, G)
1. $\hat{I} \leftarrow \Gamma(I)$, $\hat{G} \leftarrow \Gamma(G)$
2. $\hat{\mathcal{M}} \leftarrow$*minCtrAbs*($\mathcal{H}$, $\mathcal{Q}$)
3. $(b, \hat{D}, \hat{K}) \leftarrow$ *strongCtr*($\hat{\mathcal{M}}$, $\hat{I}$, $\hat{G}$)
4. **if** b **then return** (SOL, $\hat{D}, \hat{K}$)
5. $\hat{\mathcal{W}} \leftarrow$ *minFullCtrAbs* ($\mathcal{H}, \mathcal{Q}$)
6. **if** *existsWeakCtr*($\hat{\mathcal{W}}, \hat{I}, \hat{G}$) **then return** (UNK, $\hat{D}, \hat{K}$)
7. **else return** (NOSOL, $\hat{D}, \hat{K}$)

6.1. QFC Synthesis Algorithm

Our QFC synthesis algorithm (function *qCtrSyn* outlined in Alg. 1) takes as input a DTLHS $\mathcal{H} = (X, U, Y, N)$, a quantization $\mathcal{Q} = (A, \Gamma)$, and two predicates I and G over X, such that $(\mathcal{H}, I, G)$ is a DTLHS control problem. Function *qCtrSyn* returns a tuple $(\mu, \hat{D}, \hat{K})$, where: $\mu \in \{\text{SOL}, \text{NOSOL}, \text{UNK}\}$, $\hat{D} = \text{Dom}(\hat{K})$ and $\hat{K}$ is such that the controller K, defined by $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $\mathcal{Q}$ QFC (strong) solution to the control problem $(\mathcal{H}, \Gamma^{-1}(\hat{D}), G)$.

We represent boolean functions (e.g. the transition relation of $\hat{\mathcal{H}}$) and sets (by using their characteristic functions) using *Ordered Binary Decision Diagrams* (OBDD) [Bryant 1986].

For the sake of clarity, however, we will present our algorithms using a set theoretic notation for sets and predicates over sets.

Alg. 1 starts (line 1) by computing a quantization $\hat{I}$ of the initial region I and a quantization $\hat{G}$ of the goal region G (further details are given in Sect. 6.3).

Function *minCtrAbs* in line 2 computes the close to minimum $\mathcal{Q}$ control abstraction $\hat{\mathcal{M}}$ of $\mathcal{H}$ (see Sect. 6.4.1 for further details about *minFullCtrAbs*).

Line 3 determines if a strong mgo to the LTS control problem $\hat{\mathcal{P}} = (\hat{\mathcal{M}}, \hat{I}, \hat{G})$ exists by calling function *strongCtr* that implements a variant of the algorithm in [Cimatti et al. 1998]. Given $\hat{\mathcal{M}}, \hat{I}, \hat{G}$, function *strongCtr* returns a triple $(b, \hat{D}, \hat{K})$ such that $\hat{K}$ is the strong mgo to $(\hat{\mathcal{M}}, \varnothing, \hat{G})$ and $\hat{D} = \mathrm{Dom}(\hat{K})$. If b is TRUE then $\hat{K}$ is a strong mgo for $\hat{\mathcal{P}}$ (i.e. $\hat{I} \subseteq \hat{D}$), and *qCtrSyn* returns the tuple $(\textsc{Sol}, \hat{D}, \hat{K})$ (line 4). By Theor. 5.9 (point 1), $K(x,u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $\mathcal{Q}$ QFC solution to the DTLHS control problem $(\mathcal{H}, I, G)$. Otherwise, in lines 5–7 *qCtrSyn* tries to establish if such a solution may exist or not.

Function *minFullCtrAbs* in line 5 computes the close to minimum full $\mathcal{Q}$ control abstraction $\hat{\mathcal{W}}$ of $\mathcal{H}$ (see Sect. 6.4.1 for further details about *minFullCtrAbs*). Line 6 checks if the weak mgo to $\hat{\mathcal{P}}' = (\hat{\mathcal{W}}, \hat{I}, \hat{G})$ exists by calling function *existsWeakCtr*, which is based on the algorithm in [Tronci 1998].

If function *existsWeakCtr* returns FALSE, then a weak mgo to $\hat{\mathcal{P}}'$ does not exist, and since the weak mgo is unique no weak solution exists to $\hat{\mathcal{P}}'$. By Theor. 5.9 (point 3), no $\mathcal{Q}$ QFC solution exists for the DTLHS control problem $(\mathcal{H}, I, G)$ and accordingly *qCtrSyn* returns NOSOL (line 7). Otherwise no conclusion can be drawn and accordingly UNK is returned (line 6). In any case, the strong mgo $\hat{K}$ for $\hat{\mathcal{P}}$ for the (close to) minimum control abstraction is returned, together with its controlled region $\hat{D}$.

6.2. Synthesis Algorithm Correctness

The above considerations imply correctness of function *qCtrSyn* (and thus of our approach), as stated by the following theorem.

THEOREM 6.1. *Let $\mathcal{H}$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization, and $(\mathcal{H}, I, G)$ be a DTLHS control problem. Then qCtrSyn$(\mathcal{H}, \mathcal{Q}, I, G)$ returns a triple $(\mu, \hat{D}, \hat{K})$ such that: $\mu \in \{\textsc{Sol}, \textsc{NoSol}, \textsc{Unk}\}$, $\hat{D} = \mathrm{Dom}(\hat{K})$ and, for all control laws k for $\hat{K}$, $K(x,u) = (k(\Gamma(x)) = \Gamma(u))$ is a $\mathcal{Q}$ QFC solution to the control problem $(\mathcal{H}, \Gamma^{-1}(\hat{D}), G)$. Furthermore, the following holds: i) if $\mu =$ SOL then $I \subseteq \Gamma^{-1}(\hat{D})$ and K is a $\mathcal{Q}$ QFC solution to the control problem $(\mathcal{H}, I, G)$; ii) if $\mu =$ NOSOL then there is no $\mathcal{Q}$ QFC solution to the control problem $(\mathcal{H}, I, G)$.*

Remark 6.2. [Mazo and Tabuada 2011] describes a method for the automatic control software synthesis for continuous time linear systems. Function *strongCtr*, as well as the approach in [Mazo and Tabuada 2011], returns $\hat{K}$ as a (worst case) *time optimal* controller, i.e. in each state $\hat{K}$ enables the actions leading to a goal state in the least number of transitions. This stems from the fact that in both cases (*strongCtr* and [Mazo and Tabuada 2011]) the OBDD representation for the controller is computed using the approach in [Cimatti et al. 1998] where symbolic control synthesis algorithms for finite state LTSs have been studied in a universal planning setting.

Remark 6.3. Instead of computing the controller (function *strongCtr*) with [Cimatti et al. 1998], it is possible to trade the size of the synthesized controller with time optimality while preserving closed loop performances. Such an issue has been investigated in [Alimguzhin et al. 2012b].

Remark 6.4. Note however that $\hat{K}$ may not be time optimal for the real plant. In fact, self loops elimination shrinks all concrete sequences of the form $x_n, u_n, \ldots, x_m$ in every path of LTS($\mathcal{H}$) into a single abstract transition $(\hat{x}_n, \hat{u}_n, \hat{x}_m)$ of $\hat{\mathcal{M}}$ whenever $\hat{x}_n = \ldots = \hat{x}_{m-1}$ and $\hat{u}_n = \ldots = \hat{u}_{m-1}$. Thus, the length of paths in the plant model and those in the control abstraction used for the synthesis may not coincide. Moreover, nondeterminism added by quantization might lead to prefer an action $\hat{u}_1$ to an action $\hat{u}_2$ for an abstract state $\hat{x}$, whilst actions in $\hat{u}_2$ might be better for some real states inside $\hat{x}$. Finally, since we are not able to compute the minimum control abstraction, we may discard a possibly optimal action $\hat{u}$ on a state $\hat{x}$ if the following holds: $(\hat{x}, \hat{u}, \hat{x})$ is an eliminable self loop, but function *minCtrAbs* decides that it is non-eliminable. For these reasons we refer to our controller as a *near time optimal* controller.

6.3. Quantization

In the following let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$, and $(\mathcal{H}, I, G)$ be a DTLHS control problem.

In our approach we consider Γ only in problems of type $P(W) \equiv (\max, J(W), L(W) \wedge (\Gamma(W) = \hat{v}))$, where W is either X, X' or U, $J(W)$ is a linear expression, $L(W)$ a conjunctive predicate and $(\Gamma(W) = \hat{v}) \equiv \bigwedge_{i \in [|W|]} (\gamma_{w_i}(w_i) = \hat{v}_i)$, with $w_i \in W$. In order to be able to solve $P(W)$ via a MILP solver, we restrict ourselves to quantization functions γ_{w_i} for which equality tests can be represented by using conjunctive predicates. Namely, for $w \in X \cup U$, we employ the uniform quantization $\gamma_w : A_w \to [0, \Delta_w - 1]$, defined for a given Δ_w as follows. Let $\delta_w = (\sup A_w - \inf A_w)/\Delta_w$. We have that $\gamma_w(w) = \hat{z}$ if and only if the conjunctive predicate $P_{\gamma_w}(w, \hat{z}) \equiv \inf A_w + \delta_w \hat{z} \leq w \leq \inf A_w + \delta_w(\hat{z} + 1)$ holds.

We may now explain how $\hat{I}, \hat{G}$ are effectively computed in line 1 of Alg. 1. Since the initial region I is represented as a conjunctive predicate, its quantization $\hat{I}$ is computed by solving $|\Gamma(A_X)|$ feasibility problems. More precisely, $\hat{I} = \{\hat{x} \mid \mathit{feasible}(I(X) \wedge \Gamma(X) = \hat{x})\}$. Similarly, the quantization $\hat{G}$ of the goal region G is $\hat{G} = \{\hat{x} \mid \mathit{feasible}(G(X) \wedge \Gamma(X) = \hat{x})\}$.

Algorithm 2 Building control abstractions

Input: DTLHS $\mathcal{H} = (X, U, Y, N)$, quantization $\mathcal{Q} = (A, \Gamma)$.
function *minCtrAbs* $(\mathcal{H}, \mathcal{Q})$

1. $\hat{N} \leftarrow \varnothing$
2. **for all** $\hat{x} \in \Gamma(A_X)$ **do**
3. **for all** $\hat{u} \in \Gamma(A_U)$ **do**
4. **if** $\neg$ $\mathcal{Q}$-*admissible*$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u})$ **then continue**
5. **if** *selfLoop*$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u})$ **then** $\hat{N} \leftarrow \hat{N} \cup \{(\hat{x}, \hat{u}, \hat{x})\}$
6. $\mathcal{O} \leftarrow$ *overImg*$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u})$
7. **for all** $\hat{x}' \in \Gamma(\mathcal{O})$ **do**
8. **if** $\hat{x} \neq \hat{x}' \wedge$*existsTrans*$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u}, \hat{x}')$ **then**
9. $\hat{N} \leftarrow \hat{N} \cup \{(\hat{x}, \hat{u}, \hat{x}')\}$
10. **return** $\hat{N}$

6.4. Computing Minimum Control Abstractions

In this section, we present in Alg. 2 function *minCtrAbs*, which effectively computes a close to minimum $\mathcal{Q}$ control abstraction $\hat{\mathcal{M}} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$ for a given $\mathcal{H}$.

Starting from the empty transition relation (line 1) function *minCtrAbs* checks for every triple $(\hat{x}, \hat{u}, \hat{x}') \in \Gamma(A_X) \times \Gamma(A_U) \times \Gamma(A_X)$ if the transition $(\hat{x}, \hat{u}, \hat{x}')$ belongs to $\hat{\mathcal{M}}$ and accordingly adds it to $\hat{N}$ or not.

For any pair $(\hat{x}, \hat{u})$ in $\Gamma(A_X) \times \Gamma(A_U)$ line 4 checks if $\hat{u}$ is $\mathcal{Q}$-admissible in $\hat{x}$. This check is carried out by determining if the predicate $P(X, U, Y, X', \hat{x}, \hat{u}) \equiv N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge X' \notin A_X$ is not feasible. If $\hat{u}$ is not $\mathcal{Q}$-admissible in $\hat{x}$ (i.e., if $P(X, U, Y, X', \hat{x}, \hat{u})$ is feasible), no transition of the form $(\hat{x}, \hat{u}, \hat{x}')$ is added to $\hat{N}$. Note that $P(X, U, Y, X', \hat{x}, \hat{u})$ is not a conjunctive predicate, however it is possible to check its feasibility by properly calling function *feasible* $2|X|$ times (Sect. 6.9).

If $\hat{u}$ is $\mathcal{Q}$-admissible in $\hat{x}$, line 5 checks if the self loop $(\hat{x}, \hat{u}, \hat{x})$ has to be added to $\hat{N}$. To this aim, we employ a function *selfLoop* (see Sect. 6.5) which takes a (state, action) pair $(\hat{x}, \hat{u})$ and returns FALSE if the self loop $(\hat{x}, \hat{u}, \hat{x})$ is eliminable.

Function *overImg* (line 6) computes a rectangular region $\mathcal{O}$, that is a *quite tight* over-approximation of the set of one step reachable states from $\hat{x}$ via $\hat{u}$. $\mathcal{O}$ is obtained by computing for each state variable x_i the minimum and maximum possible values for the corresponding next state variable. Namely, $\mathcal{O} = \prod_{i=1,\ldots,|X|}[\gamma_{x_i}(m_i), \gamma_{x_i}(M_i)]$ where $m_i = \mathit{optimalValue}(\min,\ x_i',\ N(X, U, Y, X') \wedge A(X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u})$ and $M_i = \mathit{optimalValue}(\max,\ x_i',\ N(X, U, Y, X') \wedge A(X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u})$.

Finally, for each abstract state $\hat{x}' \in \Gamma(\mathcal{O})$ line 8 checks if there exists a concrete transition realizing the abstract transition $(\hat{x},\ \hat{u},\ \hat{x}')$ when $\hat{x} \neq \hat{x}'$. To this end, function *existsTrans* solves the MILP problem $N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge \Gamma(X') = \hat{x}'$.

Remark 6.5. From the nested loops in lines 2, 3, 7 we have that *minCtrAbs* worst case runtime is $O(|\Gamma(A_X)|^2|\Gamma(A_U)|)$. However, thanks to the heuristic implemented in function *overImg*, *minCtrAbs* typical runtime is about $O(|\Gamma(A_X)||\Gamma(A_U)|)$ as confirmed by our experimental results (see Sect. 8, Fig. 7). The same holds for function *minFullCtrAbs* (see Sect. 6.4.1).

Remark 6.6. Function *minCtrAbs* is explicit in the (abstract) states and actions of $\hat{\mathcal{H}}$ and symbolic with respect to the auxiliary variables (*modes*) in the transition relation N of $\mathcal{H}$. As a result our approach will work well with systems with just a few state variables and many modes, our target here.

6.4.1. Computing Minimum Full Control Abstraction. Function *minCtrAbs* can be easily modified in order to compute the close to minimum full $\mathcal{Q}$ control abstraction, thus obtaining function *minFullCtrAbs* called in Alg. 1, line 5. Function *minFullCtrAbs* is obtained by removing the highlighted code (on grey background) from Alg. 2, namely the admissibility check in line 4.

6.5. Self Loop Elimination

In order to exactly get the minimum control abstraction, function *selfLoop* should return TRUE iff the given self loop is non-eliminable. This is undecidable by Prop. 5.5. Function *selfLoop*, outlined in Alg. 3, checks a sufficient *gradient based* condition for self loop elimination that in practice turns out to be very effective (see Tabs. I and II in Sect. 8). That is, function *selfLoop* returns FALSE when a self loop is eliminable (or there is not a concrete witness for it). On the other hand, if function *selfLoop* returns TRUE, then the self loop under consideration may be non-eliminable as well as eliminable. In a conservative way, we assume self loops for which function *selfLoop* returns TRUE to be non-eliminable (i.e. they are added to $\hat{\mathcal{M}}$, see line 5 of Alg. 2).

Function *selfLoop* in Alg. 3, which correctness is proved in Sect. 6.6, works as follows. First of all it checks if there is a concrete witness for the self loop under consideration. If it is not the case, *selfLoop* returns FALSE (line 1). Otherwise, for each real variable x_i, it tries to

establish if x_i is either always increasing (line 4) or always decreasing (line 6) inside $\Gamma^{-1}(\hat{x})$ by performing actions in $\Gamma^{-1}(\hat{u})$. If this is the case, we have that, being $\Gamma^{-1}(\hat{x})$ a compact set, no Zeno-phenomena may arise, thus executing actions in $\Gamma^{-1}(\hat{u})$ it is guaranteed that $\mathcal{H}$ will eventually leave the region $\Gamma^{-1}(\hat{x})$. Otherwise, TRUE is returned in line 7.

Algorithm 3 Self loop elimination

Input: DTLHS $\mathcal{H} = (X, U, Y, N)$, quantization $\mathcal{Q} = (A, \Gamma)$, abstract state $\hat{x}$, abstract action $\hat{u}$.

function *selfLoop*$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u})$

1. **if** $\neg$*existsTrans*$(\hat{x}, \hat{u}, \hat{x})$ **then return** FALSE
2. **for** x_i **in** X^r **do**
3. $w_i \leftarrow$ *optimalValue*$(\min, x_i' - x_i, N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge \Gamma(X') = \hat{x})$
4. **if** $w_i > 0$ **then return** FALSE
5. $W_i \leftarrow$ *optimalValue*$(\max, x_i' - x_i, N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge \Gamma(X') = \hat{x})$
6. **if** $W_i < 0$ **then return** FALSE
7. **return** TRUE

6.6. Proof of Function *selfLoop* Correctness

In this section we prove correctness of Alg. 3. This section can be skipped at a first reading.

PROPOSITION 6.7. *Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$, $\hat{x} \in \Gamma(A_X)$, and $\hat{u} \in \Gamma(A_U)$. If the abstract self loop $(\hat{x}, \hat{u}, \hat{x})$ has a concrete witness and selfLoop$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u})$ returns* FALSE*, then $(\hat{x}, \hat{u}, \hat{x})$ is an eliminable self loop.*

PROOF. Suppose by absurd that the abstract self loop $(\hat{x}, \hat{u}, \hat{x})$ has a concrete witness, *selfLoop*$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u})$ returns FALSE, and $(\hat{x}, \hat{u}, \hat{x})$ is a non-eliminable self loop. Then there exists an infinite run $\pi = x_0 u_0 x_1 u_1 \dots$ such that for all $t \in \mathbb{N}$ $x_t \in \Gamma^{-1}(\hat{x})$ and $u_t \in \Gamma^{-1}(\hat{u})$.

For $i \in [|X^r|]$, let $w_i \leq W_i$ be the values computed in lines 3 and 5 of Alg. 3, i.e. $w_i =$ *optimalValue*$(\min, x_i' - x_i, N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge \Gamma(X') = \hat{x})$ and $W_i =$ *optimalValue*$(\max, x_i' - x_i, N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge \Gamma(X') = \hat{x})$.

Since *selfLoop*$(\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u})$ returns FALSE, there exists at least an index $j \in [|X^r|]$ such that $w_j > 0$ or $W_j < 0$ (see lines 4 and 6 of Alg. 3 resp.). Let us consider the former case (note that $w_j > 0$ implies $W_j > 0$).

For all $k \in \mathbb{N}$, we have that $|(x_k)_j - (x_0)_j| = (x_k)_j - (x_0)_j \geq k w_j$. If we take $\tilde{k} > \frac{\|\gamma_{x_j}\|}{w_j}$, we have that $|(x_{\tilde{k}})_j - (x_0)_j| > \|\gamma_{x_j}\|$ and hence $x_{\tilde{k}}$ cannot belong to $\Gamma^{-1}(\hat{x})$.

Analogously, if $w_j \leq W_j < 0$ then we have that $|(x_k)_j - (x_0)_j| = (x_0)_j - (x_k)_j \geq k w_j$. If we take $\tilde{k} > \frac{\|\gamma_{x_j}\|}{w_j}$, we have that $|(x_{\tilde{k}})_j - (x_0)_j| > \|\gamma_{x_j}\|$ and hence $x_{\tilde{k}}$ cannot belong to $\Gamma^{-1}(\hat{x})$.

In both cases we have a contradiction, thus the thesis is proved. □

6.7. Proof of Functions *minCtrAbs* and *minFullCtrAbs* Correctness

In this section we prove correctness of functions *minCtrAbs* (Alg. 2) and *minFullCtrAbs* used in Alg. 1. This section can be skipped at a first reading.

PROPOSITION 6.8. *Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS and $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$.*

If $\hat{N}$ is the transition relation computed by minCtrAbs($\mathcal{H}$, $\mathcal{Q}$) then $\hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$ is an admissible $\mathcal{Q}$ control abstraction of $\mathcal{H}$.

If $\hat{N}$ is the transition relation computed by minFullCtrAbs($\mathcal{H}$, $\mathcal{Q}$) then $\hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$ is a full $\mathcal{Q}$ control abstraction of $\mathcal{H}$.

PROOF. Here we prove only the part regarding function *minCtrAbs*, since the other part may be proved analogously. We first show that the control abstraction $\hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$ satisfies conditions 1–3 of Def. 5.3.

(1) Each transition $(\hat{x}, \hat{u}, \hat{x}')$ is added to $\hat{N}$ in line 5 or in line 9 of Alg. 2. In both cases, it has been checked by function *existsTrans* that $\exists x \in \Gamma^{-1}(\hat{x})$, $u \in \Gamma^{-1}(\hat{u})$, $x' \in \Gamma^{-1}(\hat{x}')$, $y \in A_Y$ such that $N(x, u, y, x')$ (in the latter case the check is inside function *selfLoop*).
(2) Let $x, s' \in A_X$ and $u \in A_U$ be such that $\exists y : N(x, u, y, x')$ and $\Gamma(x) \neq \Gamma(x')$. Since *minCtrAbs* examines all tuples in $\Gamma(A_X) \times \Gamma(A_U) \times \Gamma(A_X)$, it will eventually examine the tuple $(\hat{x}, \hat{u}, \hat{x}')$ s.t. $\hat{x} = \Gamma(x)$, $\hat{u} = \Gamma(u)$, and $\hat{x}' = \Gamma(x')$. If $\hat{u}$ is not $\mathcal{Q}$-admissible in $\hat{x}$ no transition is added to $\hat{N}$ because of the check in line 4. Otherwise, since $\exists y : N(x, u, y, x')$ holds, *existsTrans*($\hat{x}$, $\hat{u}$, $\hat{x}'$) returns TRUE and the transition $(\hat{x}, \hat{u}, \hat{x}')$ is added to $\hat{N}$ in line 9 of Alg. 2.
(3) Note that condition 3 of Def. 5.3 may be rephrased as follows: if $(\hat{x}, \hat{u}, \hat{x})$ is a non-eliminable self loop, then $\hat{N}(\hat{x}, \hat{u}, \hat{x})$ must hold. That is, if $\hat{N}(\hat{x}, \hat{u}, \hat{x}) = 0$ then either there is not a concrete witness for the self loop $(\hat{x}, \hat{u}, \hat{x})$, or $(\hat{x}, \hat{u}, \hat{x})$ is an eliminable self loop. This is exactly the case for which function *selfLoop*($\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u}$) returns FALSE (resp. by line 1 of Alg. 3 and by Prop. 6.7). Since a self loop $(\hat{x}, \hat{u}, \hat{x})$ is not added to $\hat{N}$ only if *selfLoop*($\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u}$) returns FALSE in line 5 of Alg. 2, and since function *selfLoop*($\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u}$) is eventually invoked for all $\hat{x} \in \Gamma(A_X)$ and $\hat{u} \in \Gamma(A_U)$, the thesis is proved.

□

6.8. Proof of Synthesis Algorithm Correctness

In this section we prove Theor. 6.1. This section can be skipped at a first reading.

PROOF THEOREM 6.1. If function *qCtrSyn* returns (SOL, $\hat{D}$, $\hat{K}$), then function *minCtrAbs* has found an admissible $\mathcal{Q}$ control abstraction $\hat{\mathcal{M}}$ of $\mathcal{H}$ (see Prop. 6.8) and function *strongCtr* has found the strong mgo $\hat{K}$ to the control problem ($\hat{\mathcal{M}}$, $\Gamma(I)$, $\Gamma(G)$). By Theor. 5.9 (point 1) the controller K, defined by $K(x, u) = (k(\Gamma(x)) = \Gamma(u))$ with k control law for $\hat{K}$, is a $\mathcal{Q}$ QFC strong solution to the control problem $(\mathcal{H}, I, G)$.

If function *qCtrSyn* returns (NOSOL, $\hat{D}$, $\hat{K}$), there is no weak solution to the control problem ($\hat{\mathcal{W}}$, $\Gamma(I)$, $\Gamma(G)$), where $\hat{\mathcal{W}}$ is the close to minimum full control abstraction of $\mathcal{H}$ computed by function *minFullCtrAbs* (Prop. 6.8). Therefore, by Theor. 5.9 (point 3) there is no $\mathcal{Q}$ QFC solution to the control problem $(\mathcal{H}, I, G)$. □

6.9. Details on Actions Admissibility Check

In this section we show how we can check for action admissibility. This section can be skipped at a first reading.

In Sect. 6.4, for any pair $(\hat{x}, \hat{u})$ in $\Gamma(A_X) \times \Gamma(A_U)$ line 4 of Alg. 2 checks if $\hat{u}$ is $\mathcal{Q}$-admissible in $\hat{x}$. This check is carried out by determining if the predicate $P(X, U, Y, X', \hat{x}, \hat{u}) \equiv N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge X' \notin A_X$ is not feasible.

Note that $X' \notin A_X$ is not a conjunctive predicate, thus feasibility of predicate $P(X, U, Y, X', \hat{x}, \hat{u})$ cannot be directly checked via function *feasible*. We implement such a check by calling $2|X|$ times function *feasible* in the following way. For each $x' \in X'$, let $P^{-}_{x'}(X, U, Y, X', \hat{x}, \hat{u}) \equiv N(X, U, Y, X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge x' \leq \inf A_x$ and

$P^+_{x'}(X,U,Y,X',\hat{x},\hat{u}) \equiv N(X,U,Y,X') \wedge \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge x' \geq \sup A_x$. For each $x' \in X'$, we call function *feasible* on $P^+_{x'}$ and $P^-_{x'}$ separately. If all such $2|X|$ calls return FALSE, then P is not feasible, otherwise P is feasible.

Note that by Def. 5.3 we should also check that $\forall x \in \Gamma^{-1}(\hat{x})\ \forall u \in \Gamma^{-1}(\hat{u})\ \exists x' \in \mathcal{D}_X\ \exists y \in \mathcal{D}_Y : N(x,u,y,x')$. This cannot be checked via function *feasible*. We therefore perform such a check by using a tool for quantifier elimination, namely Mjollnir [Monniaux 2010]. More in detail, we call Mjollnir only once, as a precomputation of Alg. 2, on the formula $\Phi(\hat{x},\hat{u}) \equiv \exists x \in \mathcal{D}_X\ \exists u \in \mathcal{D}_U\ \Gamma(X) = \hat{x} \wedge \Gamma(U) = \hat{u} \wedge \neg[\exists x' \in \mathcal{D}_X\ \exists y \in \mathcal{D}_Y : N(x,u,y,x')]$. The output of Mjollnir is a formula $\tilde{\Phi}(\hat{x},\hat{u})$ s.t. $\tilde{\Phi}(\hat{x},\hat{u}) \equiv \Phi(\hat{x},\hat{u})$ and $\tilde{\Phi}(\hat{x},\hat{u})$ does not contain quantifiers (i.e., the only variables in $\tilde{\Phi}(\hat{x},\hat{u})$ are $\hat{x}$ and $\hat{u}$). $\tilde{\Phi}(\hat{x},\hat{u})$ is true if $\hat{u}$ is not safe in $\hat{x}$. Since $\tilde{\Phi}(\hat{x},\hat{u})$ only depends on bounded discrete variables, we may turn it into an OBDD $\hat{L}$. This is the last step of the precomputation. Then, we use $\hat{L}$ as follows. Each time that function $\mathcal{Q}$-*admissible* (line 4 of Alg. 2) is invoked, it first checks if $(\hat{x},\hat{u}) \in \hat{L}$. If this holds, then function $\mathcal{Q}$-*admissible* directly returns FALSE. Otherwise, the above described check (involving at most $2|X|$ calls to function *feasible*) is performed.

7. CONTROL SOFTWARE GENERATION

In this section we describe how we synthesize the actual control software (C functions `Control_Law` and `Controllable_Region` in Sect. 1) and show how we compute its WCET. More details are given in [Mari et al. 2011a].

First, we note that given an OBDD B, we can easily generate a C function implementation *obdd2c*(B) for the boolean function (defined by) B by implementing in C the semantics of OBDD B. We do this by replacing each OBDD node with an `if-then-else` block and each OBDD edge with a `goto` instruction. Let $(\mu, \hat{D}, \hat{K})$ be the output of function *qCtrSyn* in Alg. 1. We synthesize function `Controllable_Region` by computing *obdd2c*($\hat{D}$). As for function `Control_Law`, let r (resp. n) be the number of bits used to represent plant actions (resp. states). We compute [Tronci 1998] a boolean function $F : \mathbb{B}^n \to \mathbb{B}^r$ that, for each quantized state $\hat{x}$ in the controllable region $\hat{D}$, returns a quantized action $\hat{u}$ such that $\hat{K}(\hat{x},\hat{u})$ holds. Let $F_i : \mathbb{B}^n \to \mathbb{B}$ be the boolean function computing the i-th bit of F. That is, $F(\hat{x}) = [F_1(\hat{x}), \ldots, F_r(\hat{x})]$. We take function `Control_Law` to be (the C implementation of) $[obdd2c(F_1), \ldots, obdd2c(F_r)]$.

7.1. Control Software WCET

We can easily compute the WCET for our control software. In fact all OBDDs we are considering have at most n variables. Accordingly, the execution of the resulting C code will go through at most n instruction blocks consisting essentially of an `if-then-else` and a `goto` statement. Let T_B be the time needed to compute one such a block on the microcontroller hosting the control software. Then we have that the WCET of `Controllable_Region` [`Control_Law`] is less than or equal to $n \cdot T_B$ [$r \cdot n \cdot T_B$]. Thus, neglecting I/O times, each iteration of the control loop (see Fig. 1) takes time (control software WCET) at most $(r+1) \cdot n \cdot T_B$. Note that a more strict upper bound for the WCET may be obtained by taking into account OBDDs heights (which are by construction at most n). The control loop (Fig. 1) poses the hard real time requirement that the control software WCET be less than or equal to the sampling time T. This is the case when WCET $\leq T$ holds. Such an equation allows us to know, before hand, the realizability of the foreseen control schema.

8. EXPERIMENTAL RESULTS

We implemented our QFC synthesis algorithm in C programming language, using GLPK to solve MILP problems and the CUDD package for OBDD based computations. We

Table I. Buck DC-DC converter (Sect. 3): control abstraction & controller synthesis results. Part I.

	Control Abstraction					Controller Synthesis	
b	CPU	MEM	Arcs	MaxLoops	LoopFrac	CPU	$\|K\|$
8	1.95e+03	4.41e+07	6.87e+05	2.55e+04	0.00333	2.10e-01	1.39e+02
9	9.55e+03	5.67e+07	3.91e+06	1.87e+04	0.00440	2.64e+01	3.24e+03
10	1.42e+05	8.47e+07	2.61e+07	2.09e+04	0.00781	7.36e+01	1.05e+04
11	8.76e+05	1.11e+08	2.15e+08	2.26e+04	0.01435	2.94e+02	2.88e+04

Table II. Buck DC-DC converter (Sect. 3): control abstraction & controller synthesis results. Part II.

	Total		
b	CPU	MEM	μ
8	1.96e+03	4.46e+07	Unk
9	9.58e+03	7.19e+07	Sol
10	1.42e+05	1.06e+08	Sol
11	8.76e+05	2.47e+08	Sol

name the resulting tool *Quantized feedback Kontrol Synthesizer* (QKS) (publicly available at [QKS Web Page 2011]).

Our methods focus on centralized control software synthesis problems. Therefore we focus our experimental results on such cases. Distributed control problems (such as TCAS [Platzer and Clarke 2009]), widely studied in a verification setting, are outside our scopes.

In this section we present our experiments that aim at evaluating effectiveness of: the control abstraction generation, the synthesis of OBDD representation of control law, and the control software size, performance, and guaranteed operational ranges (i.e. controllable region). In Sects. 8.1, 8.2, and 8.3 we present results for the *buck DC-DC converter* case study. In Sects. 8.4, 8.5, and 8.6 we shortly outline results for the *inverted pendulum* case study. Note that control software reaction time (WCET) is known a priori from Sect. 7.1 and its robustness to parameter variations in the controlled system as well as enforcement of safety bounds on state variables are an input to our synthesis algorithm (see Ex. 3.2 and Sect. 8.1).

8.1. Buck DC-DC Converter: Experimental Settings

In this section (and in Sects. 8.2, 8.3) we present experimental results obtained by using QKS on a version of the buck DC-DC converter described in Sect. 3.1. Further case studies (namely, the inverted pendulum and the multi-input buck DC-DC converter) can be found in [Alimguzhin et al. 2012a] and [Alimguzhin et al. 2012b]. We denote with $\mathcal{H} = (X, U, \tilde{Y}, \tilde{N})$ the DTLHS modeling such a converter, where X, U are as in Sect. 3.1. We set the parameters of $\mathcal{H}$ as follows: $T = 10^{-6}$ secs, $L = 2 \cdot 10^{-4}$ H, $r_L = 0.1\ \Omega$, $r_C = 0.1\ \Omega$, $R = 5 \pm 25\%\ \Omega$, $R_{off} = 10^4\ \Omega$, $C = 5 \cdot 10^{-5}$ F, $V_i = 15 \pm 25\%$ V. Thus, we require our controller to be robust to foreseen variations (25%) in the load (R) and in the power supply (V_i). To this aim, $\tilde{N}$ is obtained by extending N of Sect. 3.1 as follows. As for variations in the power supply V_i, they are modeled analogously to Ex. 3.2. As for variations in the load R, much more work is needed [Mari et al. 2011c] since $\mathcal{H}$ dynamics is not linear in R. For the sake of brevity, we simply point out that modeling variations in the load R requires 11 auxiliary boolean variables to be added to Y, thus obtaining $\tilde{Y}$, and 15 (guarded) constraints to be added to $\tilde{N}$.

For converters, *safety* (as well as physical) considerations set requirements on admissible values for state variables (admissible regions). We set $A_{i_L} = [-4, 4]$ and $A_{v_O} = [-1, 7]$. We

Table III. Buck DC-DC converter: number of MILPs and time to solve them (secs). Part I.

	$b = 8$			$b = 9$		
MILP	Num	Avg	Time	Num	Avg	Time
1	6.6e+04	7.0e-05	4.6e+00	2.6e+05	7.0e-05	1.8e+01
2	4.0e+05	1.5e-03	3.3e+02	1.6e+06	1.4e-03	1.1e+03
3	2.3e+05	9.1e-04	2.1e+02	9.2e+05	9.2e-04	8.4e+02
4	7.8e+05	9.9e-04	7.7e+02	4.4e+06	1.0e-03	4.5e+03
5	4.3e+05	2.8e-04	1.2e+02	1.7e+06	2.8e-04	4.9e+02

Table IV. Buck DC-DC converter: number of MILPs and time to solve them (secs). Part II.

	$b = 10$			$b = 11$		
MILP	Num	Avg	Time	Num	Avg	Time
1	1.0e+06	2.7e-04	2.8e+02	4.2e+06	2.3e-04	9.7e+02
2	6.4e+06	3.8e-03	1.3e+04	2.5e+07	3.3e-03	4.6e+04
3	3.7e+06	3.0e-03	1.1e+04	1.5e+07	2.6e-03	3.8e+04
4	3.0e+07	2.6e-03	7.8e+04	2.6e+08	2.2e-03	5.7e+05
5	6.8e+06	1.8e-03	1.3e+04	2.7e+07	1.6e-03	4.2e+04

define $A = A_{i_L} \times A_{v_O} \times A_u$. As for auxiliary variables, we use the following safety bounds: $A_{i_u} = A_{i_D} = [-10^3, 10^3]$ and $A_{v_u} = A_{v_D} = [-10^7, 10^7]$. As a result, we add 12 further constraints to $\tilde{N}$ stating that $\bigwedge_{w \in \{i_L, v_O, i_u, i_D, v_u, v_D\}} w \in A_w$, thus obtaining a bounded DTLHS [Mari et al. 2011c].

Finally, the initial region I and goal region G are as in Ex. 4.7, thus the DTLHS control problem we consider is $P = (\mathcal{H}, I, G)$. Note that no (formally proved) robust control software is available for buck DC-DC converters.

We use a uniform quantization dividing the domain of each state variable (i_L, v_O) into 2^b equal intervals, where b is the number of bits used by AD conversion, thus w.r.t. Sect. 6.3 we have that $\Delta_{i_L} = \Delta_{v_O} = 2^b$. The resulting quantization is $\mathcal{Q}_b = (A, \Gamma_b)$, with $\|\Gamma_b\| = 2^{3-b}$. Since we have two quantized variables (i_L, v_O) each one with b bits, the number of states in the control abstraction is exactly 2^{2b}.

For each value of interest for b, we run QKS, and thus Alg. 1, on the control problem $(\mathcal{H}, I, G)$ with quantization $\mathcal{Q}_b$. In the following, we will call $\hat{\mathcal{M}}_b$ the close to minimum (admissible) $\mathcal{Q}_b$ control abstraction for $\mathcal{H}$, $\hat{\mathcal{H}}_b$ the maximum (full) $\mathcal{Q}_b$ control abstraction for $\mathcal{H}$ (which we compute for statistical reasons also when Alg. 1 returns Sol), $\hat{K}_b$ the strong mgo for $\hat{\mathcal{P}}_b = (\hat{\mathcal{M}}_b, \varnothing, \Gamma_b(G))$, $\hat{D}_b = \mathrm{Dom}(\hat{K}_b)$ the controllable region of $\hat{K}_b$, and $K_b(s, u) = \hat{K}_b(\Gamma_b(s), \Gamma_b(u))$ the $\mathcal{Q}_b$ QFC solution to $\mathcal{P}_b = (\mathcal{H}, \Gamma_b^{-1}(\hat{D}_b), G)$. All our experiments have been carried out on a 3.0 GHz Intel hyperthreaded Quad Core Linux PC with 8 GB of RAM.

8.2. Buck DC-DC Converter: QKS Performance

In this section we will show the performance (in terms of computation time and memory) of algorithms discussed in Sect. 6.

Tabs. I, II, III and IV show our experimental results for QKS (and thus for Alg. 1). Columns in Tab. I have the following meaning. Column b shows the number of AD bits. Columns labeled *Control Abstraction* show performance for Alg. 2 (computation of $\hat{\mathcal{M}}_b$) and they show running time (column *CPU*, in secs), memory usage (*MEM*, in bytes), the number of transitions in $\hat{\mathcal{M}}_b$ (*Arcs*), the number of self loops in $\hat{\mathcal{H}}_b$ (*MaxLoops*), and the fraction of self loops that are kept in $\hat{\mathcal{M}}_b$ w.r.t. the number of self loops in $\hat{\mathcal{H}}_b$ (*LoopFrac*).

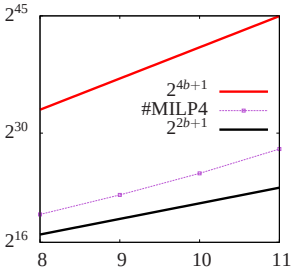


Fig. 7. Buck DC-DC Converter: Number of MILP4 calls.

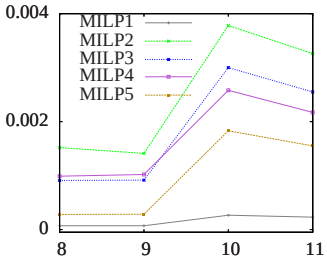


Fig. 8. Buck DC-DC Converter: Average of MILP calls.

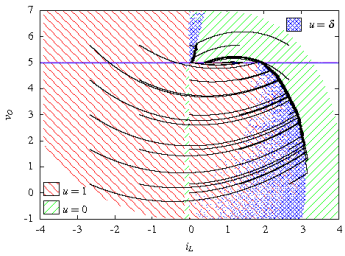


Fig. 9. Buck DC-DC Converter: Controlled region with $b = 10$ bits.

Columns labeled *Controller Synthesis* show the computation time (column *CPU*, in secs) for the generation of $\hat{K}_b$, and the size of its OBDD representation ($|K|$, number of nodes). The latter is also the size (number of lines) of $\hat{K}_b$ C code synthesized implementation. Columns in Tab. II have the following meaning. Column b shows the number of AD bits. Columns labeled *Total* show the total computation time (column *CPU*, in secs) and the memory (*MEM*, in bytes) for the whole process (i.e., control abstraction plus controller source code generation), as well as the final outcome $\mu \in \{\text{SOL}, \text{NOSOL}, \text{UNK}\}$ of Alg. 1.

From Tabs. I and II we see that computing control abstractions (i.e. Alg. 2) is the most expensive operation in QKS and that thanks to function *SelfLoop* $\hat{\mathcal{M}}_b$ contains no more than 2% of the loops in $\hat{\mathcal{H}}_b$.

8.2.1. MILP problems Analysis. For each MILP problem solved in QKS, Tabs. III and IV show (as a function of b) the total and the average CPU time (in seconds) spent solving MILP problems, together with the number of MILP problems solved, divided by different kinds of MILP problems as follows. MILP1 refers to the MILP problems described in Sect. 6.3, i.e. those computing the quantization for I and G, MILP2 refers to MILP problems in function *SelfLoop* (see Alg. 3), MILP3 refers to the MILP problems used in function *overImg* (line 6 of Alg. 2), MILP4 refers to MILP problems used to check actions admissibility (line 8 of Alg. 2), and MILP5 refers to MILP problems used to check transitions witnesses (line 4 of Alg. 2). Columns in Tabs. III and IV have the following meaning: *Num* is the number of times that the MILP problem of the given type is called, *Time* is the total CPU time (in secs) needed to solve all the *Num* instances of the MILP problem of the given type, and *Avg* is the average CPU time (in secs), i.e. the ratio between columns *Time* and *Num*.

CPU time standard deviation is always less than 0.003.

Fig. 7 graphically shows (as a function of b) the number of MILP4 instances solved (column *Num* of columns group MILP4 in Tabs. III and IV).

From Tabs. III and IV, column *Avg*, we see that the average time spent solving each MILP instance is small. Fig. 8 graphically shows that MILP average computation time does not heavily depend on b. As observed in Remark 6.5, Fig. 7 shows that the number of MILP4 invocations is much closer to $|\Gamma(A_X)||\Gamma(A_U)| = 2^{2b+1}$, rather than the theoretical worst case running time $|\Gamma(A_X)|^2|\Gamma(A_U)| = 2^{4b+1}$ of Alg. 2. This shows effectiveness of function *overImg* heuristic.

8.3. Buck DC-DC Converter: Control Software Performance

In this section we discuss the performance of the generated controller. Fig. 10 shows a snapshot of the QKS synthesized control software for the Buck DC-DC converter when 10 bits ($b = 10$) are used for AD conversion.

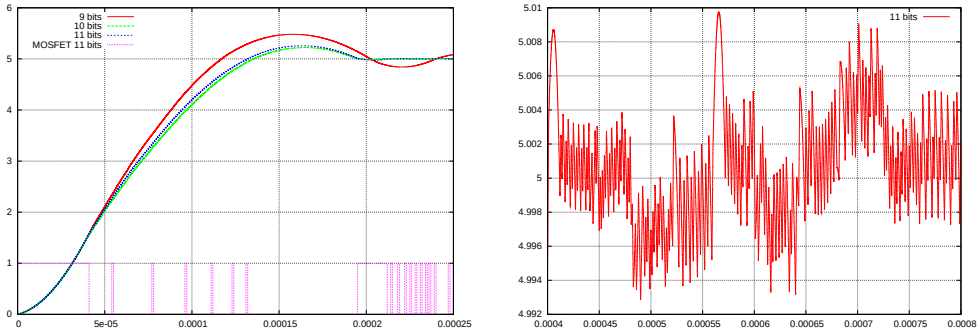
8.3.1. Controllable Region. One of the most important features of our approach is that it returns the guaranteed operational range (precondition) of the synthesized software (Theor.

```
int Controllable_Region(int *x) {
  int ret_b = 0;
  L_2af64a1: if (x[2] == 1) goto L_2b001e0;
             else { ret_b = !ret_b; goto L_2afff40; }
  L_21f95e0: return ret_b;
  L_2b07f00: if (x[14] == 1) goto L_21f95e0;
             else goto L_2b07ee0;
  /* ... */
}

int Control_Law(int *x, int *u) {
  int i;
  for(i = 0; i < 1; i++)
    u[i] = Aux_Bits(x,i);
  return 0;
}

int Aux_Bits(int *x, int b) {
  int ret_b;
  switch(b){ case 0: ret_b = 0; goto L_2af6081; }
  L_2af6081: if (x[2] == 1) goto L_2a6d2e0;
             else { ret_b = !ret_b; goto L_2af6060; }
  L_21f95e0: return ret_b;
  /* ... */
}
```

Fig. 10. A snapshot of the synthesized control software for the Buck DC-DC converter with 10 bit AD conversion.



(a) v_O from $i_L = 0, v_O = 0$ (b) *Ripple* for v_O $(b = 11)$

Fig. 11. Controller performances for the Buck DC-DC Converter: setup time and ripple.

6.1). This is the controllable region $\hat{D}$ returned by Alg. 1. In our case study, 9 bit turns out to be enough to have a controllable region that covers the initial region [Mari et al. 2011c]. Increasing the number of bits, we obtain even larger controllable regions. Fig. 9 shows the controllable region $D_{10} = \Gamma_{10}^{-1}(\hat{D}_{10})$ for K_{10} along with some trajectories (with time increasing counterclockwise) for the closed loop system. We see that the initial region $I \subseteq D_{10}$. Thus we know (on a formal ground) that 10 bit AD conversion suffices for our purposes. More details on controllable region visualization can be found in [Mari et al. 2012a].

8.3.2. Setup Time and Ripple. Our model based control software synthesis approach presently does not handle *quantitative liveness specifications*. Accordingly, quantitative system level formal specifications have to be verified a posteriori. This can be done using a classical *Hardware-In-the-Loops* (HIL) simulation approach or, even better, following a formal ap-

proach, as discussed in [Henzinger 2010; Hermanns et al. 2010]. In our context HIL simulation is quite easy since we already have a DTLHS model for the plant and the control software is generated automatically.

To illustrate such a point in this section we highlight HIL simulation results for two quantitative specifications typically considered in control systems: *Setup Time* and *Ripple*.

The setup time measures the time it takes to reach the goal (steady state) when the system is turned on. Fig. 11(a) shows trajectories starting from point $(0, 0)$ for K_9, K_{10} and K_{11} as well as the control command sent to the MOSFET (square wave in Fig. 11(a)) for K_{11}. Note that all trajectories stabilize (steady state) after only 0.0003 secs (setup time).

The ripple measures the wideness of the oscillations around the goal (steady state) once this has been reached. Fig. 11(b) shows the ripple for the output voltage after stabilization. For K_{11} we see that the ripple is about 0.01 V, that is 0.2% of the reference value $V_{\text{ref}} = 5$ V.

It is worth noticing that both setup time and ripple compare well with typical figures of commercial high-end buck DC-DC converters (e.g. see [Texas Instruments 2001]) and with the results available from the literature (e.g. [So et al. 1996; Yousefzadeh et al. 2008]).

8.4. Inverted Pendulum: Experimental Settings

In this section (and in Sects. 8.5, 8.6) we present experiment results obtained by using QKS on the inverted pendulum described in [Kreisselmeier and Birkhölzer 1994], as shown in Fig. 12. The system is modeled by taking the angle θ and the angular velocity $\dot{\theta}$ as state variables. The input of the system is the torquing force u, that can influence the velocity in both directions. Moreover, the behaviour of the system depends on the pendulum mass m, the length of the pendulum l and the gravitational acceleration g. Given such parameters, the motion of the system is described by the differential equation $\ddot{\theta} = \frac{g}{l}\sin\theta + \frac{1}{ml^2}u$.

In order to obtain a state space representation, we consider the following normalized system, where x_1 is the angle θ and x_2 is the angular speed $\dot{\theta}$.

$$\dot{x}_1 = x_2 \tag{11}$$

$$\dot{x}_2 = \frac{g}{l}\sin x_1 + \frac{1}{ml^2}u \tag{12}$$

Differently from [Kreisselmeier and Birkhölzer 1994], we consider the problem of finding a discrete controller, whose decisions may be "apply the force clockwise" ($u = 1$), "apply the force counterclockwise" ($u = -1$), or "do nothing" ($u = 0$). The intensity of the force will be given as a constant F. Finally, the discrete time transition relation N is obtained from

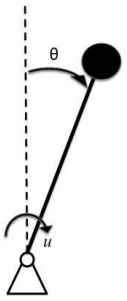


Fig. 12. Inverted Pendulum with Stationary Pivot Point.

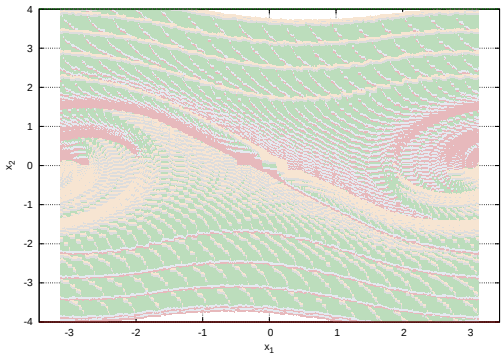


Fig. 13. Inverted Pendulum: Controlled region with $b = 9$ bits, $F = 0.5$, $T = 0.1$.

Table V. Inverted Pendulum: control abstraction & controller synthesis results with $F = 0.5$.

b	T	ρ	$\lvert K\rvert$	CPU	MEM
8	0.1	0.1	2.73e+04	2.56e+03	7.72e+04
9	0.1	0.1	5.94e+04	1.13e+04	1.10e+05
10	0.1	0.1	1.27e+05	5.39e+04	1.97e+05
11	0.01	0.05	4.12e+05	1.47e+05	2.94e+05

the equations (11-12) as the Euler approximation with sampling time T, i.e. the predicate $(x_1' = x_1 + Tx_2) \wedge (x_2' = x_2 + T\frac{g}{l}\sin x_1 + T\frac{1}{ml^2}Fu)$.

Since the system whose dynamics are in equations (11-12) is not linear, we build a linear over-approximation of it as shown in [Alimguzhin et al. 2012a]. The result is the DTLHS $\mathcal{H}$ defined in Ex. 5 of [Alimguzhin et al. 2012a]. From now on we use $\mathcal{H}$ to denote the inverted pendulum system.

In all our experiments, as in [Kreisselmeier and Birkhölzer 1994], we set parameters l and m in such a way that $\frac{g}{l} = 1$ (i.e. $l = g$) and $\frac{1}{ml^2} = 1$ (i.e. $m = \frac{1}{l^2}$). Moreover, we set the force intensity $F = 0.5$. More experiments can be found in [Alimguzhin et al. 2012a].

As we have done for the buck DC-DC converter, we use uniform quantization functions dividing the domain of each state variable $\mathcal{D}_{x_1} = [-1.1\pi, 1.1\pi]$ (we write π for a rational approximation of it) and $\mathcal{D}_{x_2} = [-4, 4]$ into 2^b equal intervals, where b is the number of bits used by AD conversion. Since we have two quantized variables, each one with b bits, the number of quantized states is exactly 2^{2b}.

The typical goal for the inverted pendulum is to turn the pendulum steady to the upright position, starting from any possible initial position, within a given speed interval. In our experiments, the goal region is defined by the predicate $G(X) \equiv (-\rho \leq x_1 \leq \rho) \wedge (-\rho \leq x_2 \leq \rho)$, where $\rho \in \{0.05, 0.1\}$, and the initial region is defined by the predicate $I(X) \equiv (-\pi \leq x_1 \leq \pi) \wedge (-4 \leq x_2 \leq 4)$.

We run QKS on the control problem $(\mathcal{H}, I, G)$ for different values of the remaining parameters, i.e. ρ (goal tolerance), T (sampling time), and b (number of bits of AD). For each of such experiments, QKS outputs a control software K in C language. In the following, we sometimes make explicit the dependence on b by writing K_b. In order to evaluate performance of K, we use an *inverted pendulum simulator* written in C. The simulator computes the next state by using Eqs. (11-12), thus simulating a path of $\mathcal{H}^{(K)}$. Such simulator also introduces random disturbances (up to 4%) in the next state computation to assess K robustness w.r.t. non-modeled disturbances. Finally, in the simulator Eqs. (11-12) are translated into the discrete time version by means of a simulation time step T_s much smaller than the sampling time T used in $\mathcal{H}$. Namely, $T_s = 10^{-6}$ seconds, whilst $T = 0.01$ or $T = 0.1$ seconds. This allows us to have a more accurate simulation. Accordingly, K is called each 10^4 (or 10^5) simulation steps of $\mathcal{H}$. When K is not called, the last chosen action is selected again (*sampling and holding*).

All experiments for the inverted pendulum have been carried out on an Intel(R) Xeon(R) CPU @ 2.27GHz, with 23GiB of RAM, Debian GNU/Linux 6.0.3 (squeeze).

8.5. Inverted Pendulum: QKS Performance

To stabilize an *underactuated* inverted pendulum (i.e. $F < 1$) from the hanging position to the upright position, a controller needs to find a non obvious strategy that consists of swinging the pendulum once or more times to gain enough momentum. QKS is able to synthesize such a controller taking as input $\mathcal{H}$ with $F = 0.5$ (note that in [Kreisselmeier and Birkhölzer 1994] $F = 0.7$). Results are in Tab. V, where each row corresponds to a QKS run, columns b, T and ρ show the corresponding inverted pendulum parameters, column $|K|$ shows the size of the C code for K_b, and columns CPU and

MEM show the computation time (in seconds) and RAM usage (in KB) needed by QKS to synthesize K_b.

8.6. Inverted Pendulum: Control Software Performance

As for K_b performance, it is easy to show that by reducing the sampling time T and the quantization step (i.e. increasing b), we increase the quality of K_b in terms of ripple and set-up time. Fig. 14(a) shows the simulations of $\mathcal{H}^{(K_9)}$ and $\mathcal{H}^{(K_{10})}$. As we can see, K_{10} drives the system to the goal with a smarter trajectory, with one swing only. This have a significant impact on the set-up time (the system stabilizes after about 8 seconds when controlled by K_{10} instead of about 10 seconds required when controlled by K_9). Fig. 13 shows that the controllable region of K_9 covers almost all states in the admissible region that we consider. Different colors mean different set of actions enabled by the controller. Finally, Fig. 14(b) shows the ripple of x_1 for $\mathcal{H}^{(K_{10})}$ inside the goal. Note that such ripple is very low (0.018 radiants).

9. RELATED WORK

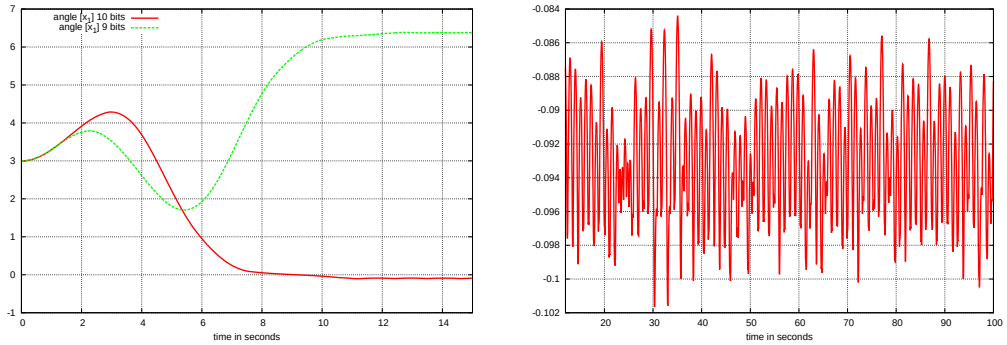
This paper is a journal version of [Mari et al. 2010] which is extended here by providing omitted proofs and algorithms.

Sect. 9.1 compares our contribution with related work on control software synthesis from system level formal specifications. For the sake of completeness, Sect. 9.2 expands such a comparison to recent results on (non control) software synthesis from formal specifications, focusing on papers using techniques related to ours (constraint solving, OBDD, supervisory control [Ramadge and Wonham 1987]). Sect. 9.3 describes Tab. VI, which summarizes the novelty of our contribution with respect to automatic methods for control software synthesis.

9.1. Control Software Synthesis from System Level Formal Specifications

Control Engineering has been studying control law design (e.g., optimal control, robust control, etc.) for more than half a century (e.g., see [Brogan 1991]). As explained in Sect. 1.1 such results cannot be directly used in our (formal) software synthesis context. On the other hand we note that there are many control systems that are not software based (e.g., in analog circuit design). In such cases, of course, our approach cannot be used.

9.1.1. Control of Linear and Switched Hybrid Systems. The paper closer to our is [Kreisselmeier and Birkhölzer 1994] which studies the problem of control synthesis for discrete time hybrid systems. However, while we present an automatic method, the approach



(a) x_1 from $x_1 = \pi$, $x_2 = 0$ (b) Ripple for x_1 ($b = 10$)

Fig. 14. Controller performances for the Inverted Pendulum with $F = 0.5$, $b = 9, 10$: setup time and ripple.

in [Kreisselmeier and Birkhölzer 1994] is not automatic since it requires the user to provide a suitable Lyapunov function (a far from trivial task even for linear hybrid systems).

Quantized Feedback Control has been widely studied in control engineering (e.g. see [Fu and Xie 2005]). However such research addresses linear systems (not the general case of hybrid systems) and focuses on control law design rather than on control software synthesis (our goal). Furthermore, all control engineering approaches model *quantization errors* as statistical *noise*. As a result, correctness of the control law holds in a probabilistic sense. Here instead, we model quantization errors as nondeterministic (*malicious*) *disturbances*. This guarantees system level correctness of the generated control software (not just that of the control law) with respect to *any* possible sequence of quantization errors.

Control software synthesis for continuous time linear systems has been widely studied (e.g. see [Brogan 1991]). However such research does not account for quantization. Control software synthesis for continuous time linear systems with quantization has been investigated in [Mazo and Tabuada 2011]. This paper presents an automatic method which, taking as input a continuous time linear system and a goal specification, produces a control law (represented as an OBDD) through PESSOA [Mazo et al. 2010]. While [Mazo and Tabuada 2011] applies to (continuous time) linear systems, our contribution focuses on (discrete time) linear *hybrid* systems (DTLHSs). Furthermore, although taking into account the quantization process, [Mazo and Tabuada 2011] does not supply an effective method to generate control software (as we do in Sect. 6.1). As a consequence [Mazo and Tabuada 2011] gives no guarantee on WCET, an important issue since an SBCS is a hard real-time system.

[Girard et al. 2010] presents a method to find an over-approximation of *switched systems*, under certain stability hypotheses. A switched system is a hybrid system whose mode transitions only depend on control inputs. Such a line of research goes back to [Pola et al. 2007] which presents a method to compute symbolic models for nonlinear control systems. In combination with [Mazo and Tabuada 2011], such results provide a semi-automatic method for the construction of a control law for switched and nonlinear systems. However, we note that nonlinear systems in [Pola et al. 2007] are not hybrid systems, since they cannot handle discrete variables. Moreover, while a switched system as in [Girard et al. 2010] is a linear hybrid system the converse is false since in a linear hybrid system mode transitions can be triggered by state changes (without any change in the input). For example, our approach can synthesize controllers both for the buck DC-DC converter of Fig. 3 (a linear hybrid system) and for the boost DC-DC converter in [Girard et al. 2010] (a switched system). However, the approach in [Girard et al. 2010] cannot handle the buck DC-DC converter of Fig. 3 because of the presence of the diode which triggers state dependent mode changes. Moreover, [Mazo and Tabuada 2011] combined with [Girard et al. 2010] and [Pola et al. 2007] provide semi-automatic methods since they rely on a Lyapunov function provided by the user, much in the spirit of [Kreisselmeier and Birkhölzer 1994].

9.1.2. Control of Timed Automata and Linear Hybrid Automata. When the plant model is a *Timed Automaton* (TA) [Alur and Madhusudan 2004; Maler et al. 1992] the reachability and control law synthesis problems have both been widely studied. Examples are in [Larsen et al. 1997; Cassez et al. 2005; Maler et al. 2007; Asarin and Maler 1999; Peter et al. 2011] and citations thereof. When the plant model is a *Linear Hybrid Automaton* (LHA) [Alur et al. 1995; Alur et al. 1996] reachability and existence of a control law are both undecidable problems [Henzinger and Kopke 1997; Henzinger et al. 1998]. This, of course, has not prevented devising effective (semi) algorithms for such problems. Examples are in [Alur et al. 1996; Henzinger et al. 1997; Frehse 2008; Wong-Toi 1997; Benerecetti et al. 2011]. Much in the same spirit here we give necessary and sufficient constructive conditions for control software existence. Note that none of the above mentioned papers address control software synthesis since they all

assume *exact* (i.e. real valued) state measures (that is, state feedback quantization is not considered).

Continuous-time linear hybrid systems with time delays and given precision of state measurement, *Lazy Linear Hybrid Automaton* (LLHA), have been studied in [Agrawal and Thiagarajan 2005]. The reachability problem is shown to be undecidable for LLHAs in [Agrawal et al. 2006]. Note that such results do not directly apply to our context (Theorem 4.16) since we are addressing discrete-time systems.

9.1.3. Control of Piecewise Affine and Nonlinear Hybrid Systems. Finite horizon control of *Piecewise Affine Discrete Time Hybrid Systems* (PWA-DTHS) has been studied using a MILP based approach. See, for example, [Bemporad and Giorgetti 2004]. PWA-DTHSs form a strict subclass of DTLHSs since PWA-DTHS cannot handle linear constraints consisting of discrete state variables whereas DTLHSs can. Such approaches cannot be directly used in our context since they address synthesis of finite horizon controllers and do not account for quantization.

Much in the spirit of [Kreisselmeier and Birkhölzer 1994], [Della Penna et al. 2008] presents an explicit control synthesis algorithm for discrete time (possibly nonlinear) hybrid systems, by avoiding the needs of providing Lyapunov functions. Moreover, [Della Penna et al. 2009] presents control synthesis algorithms for discrete time hybrid systems cast as universal planning problems. Such approaches cannot be directly used in our context since they do not account for quantization.

Hybrid Toolbox [Bemporad 2004] considers continuous time piecewise affine systems. Such a tool outputs a feedback control law that is then passed to Matlab in order to generate control software. We note that such an approach does not account for state feedback quantization and thus, as explained in Sect. 1.1, does not offer any formal guarantee about system level correctness of the generated software, which is instead our focus here.

Using the engine proposed in this paper and computing suitable over-approximations of nonlinear functions, it is possible to address synthesis for nonlinear hybrid systems as done in [Alimguzhin et al. 2012a].

9.1.4. Software Synthesis in a Finite Setting. Correct-by-construction software synthesis in a finite state setting has been studied, for example, in [Tronci 1997; Tronci 1998; Tronci 1999b; Tronci 1999a]. An automatic method for the generation of supervisory controllers for finite state systems is presented in [Tronci 1996]. Control software synthesis in non-deterministic finite domains is studied in [Cimatti et al. 1998] (cast as a universal planning problem). Such approaches cannot be directly used in our context since they cannot handle continuous state variables.

In Sect. 6.1 we presented our QFC synthesis algorithm (Alg. 1). Line 3 of Alg. 1 calls function *strongCtr* (implementing a variant of the algorithm in [Cimatti et al. 1998]) in order to compute a time optimal controller for the finite state quantized system. [Alimguzhin et al. 2012b] presents a method to obtain a compressed non time optimal controller for a finite state system. This is done by trading the size of the synthesized controller with time optimality while preserving closed loop performances (Remark 6.3). Such a method can be implemented in function *strongCtr*. Thus, [Alimguzhin et al. 2012b] is not an improvement to the present paper but it is a contribution on controller synthesis for finite state systems.

9.1.5. Switching Logic. Optimal switching logic for hybrid systems has been also widely investigated. For example, see [Taly et al. 2009; Jha et al. 2010; Jha et al. 2011] and citations thereof. Such approaches, by ignoring the quantization process, indeed focus on the control law design (see Sect. 1.1). However we note that [Jha et al. 2010; Jha et al. 2011] address dwell-time and optimality issues which are not covered by our approach.

9.1.6. Abstraction. Quantization can be seen as a sort of abstraction (the reason for the name *control abstraction*), which has been widely studied in a hybrid system formal verification context (e.g., see [Alur et al. 2000; Alur et al. 2006; Tiwari 2008; Sankaranarayanan and Tiwari 2011]). Note however that in a verification context abstractions are designed so as to ease the verification task whereas in our setting quantization is a design requirement since it models a hardware component (AD converter) which is part of the specification of the control software synthesis problem. Indeed, in our setting, we have to design a controller *notwithstanding* the nondeterminism stemming from the quantization process. As a result, the techniques used to devise clever abstractions in a verification setting cannot be directly used in our synthesis setting where the quantization to be used is given.

9.2. Software Synthesis from Formal Specifications

Much as control software synthesis, also software synthesis has been widely studied since a long time in many contexts. For examples, see [Pnueli and Rosner 1989a; Pnueli and Rosner 1989b; Schewe and Finkbeiner 2006; Girault and Rutten 2009]. We give a glimpse of recent results on (non control) software synthesis approaches using techniques related to ours (constraint solving, OBDD, supervisory control).

[Attie et al. 2004] shows how to mechanically synthesize fault-tolerant concurrent programs for various fault classes. [Srivastava et al. 2010] presents a method that synthesizes a program, if there exists one, that meets the input/output specification and uses only the given resources. [Gulwani et al. 2011] addresses the problem of synthesizing loop-free programs starting from logical relations between input and output variables. [Srivastava et al. 2011] proposes a synthesis technique and applies it to the inversion of imperative programs (e.g., such as insert/delete operations, compressors/decompressors). [Cerný et al. 2011] presents a method for the quantitative, performance-aware synthesis of concurrent programs. Procedures and tools for the automated synthesis of code fragments are also proposed in [Kuncak et al. 2012; Gvero et al. 2011; Kuncak et al. 2010].

Such approaches build on techniques (constraint solving, OBDD, supervisory control) related to ours, but do not address control software synthesis from system level formal specifications.

9.3. Summary

Tab. VI summarizes the novelty of our contribution with respect to automatic methods for control software synthesis (our focus here). For this reason, it only considers papers addressing control software synthesis. Namely, those in Sect. 9.1 but the ones focusing on abstraction (since Sect. 9.2 results do not address control software synthesis).

Tab. VI is organized as follows. Each row refers to a citation. Each column represents a feature of a cited work. A bullet in a cell means that the citation in the cell row has the feature in the cell column. Where the feature is missing, the cell is empty. The group of columns labeled T denotes whether the input model is expressed in *continuous time* or *discrete time*. The group of columns labeled *Input System* lists the kind of input models we are interested in. Namely: *finite state*, *linear*, *switched*, *piecewise affine*, *TA or LHA*, *linear hybrid sys.*, *nonlinear*, *nonlinear hybrid sys.* Note that the combination of columns *linear hybrid sys.* and *discrete time* denotes our class of DTLHSs. The column labeled *Quantization* denotes that the row supplies the quantization process. The group of columns labeled K lists the output controller characteristics we are interested in. In particular: *Formally verified* denotes if the output controller is guaranteed to satisfy the given input specification; *Control software* indicates if the presented method outputs a control software implementation; *Guaranteed WCET* denotes if the output controller has a guaranteed WCET. Finally, the group of columns labeled *Impl* considers implementation issues, namely if a method is *fully automatic* or *semi automatic*, and if there exists a *tool available* implementing the

Table VI. Summary of Related Work. '•' stands for 'Yes'. An empty cell means that feature is not supported.

Citation	*T*		*Input System*										*K*			*Impl*		
	Continuous time	Discrete time	Finite state	Linear	Switched	TA, LHA	Piecewise affine	Linear hybrid sys.	Nonlinear	Nonlinear hybrid sys.	Quantization	Formally verified	Control Software	Guaranteed WCET	Fully automatic	Semi automatic	Tool available
[Mari et al. 2010] and this paper		•		•	•	•	•	•			•	•	•	•	•		•
[Alimguzhin et al. 2012a]		•								•		•	•	•	•		•
[Alimguzhin et al. 2012b]		•	•									•	•	•	•		•
[Asarin and Maler 1999]	•					•						•			•		
[Bemporad 2004]	•						•						•		•		•
[Bemporad and Giorgetti 2004]		•					•								•		
[Benerecetti et al. 2011]	•					•						•			•		•
[Cassez et al. 2005]	•					•						•			•		
[Cimatti et al. 1998]		•	•									•	•		•		•
[Della Penna et al. 2008]		•		•	•	•	•	•	•	•					•		•
[Della Penna et al. 2009]		•		•	•	•	•	•							•		•
[Fu and Xie 2005]	•			•							•						
[Girard et al. 2010]	•				•						•	•				•	
[Jha et al. 2010]	•				•							•			•		
[Jha et al. 2011]	•				•							•			•		
[Kreisselmeier and Birkhölzer 1994]		•		•					•		•	•				•	
[Larsen et al. 1997]	•					•						•			•		
[Maler et al. 2007]	•					•						•			•		
[Mazo and Tabuada 2011]	•			•							•	•			•		•
[Peter et al. 2011]	•					•						•	•		•		•
[Pola et al. 2007]	•								•		•	•				•	
[Taly et al. 2009]	•				•							•			•		
[Tronci 1996]		•	•									•	•		•		
[Tronci 1997]		•	•									•	•		•		
[Tronci 1998]		•	•									•	•		•		
[Tronci 1999b]		•	•									•	•		•		
[Tronci 1999a]		•	•									•	•		•		
[Wong-Toi 1997]	•					•						•			•		

presented method. Note that [Girard et al. 2010] and [Pola et al. 2007] in Tab. VI represent their combination with [Mazo and Tabuada 2011].

Summing up, to the best of our knowledge, no previously published result is available about *fully automatic* generation (with a *tool available*) of *correct-by-construction control software* with a *guaranteed WCET* from a DTLHS model of the plant, *system level formal specifications* and *implementation specifications* (*quantization*, that is number of bits in AD conversion).

10. CONCLUSIONS

We presented an algorithm and a tool QKS implementing it, to support a *Formal Model Based Design* approach to control software. Our tool takes as input a formal DTLHS model of the plant, *implementation specifications* (namely, number of bits in AD conversion), and *system level formal specifications* (namely, safety and liveness properties for the closed loop system). It returns as output a correct-by-construction C implementation (if any) of the control software (namely, `Control_Law` and `Controllable_Region`) with a WCET guaranteed to be linear in the number of bits of the quantization schema. We have shown feasibility

of our proposed approach by presenting experimental results on using it to synthesize C controllers for the buck DC-DC converter and the inverted pendulum.

In order to speed-up the computation and to avoid possible numerical errors due to MILP solvers [Neumaier and Shcherbina 2004], a natural possible future research direction is to investigate fully symbolic control software synthesis algorithms based on efficient *quantifier elimination* procedures (e.g., see [Monniaux 2010] and citations thereof).

ACKNOWLEDGMENT

We gratefully acknowledge partial support from FP7 projects GA218815 (ULISSE), 317761 (SmartHG), 600773 (PAEON) and MIUR project DM24283 (TRAMP).

ACRONYMS

AD. Analog-to-Digital. 1–3, 9, 10, 19, 21, 25, 26, 43
COBDD. OBDD with complemented edges. 45–48
DA. Digital-to-Analog. 1, 2, 9
DTLHS. Discrete Time Linear Hybrid System. 3, 5–19, 22–26, 34–38, 40, 41, 43, 46, 48–50, 52
DVFS. Dynamic Voltage and Frequency Scaling. 6
HIL. Hardware-In-the-Loop. 22
LHA. Linear Hybrid Automaton. 23, 25
LLHA. Lazy Linear Hybrid Automaton. 24
LTS. Labeled Transition System. 5–8, 10–15, 31, 33, 34, 38–41, 46, 52
MILP. Mixed Integer Linear Programming. 3–5, 15, 16, 18–21, 24, 27, 43
NDTCM. Non-Deterministic Two-Counter Machine. 36
OBDD. Ordered Binary Decision Diagram. 14, 15, 17–19, 22, 23, 25, 33, 44, 45
PWA-DTHS. Piecewise Affine Discrete Time Hybrid Systems. 24
QFC. Quantized Feedback Control. 10, 12–15, 18, 19, 24, 34, 37, 43
QKS. Quantized feedback Kontrol Synthesizer. 3, 18–21, 26
SBCS. Software Based Control System. 1, 2, 23
TA. Timed Automaton. 23, 25
WCET. Worst Case Execution Time. 2, 3, 17, 18, 23, 25, 26, 46, 48

REFERENCES

Agrawal, M., Stephan, F., Thiagarajan, P. S., and Yang, S. 2006. Behavioural approximations for restricted linear differential hybrid automata. In *Proceedings of Hybrid Systems: Computation and Control, 9th International Workshop, HSCC*, J. P. Hespanha and A. Tiwari, Eds. Lecture Notes in Computer Science Series, vol. 3927. Springer, 4–18.

Agrawal, M. and Thiagarajan, P. S. 2005. The discrete time behavior of lazy linear hybrid automata. In *Proceedings of Hybrid Systems: Computation and Control, 8th International Workshop, HSCC*, M. Morari and L. Thiele, Eds. Lecture Notes in Computer Science Series, vol. 3414. Springer, 55–69.

Alimguzhin, V., Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2012a. Automatic control software synthesis for quantized discrete time hybrid systems. In *Conference on Decision and Control, CDC*. To Appear. A preliminary version can be found at `http://arxiv.org/abs/1207.4098`.

Alimguzhin, V., Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2012b. On model based synthesis of embedded control software. In *International Conference on Embedded Software, EMSOFT*. To Appear. A preliminary version can be found at `arxiv.org`.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P. H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138,* 1, 3 – 34.

Alur, R., Dang, T., and Ivančić, F. 2006. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. on Embedded Computing Sys. 5,* 1, 152–199.

Alur, R., Henzinger, T., Lafferriere, G., and Pappas, G. 2000. Discrete abstractions of hybrid systems. *Proceedings of the IEEE 88,* 7, 971–984.

Alur, R., Henzinger, T. A., and Ho, P.-H. 1996. Automatic symbolic verification of embedded systems. *IEEE Trans. Softw. Eng. 22,* 3, 181–201.

Alur, R. and Madhusudan, P. 2004. Decision problems for timed automata: A survey. In *SFM*. LNCS 3185. 1–24.

Asarin, E. and Maler, O. 1999. As soon as possible: Time optimal control for timed automata. In *HSCC*. LNCS 1569. 19–30.

Attie, P. C., Arora, A., and Emerson, E. A. 2004. Synthesis of fault-tolerant concurrent programs. *ACM Transactions on Programming Languages Systems (TOPLAS) 26,* 1, 125–185.

Bemporad, A. 2004. Hybrid Toolbox. `http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox/`.

Bemporad, A. and Giorgetti, N. 2004. A sat-based hybrid solver for optimal control of hybrid systems. In *HSCC*. LNCS 2993. 126–141.

Benerecetti, M., Faella, M., and Minopoli, S. 2011. Revisiting synthesis of switching controllers for linear hybrid systems. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. 4753 –4758.

Brogan, W. L. 1991. *Modern control theory (3rd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Bryant, R. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers C-35,* 8, 677–691.

Cassez, F., David, A., Fleury, E., Larsen, K. G., and Lime, D. 2005. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*. LNCS 3653. 66–80.

Cerný, P., Chatterjee, K., Henzinger, T. A., Radhakrishna, A., and Singh, R. 2011. Quantitative synthesis for concurrent programs. In *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Springer, 243–259.

Cimatti, A., Roveri, M., and Traverso, P. 1998. Strong planning in non-deterministic domains via model checking. In *AIPS*. 36–43.

Della Penna, G., Magazzeni, D., Mercorio, F., and Intrigila, B. 2009. UPMurphi: A tool for universal planning on pddl+ problems. In *ICAPS*.

Della Penna, G., Magazzeni, D., Tofani, A., Intrigila, B., Melatti, I., and Tronci, E. 2008. *Automated Generation of Optimal Controllers through Model Checking Techniques*. Lecture Notes in Electrical Engineering Series, vol. 15. Springer.

Dominguez-Garcia, A. and Krein, P. 2008. Integrating reliability into the design of fault-tolerant power electronics systems. In *PESC*. IEEE, 2665–2671.

Eker, J., Janneck, J., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Sachs, S., and Xiong, Y. 2003. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE 91,* 1, 127–144.

Frehse, G. 2008. Phaver: algorithmic verification of hybrid systems past hytech. *Int. J. Softw. Tools Technol. Transf. 10,* 3, 263–279.

Fu, M. and Xie, L. 2005. The sector bound approach to quantized feedback control. *IEEE Trans. on Automatic Control 50,* 11, 1698–1711.

Girard, A., Pola, G., and Tabuada, P. 2010. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control 55,* 1, 116–126.

Girault, A. and Rutten, É. 2009. Automating the addition of fault tolerance with discrete controller synthesis. *Formal Methods in System Design 35,* 2, 190–225.

Gulwani, S., Jha, S., Tiwari, A., and Venkatesan, R. 2011. Synthesis of loop-free programs. In *32nd Conference on Programming Language Design and Implementation, PLDI*, M. W. Hall and D. A. Padua, Eds. ACM, 62–73.

Gvero, T., Kuncak, V., and Piskac, R. 2011. Interactive synthesis of code snippets. In *Computer Aided Verification - 23rd International Conference, CAV*, G. Gopalakrishnan and S. Qadeer, Eds. Lecture Notes in Computer Science Series, vol. 6806. Springer, 418–423.

Henzinger, T., Ho, P.-H., and Wong-Toi, H. 1997. Hytech: A model checker for hybrid systems. *STTT 1,* 1, 110–122.

Henzinger, T. A. 2010. From boolean to quantitative notions of correctness. In *POPL*. ACM, 157–158.

Henzinger, T. A. and Kopke, P. W. 1997. Discrete-time control for rectangular hybrid automata. In *ICALP*. 582–593.

Henzinger, T. A., Kopke, P. W., Puri, A., and Varaiya, P. 1998. What's decidable about hybrid automata? *J. of Computer and System Sciences 57,* 1, 94–124.

Henzinger, T. A. and Sifakis, J. 2006. The embedded systems design challenge. In *FM*. LNCS 4085. 1–15.

Hermanns, H., Larsen, K. G., Raskin, J.-F., and Tretmans, J. 2010. Quantitative system validation in model driven design. In *EMSOFT*. ACM, 301–302.

Jha, S., Seshia, S. A., and Tiwari, A. 2011. Synthesis of optimal switching logic for hybrid systems. In *EMSOFT*. ACM, 107–116.

Jha, S. K., Gulwani, S., Seshia, S. A., and Tiwari, A. 2010. Synthesizing switching logic for safety and dwell-time requirements. Tech. Rep. UCB/EECS-2010-28, EECS Department, University of California, Berkeley. Mar.

Kim, W., Gupta, M. S., Wei, G.-Y., and Brooks, D. M. 2007. Enabling on-chip switching regulators for multi-core processors using current staggering. In *ASGI*.

Kreisselmeier, G. and Birkhölzer, T. 1994. Numerical nonlinear regulator design. *IEEE Trans. on on Automatic Control 39,* 1, 33–46.

Kuncak, V., Mayer, M., Piskac, R., and Suter, P. 2010. Comfusy: A tool for complete functional synthesis. In *Computer Aided Verification, 22nd International Conference, CAV*, T. Touili, B. Cook, and P. Jackson, Eds. Lecture Notes in Computer Science Series, vol. 6174. Springer, 430–433.

Kuncak, V., Mayer, M., Piskac, R., and Suter, P. 2012. Software synthesis procedures. *Commun. ACM 55,* 2, 103–111.

Larsen, K. G., Pettersson, P., and Yi, W. 1997. Uppaal: Status & developments. In *CAV*. LNCS 1254. 456–459.

Maler, O., Manna, Z., and Pnueli, A. 1992. From timed to hybrid systems. In *Proceedings of Real-Time: Theory in Practice, REX Workshop*, J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, Eds. Lecture Notes in Computer Science Series, vol. 600. Springer, 447–484.

Maler, O., Nickovic, D., and Pnueli, A. 2007. On synthesizing controllers from bounded-response properties. In *CAV*. LNCS 4590. Springer, 95–107.

Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2010. Synthesis of quantized feedback control software for discrete time linear hybrid systems. In *CAV*. LNCS 6174. 180–195.

Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2011a. From boolean relations to control software. In *ICSEA*.

Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2011b. Quantized feedback control software synthesis from system level formal specifications. *CoRR abs/1107.5638v1*.

Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2011c. Quantized feedback control software synthesis from system level formal specifications for buck dc/dc converters. *CoRR abs/1105.5640*.

Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2012a. Control software visualization. In *Proceedings of INFOCOMP 2012, The Second International Conference on Advanced Communications and Computation*. ThinkMind, 15–20.

Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2012b. Linear constraints as a modeling language for discrete time hybrid systems. In *Proceedings of ICSEA 2012, The Seventh International Conference on Software Engineering Advances*. ThinkMind, 664–671.

Mari, F., Melatti, I., Salvo, I., and Tronci, E. 2012c. Undecidability of quantized state feedback control for discrete time linear hybrid systems. In *Proceedings of the International Colloquium on Theoretical Aspects of Computing, ICTAC*, A. Roychoudhury and M. D'Souza, Eds. LNCS Series, vol. 7521. Springer-Verlag Berlin Heidelberg, 243–258.

Mazo, M., Davitian, A., and Tabuada, P. 2010. Pessoa: A tool for embedded controller synthesis. In *CAV*. LNCS 6174. 566–569.

Mazo, M. J. and Tabuada, P. 2011. Symbolic approximate time-optimal control. *Systems & Control Letters 60,* 4, 256–263.

Monniaux, D. 2010. Quantifier elimination by lazy model enumeration. In *CAV*. LNCS 6174. 585–599.

Neumaier, A. and Shcherbina, O. 2004. Safe bounds in linear and mixed-integer programming. *Mathematical Programming, Ser. A 99*, 283–296.

Peter, H.-J., Ehlers, R., and Mattmüller, R. 2011. Synthia: Verification and synthesis for timed automata. In *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Springer, 649–655.

Platzer, A. and Clarke, E. M. 2009. Formal verification of curved flight collision avoidance maneuvers: A case study. In *Proceedings of FM 2009: Formal Methods, Second World Congress*, A. Cavalcanti and D. Dams, Eds. Lecture Notes in Computer Science Series, vol. 5850. Springer, 547–562.

Pnueli, A. and Rosner, R. 1989a. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, POPL*. ACM Press, 179–190.

Pnueli, A. and Rosner, R. 1989b. On the synthesis of an asynchronous reactive module. In *Automata, Languages and Programming, 16th International Colloquium, ICALP*, G. Ausiello, M. Dezani-

Ciancaglini, and S. R. D. Rocca, Eds. Lecture Notes in Computer Science Series, vol. 372. Springer, 652–671.

Pola, G., Girard, A., and Tabuada, P. 2007. Symbolic models for nonlinear control systems using approximate bisimulation. In *Decision and Control, 2007 46th IEEE Conference on*. 4656 –4661.

QKS Web Page 2011. `http://mclab.di.uniroma1.it/`.

Ramadge, P. J. and Wonham, W. M. 1987. Supervisory control of a class of discrete event processes. *SIAM Journal Control Optimization 25,* 1, 206–230.

Sankaranarayanan, S. and Tiwari, A. 2011. Relational abstractions for continuous and hybrid systems. In *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Springer, 686–702.

SCADE Web Page 2012. `http://www.esterel-technologies.com/products/scade-system/`.

Schewe, S. and Finkbeiner, B. 2006. Synthesis of asynchronous systems. In *Logic-Based Program Synthesis and Transformation, 16th International Symposium, LOPSTR*, G. Puebla, Ed. Lecture Notes in Computer Science Series, vol. 4407. Springer, 127–142.

Schrom, G., Hazucha, P., Hahn, J., Gardner, D., Bloechel, B., Dermer, G., Narendra, S., Karnik, T., and De, V. 2004. A 480-mhz, multi-phase interleaved buck dc-dc converter with hysteretic control. In *PESC*. IEEE, 4702–4707 vol. 6.

Simulink Web Page 2012. `http://www.mathworks.it/products/simulink/`.

So, W.-C., Tse, C., and Lee, Y.-S. 1996. Development of a fuzzy logic controller for dc/dc converters: design, computer simulation, and experimental evaluation. *IEEE Trans. on Power Electronics 11,* 1, 24–32.

Srivastava, S., Gulwani, S., Chaudhuri, S., and Foster, J. S. 2011. Path-based inductive synthesis for program inversion. In *32nd Conference on Programming Language Design and Implementation, PLDI*, M. W. Hall and D. A. Padua, Eds. ACM, 492–503.

Srivastava, S., Gulwani, S., and Foster, J. S. 2010. From program verification to program synthesis. In *37th Symposium on Principles of Programming Languages, POPL*, M. V. Hermenegildo and J. Palsberg, Eds. ACM, 313–326.

Taly, A., Gulwani, S., and Tiwari, A. 2009. Synthesizing switching logic using constraint solving. In *Proc. 10th Intl. Conf. on Verification, Model Checking and Abstract Interpretation, VMCAI*. LNCS Series, vol. 5403. Springer, 305–319.

Texas Instruments 2001. Slvp182: High accuracy synchronous buck dc-dc converter: `http://focus.ti.com.cn/cn/lit/ug/slvu046/slvu046.pdf`.

Tiwari, A. 2008. Abstractions for hybrid systems. In *Formal Methods in Systems Design*.

Tronci, E. 1996. Optimal finite state supervisory control. In *CDC*. IEEE.

Tronci, E. 1997. On computing optimal controllers for finite state systems. In *CDC*. IEEE, 3592–3593 vol. 4.

Tronci, E. 1998. Automatic synthesis of controllers from formal specifications. In *ICFEM*. IEEE, 134–143.

Tronci, E. 1999a. Automatic synthesis of control software for an industrial automation control system. In *ASE*. IEEE, 247–250.

Tronci, E. 1999b. Formally modeling a metal processing plant and its closed loop specifications. In *HASE*. IEEE, 151.

Wong-Toi, H. 1997. The synthesis of controllers for linear hybrid automata. In *CDC*. IEEE, 4607–4612 vol. 5.

Yousefzadeh, V., Babazadeh, A., Ramachandran, B., Alarcon, E., Pao, L., and Maksimovic, D. 2008. Proximate time-optimal digital control for synchronous buck dc–dc converters. *IEEE Trans. on Power Electronics 23,* 4, 2018–2026.