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) and 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

ACM Reference Format:
Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci. 2014. Model-based synthesis of control software from system-level formal specifications. ACM Trans. Softw. Eng. Methodol. 23, 1, Article 6 (February 2014), 42 pages.
DOI: http://dx.doi.org/10.1145/2559934

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 Figure 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.

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

Authors' addresses: F. Mari (corresponding author), I. Melatti, I. Salvo, and E. Tronci, Dipartimento di Informatica, Sapienza Universita di Roma, Via Salaria 113, 00198 Roma, Italy; email: mari@di.uniroma.it.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2014 ACM 1049-331X/2014/02-ART6 $15.00
DOI: http://dx.doi.org/10.1145/2559934
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. Figure 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). Figure 2 summarizes the complete closed loop system forming an SBCS.

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, 1996; Henzinger et al. 1997] (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 (Figure 2), the control software may have a WCET greater than the sampling time (line 1 in Figure 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 aforesaid 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 2012], or MathWorks Simulink [Simulink 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 article.

For a more in-depth discussion of the literature related to the present article, we refer the reader to Section 9 and Table VI.

1.2. Our Main Contributions

We model the controlled system (plant) as a Discrete-Time Linear Hybrid System (DTLHS) (see Section 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 Section 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], and 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 Sections 4 and 5). Given a DTLHS model $H$ for the plant, a quantization schema (i.e., how many bits we use for AD conversion) and system-level formal specifications, our algorithm (see Section 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 Section 7) Control_Law, Controllable_Region such that: function Control_Law implements a Quantized Feedback Controller (QFC) for $H$ 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 guaranteed to be linear in the number of bits of the state quantization schema (see Section 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 [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] and 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 setup 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 \(D_x\) either of the reals or of the integers (discrete variables). We denote with \(D_X\) the set \(\prod_{x \in X} D_x\). To clarify that a variable \(x\) is continuous (i.e., real valued) we may write \(x^c\). 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 \(X^c\) \((X^d, X^b)\) denotes the sequence of real (integer, boolean) variables in \(X\). Unless otherwise stated, we suppose \(D_{X^c} = \mathbb{R}^{\lvert X\rvert}\) and \(D_{X^d} = \mathbb{Z}^{\lvert X\rvert}\). 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, \(\sum_{x \in X} a_x x\). 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 \((A(X) \land B(X))\) and \((A(X) \lor B(X))\) 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 \((L(X) \leq b) \land (L(X) \geq b)\) and \(a \leq x \leq b\) for \(x \geq a \land x \leq b\), 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 \(v(x) \in D_x\). Given a valuation \(v\), we denote with \(X^v \in D_X\) the sequence of values \([v(x_1), \ldots, v(x_n)]\). By abuse of language, we call valuation also the sequence of values \(X^v\). A satisfying assignment to a predicate \(P\) over \(X\) is a valuation \(X^v\) such that \(P(X^v)\) 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 \( a, b \in D_x \) 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 \( y \to C(X) \) (if \( y \) then \( C(X) \)) denotes the predicate \( ((y = 0) \vee C(X)) \). Similarly, we use \( \neg y \to C(X) \) (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 \( \text{feasible}(A(X)) \) that returns \( \text{true} \) if \( A(X) \) is feasible and \( \text{false} \) otherwise, and function \( \text{optimal-Value}(\max, J(X), A(X)) \) 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 \( S = (S, A, T) \) 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 \( S \). We say that \( T \) (and \( S \)) 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 \( \text{Adm}(S, s) \) the set of actions admissible in \( s \), that is \( \text{Adm}(S, s) = \{ a \in A \mid \exists s' : T(s, a, s') \} \) and with \( \text{Img}(S, s, a) \) the set of next states from \( s \) via \( a \), that is \( \text{Img}(S, s, a) = \{ s' \in S \mid T(s, a, s') \} \). We call transition a triple \((s, a, s') \in S \times A \times S \) and self-loop a transition \((s, a, s) \). A transition \((s, a, s') \) is a (self-loop) transition of \( S \) iff \( T(s, a, s') \). A run or path for an LTS \( S \) 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 \( \pi \) is the number of actions in \( \pi \). We denote with \( \pi^{(s)}(t) \) the \( (t + 1) \)-th state element of \( \pi \), and with \( \pi^{(a)}(t) \) the \( (t + 1) \)-th action element of \( \pi \). That is \( \pi^{(s)}(t) = s_t \) and \( \pi^{(a)}(t) = a_t \).

Given two LTSs \( S_1 = (S, A, T_1) \) and \( S_2 = (S, A, T_2) \), we say that \( S_1 \) refines \( S_2 \) (denoted by \( S_1 \subseteq S_2 \)) 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 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 \( X^r \) the sequence of next state variables obtained by decorating with \( ^r \) all variables in \( X \).

\( -U = U^r \cup U^d \) is a finite sequence of input variables.
which variables are boolean, and thus may be used as guards in guarded constraints. To this aim, we will also clarify (see Section 2.1), for the sake of readability we will use bounded guarded predicates to describe the transition relation of bounded DTLHSs. 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 Section 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′) ≡ [u → x′ = αx] ∧ [u → x′ = βx] ∧ −4 ≤ x ≤ 4 be a guarded predicate with α = 1/2 and β = 3/2. Then H = (x), (u), ∅, N) is a bounded DTLHS. Note that H is deterministic. Adding nondeterminism to H allows us to address the problem of (bounded) variations in the DTLHS parameters. For example, variations in the parameter a can be modelled with a tolerance ρ ∈ [0, 1] for α. This replaces N with: N(a) ≡ [u → x′ ≤ (1 + ρ)αx] ∧ [u → x′ ≥ (1 − ρ)αx] ∧ [u → x′ = βx]. We have that H(a) = (x), (u), ∅, N(a)), for ρ ∈ (0, 1), is a nondeterministic DTLHS. Note that, as expected, H(0) = H.

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

Definition 3.3 (DTLHS Dynamics). Let H = (X, U, Y, N) be a DTLHS. The dynamics of H is defined by the Labeled Transition System LTS(H) = (D X, D U, N) where: N : D X × D U × D X → B is a function such that N(x, u, x′) ≡ ∃y ∈ D Y : N(x, u, y, x′). A state x for H is a state x for LTS(H) and a run (or path) for H is a run for LTS(H) (Section 2.3).

Example 3.4. Let H be the DTLHS of Example 3.2. Then a sequence π is a run for H iff state π(S)i + 1 is obtained by multiplying π(S)i by 3/2 when π(A)i = 1, and by 1/2 when π(A)i = 0.

3.1. Buck DC-DC Converter as a DTLHS

The buck DC-DC converter (Figure 3) is a mixed-mode analog circuit converting the DC input voltage (Vi in Figure 3) to a desired DC output voltage (VO in Figure 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] and 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], and Yousefzadeh et al. [2008]). The typical software-based approach (e.g., see So et al. [1996]) is to control the switch \( u \) in Figure 3 (typically implemented with a MOSFET) with a microcontroller.

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 Figure 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

\[
\begin{align*}
q & \to v_D = 0 & (3) & \quad i_D = i_L - i_u & (6) & \quad \bar{u} & \to v_u = R_{off} i_u & (9) \\
q & \to i_D \geq 0 & (4) & \quad \bar{q} & \to v_D \leq 0 & (7) & \quad v_D & = v_u - V_i & (10)
\end{align*}
\]

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

4. QUANTIZED FEEDBACK CONTROL

In this section, we formally define the quantized feedback control problem for DTLHSs (Section 4.3). To this end, first we give the definition of feedback control problem for LTSs (Section 4.1), and then for DTLHSs (Section 4.2). Finally, we show that our definitions are well founded (Section 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 \( S = (S, A, T) \) 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 \( S \) is a function \( K : S \times A \to \mathbb{B} \) such that \( \forall s \in S, \forall a \in A, \text{if } K(s, a) \text{ then } a \in \text{Adm}(S, s) \). We denote with \( \text{Dom}(K) \) the set of states for which a control action is defined. Formally, \( \text{Dom}(K) = \{ s \in S \mid \exists a : K(s, a) \} \). \( S^\text{cl(K)} \) denotes the closed loop system, that is the LTS \( (S, A, T^{\text{cl(K)}}) \), where \( T^{\text{cl(K)}}(s, s') = T(s, a, s') \land K(s, a) \). A control law for a controller \( K \) is a (partial) function \( k : S \to A \) such that for all \( s \in \text{Dom}(K) \) 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 \( (K(x, a) \land K(x, b)) \to (a = b) \). An LTS control problem is a triple \( (S, I, G) \).

**Example 4.2.** Let \( S = \{-1, 0, 1\} \) and \( A = \{0, 1\} \). Let \( S_0 \) be the LTS \( (S, A, T_0) \), where the transition relation \( T_0 \) consists of the continuous arrows in Figure 4. A function \( K \) is a controller for \( S_0 \) iff \( (s \neq 0) \to (K(s, 1) = 0) \). As an example, we have that \( K \) defined as \( K(s, a) = ((s \neq 0) \to (a = 1)) \) 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).

Definition 4.1 also introduces the formal definition of control law, as our model of control software, that is, of how function \texttt{ControlLaw} in Figure 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 $\pi$ fullpath if either it is infinite or its last state $\pi^{(S)}(|\pi|)$ has no successors (i.e., $\text{Adm}(S, \pi^{(S)}(|\pi|)) = \emptyset$). We denote with $\text{Path}(S, s, a)$ the set of fullpaths of $S$ starting in state $s$ with action $a$, that is, the set of fullpaths $\pi$ such that $\pi^{(S)}(0) = s$ and $\pi^{(A)}(0) = a$.

Given a path $\pi$ in $S$, we define the measure $J(S, G, \pi)$ on paths as the distance of $\pi^{(S)}(0)$ to the goal on $\pi$. That is, if there exists $n > 0$ s.t. $\pi^{(S)}(n) \in G$, then $J(S, G, \pi) = \min\{n \mid n > 0 \land \pi^{(S)}(n) \in G\}$. Otherwise, $J(S, G, \pi) = +\infty$. 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 $J^{\text{strong}}(S, G, s) = \sup\{J^{(S)}(S, G, s, a) \mid a \in \text{Adm}(S, s)\}$, where: $J^{(S)}(S, G, s, a) = \sup\{J(S, G, \pi) \mid \pi \in \text{Path}(S, s, a)\}$. The best-case distance (optimistic view) of a state $s$ from the goal region $G$ is $J^{\text{weak}}(S, G, s) = \sup\{J^{(W)}(S, G, s, a) \mid a \in \text{Adm}(S, s)\}$, where: $J^{(W)}(S, G, s, a) = \inf\{J(S, G, \pi) \mid \pi \in \text{Path}(S, s, a)\}$.

Definition 4.3 (Solution to LTS Control Problem). Let $\mathcal{P} = (S, I, G)$ be an LTS control problem and $K$ be a controller for $S$ such that $I \subseteq \text{Dom}(K)$. $K$ is a strong [weak] solution to $\mathcal{P}$ if for all $s \in \text{Dom}(K)$, $J^{\text{strong}}(S^{(K)}, G, s) [J^{\text{weak}}(S^{(K)}, G, s)]$ 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 $J^{\text{strong}}(S^{(K^*)}, G, s) \leq J^{\text{strong}}(S^{(K)}, G, s) [J^{\text{weak}}(S^{(K^*)}, G, s) \leq J^{\text{weak}}(S^{(K)}, G, s)]$.

Intuitively, a strong solution $K$ takes a pessimistic view by requiring that for each initial state, all runs in the closed loop system $S^{(K)}$ 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 $\text{Path}(S^{(K)}, s, a)$ 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 Section 4.4 we show that the definition of mgo is well posed.

Example 4.4. Let $S_0, S_1$ be the LTSs in Figure 4 (see also Example 4.2). Let $\mathcal{P}_0 = (S_0, I, G)$ and $\mathcal{P}_1 = (S_1, I, G)$ 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 $K(s, a) \equiv a = 0$ is such that $J^{\text{strong}}(S_0^{(K)}), G, 0) = 1 < 2 = J^{\text{strong}}(S_0^{(K)}), G, 0)$. 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 \( J_{\text{strong}}(S_1^{(K)}, \hat{G}, 1) = J_{\text{strong}}(S_1^{(K)}, \hat{G}, -1) = \infty \) 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 \( \hat{K} \) 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 \( S^{(K)} \) is stable from \( I \) to \( G \), that is each run in \( S^{(K)} \) starting from a state in \( I \) leads to a state in \( G \). In fact, from Definition 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 \( S^{(K)} \) 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 \( (LT S(\mathcal{H}), I, G) \). Thus, a controller \( K : D X \times D U \rightarrow E \) is a strong [weak] solution to \((\mathcal{H}, I, G)\) iff it is a strong [weak] solution to \((LT S(\mathcal{H}), I, G)\).

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 Example 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 Section 3.1) and will not measure auxiliary variables (e.g., diode state in Section 3.1).

**Example 4.7.** The typical goal of a controller for the buck DC-DC converter in Section 3.1 is keeping the output voltage \( v_O \) close enough to a given reference value \( V_{\text{ref}} \). This leads to the DTLHS control problem \( \mathcal{P} = (\mathcal{H}, I, G) \) where \( \mathcal{H} \) is defined in Section 3.1, \( I \equiv (|i_L| \leq 2) \land (0 \leq v_O \leq 6.5), G \equiv (|v_O - V_{\text{ref}}| \leq \theta) \land (|i_L| \leq 2), \) 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 \( \gamma \) for a real interval \( I = [a, b] \) is a nondecreasing 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 \( \gamma \), denoted by \( ||\gamma|| \), is defined as \( \sup \{|w - z| : w, z \in I \land \gamma(w) = \gamma(z)\} \).

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 $Q$ for $\mathcal{H}$ is a pair $(A, \Gamma)$, where:

- $A$ is a predicate of form $\wedge_{w \in W} (a_w \leq w \leq b_w)$ with $a_w, b_w \in D_w$. For each $w \in W$, we define $A_w = \{v \in D_w \mid a_w \leq v \leq b_w\}$ 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 $V$.

- $\Gamma$ is a set of maps $\Gamma = \{\gamma_w \mid w \in W\}$ and $\gamma_w$ is a quantization function for $A_w$.

Let $V = \{v_1, \ldots, v_k\}$ and $v = [v_1, \ldots, v_k] \in A_V$, where $V \subseteq W$. We write $\Gamma(v)$ (or $\hat{v}$) for the tuple $[\gamma_{v_1}(v_1), \ldots, \gamma_{v_k}(v_k)]$ and $\Gamma^{-1}(v)$ for the set $\{v \in A_V \mid \Gamma(v) = \hat{v}\}$. Finally, the quantization step $\|\Gamma\|$ for $\Gamma$ 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 Example 3.2. Let us consider the quantization $Q = (A, \Gamma)$, 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 $\gamma_x(x) = \text{round}(x/2)$ (where round$(x) = |x| + [2(x - |x|)]$ 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, that is, 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 $\varepsilon$-solution. Essentially, we require that the controller drives the plant “near enough” (up to a given error $\varepsilon$) to the goal region $G$.

Definition 4.11 ($\varepsilon$-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 $\varepsilon$) $B_\varepsilon(W) = \{(z_1, \ldots, z_n, q_1, \ldots, q_m) \mid \exists(x_1, \ldots, x_n, q_1, \ldots, q_m) \in W \text{ and } \forall i \in \{1, \ldots, n\} \ |z_i - x_i| \leq \varepsilon\}$.

Definition 4.12 ($\varepsilon$-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] $\varepsilon$-solution to $\mathcal{P}$ is a strong [weak] solution to the LTS control problem ($\text{LTS}(\mathcal{H}), I, B_\varepsilon(G)$).

Example 4.13. Let $\mathcal{H}$ be the DTLHS described in Example 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 $\varepsilon$-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 $Q = (A, \Gamma)$. 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, $Q = (A, \Gamma)$ 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 $K : D_X \times D_U \to \mathbb{B}$ 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 $Q$. 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 Q QFC solutions by only
specifying their behavior inside the admissible region.

Example 4.15. Let $P$ be the DTLHS control problem defined in Example 4.13 and $Q = (A, \Gamma)$ be the quantization defined in Example 4.10. Let $K$ be defined by $\hat{K}(\hat{x}, \hat{u}) \equiv [\hat{x} \neq 0 \rightarrow \hat{u} = 0]$. For any $\varepsilon > 0$, the quantized controller $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is an $\varepsilon$-solution to $P$, and hence it is a $Q$ 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 $Q$ QFC solution to a DTLHS control problem (DTLHS quantized control problem) is undecidable, as shown in Mari et al. [2012].

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 Section 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 $P$ is an optimal strong [weak] solution $K$ to $P$ such that for all other optimal strong [weak] solutions $K'$ to $P$, for all $s \in S$, for all $a \in A$ we have that $K(s, a) \rightarrow K'(s, a)$.

Proposition 4.18. An LTS control problem $(S, \emptyset, G)$ has always an unique strong mgo $K^\ast$. Moreover, for all $I \subseteq S$, we have:

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

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

$-D_0 = \emptyset$

$-F_1 = \{s \in S | \exists a \in A : a \in \text{Adm}(S, s) \land \text{Img}(S, s, a) \subseteq G\}$

$-F_{n+1} = \{s \in S \setminus D_n | \exists a \in A : a \in \text{Adm}(S, s) \land \text{Img}(S, s, a) \subseteq D_n\}$

$-D_{n+1} = D_n \cup F_{n+1}$

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 $F_n = \emptyset$ for some $n \geq 1$, then for all $m \geq n$, $F_m = \emptyset$. In fact, if $F_n = \emptyset$, then $D_n = D_{n-1}$, and hence $F_{n+1} = F_n = \emptyset$.

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 $\emptyset$). 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. $F_j \cap F_i = \emptyset$ for all $i \neq j$. We have that if $s \in F_{n+1}$ then $s \not\in D_n$. By previous point 3, we have that $s \not\in D_n$ implies $s \not\in F_j$ for $1 \leq j \leq n$. Hence, $s \in F_{n+1}$ implies that $s \not\in F_j$ for all $1 \leq j \leq n$. If by absurd a state $s$ exists such that $s \in F_i \cap F_j$ for some $i > j$, then $s \in F_{j}$ would imply $s \not\in F_j$. 

ACM Transactions on Software Engineering and Methodology, Vol. 23, No. 1, Article 6, Pub. date: February 2014.
Let us suppose that there exists another optimal solution \( K \) such that for all \( z \in Z \) there exists an action \( a \) such that \( K(z, a) \) and \( -K(z, a) \) holds. Let \( z_0 \in Z \) be a state for which \( J_{\text{strong}}(K^*, G, z_0) = n \) is minimal in \( Z \).

If \( n > 1 \), for all \( s \in \text{Im}(G, z_0, a) \), we have that \( J_{\text{strong}}(K^*, G, s) \leq n - 1 \). Since \( n \) is the minimal distance for which \( J_{\text{strong}}(K^*, G, z) > J_{\text{strong}}(K^*, G, z) = n \), we have that for all \( s \in \text{Im}(G, z_0, a) \), \( J_{\text{strong}}(K^*, G, s) \leq J_{\text{strong}}(K^*, G, s) \leq n - 1 \). This implies that, if \( n = 1 \) we have that \( \text{Im}(G, z_0, a) \subseteq G \) and thus \( z_0 \in F_1 \) and \( \bar{K}(z_0, a) \), which leads to a contradiction.
If \( n > 1 \), by minimality of \( J_{\text{strong}}(S^k, G, z_0) \) in \( Z \) we have that, for all \( s \in \text{Img}(S, z_0, \alpha) \), \( K(s, u) \) implies \( K(s, u) \). This implies that \( \text{Img}(S, z_0, \alpha) \in D_{n-1} \) and thus \( K(z_0, \alpha) \) holds. \( \square \)

5. CONTROL ABSTRACTION

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

Control abstraction (Definition 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 (Definition 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 \( Q = (A, \Gamma) \) 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 \( Q \)-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 Example 3.2 and \( Q \) as in Example 4.10. We have that action \( u = 1 \) is not \( A \)-admissible in the state \( s = 2 \), thus \( \hat{u} = 1 \) is not \( Q \)-admissible in the state \( \hat{s} = 1 \). Analogously, \( \hat{u} = 1 \) is not \( Q \)-admissible in \( \hat{s} = -1 \). It is easy to see that no other \( \hat{u}, \hat{s} \) exist such that \( \hat{u} \) is not \( Q \)-admissible in \( \hat{s} \).

**Definition 5.3 (Control Abstraction).** Let \( \mathcal{H} = (X, U, Y, N) \) be a DTLHS and \( Q = (A, \Gamma) \) be a quantization for \( \mathcal{H} \). We say that the LTS \( \hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N}) \) is a 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 \( Q \)-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 \( Q \)-admissible in \( \Gamma(s) \) and \( \Gamma(s) \neq \Gamma(s') \) then \( N(\Gamma(s), \Gamma(u), \Gamma(s')) \).
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 there exists an infinite run \( \pi \) 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 preceding property is said to be a noneliminable self-loop, and eliminable self-loop otherwise.

**Example 5.4.** Let \( \mathcal{H} \) be as in Example 3.2 and \( Q \) be as in Example 4.10. Any 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 Figure 4 and some dotted arrows. Note that the only noneliminable self-loops are \( (0, 0, 0) \) and \( (0, 1, 0) \).

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

**Proposition 5.5.** Given a DTLHS \( \mathcal{H} \) and a quantization \( Q \), it is undecidable to determine if a self-loop is noneliminable.
Note that if in Definition 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 nonadmissible 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 \( Q = (A, \Gamma) \) be a quantization for \( \mathcal{H} \). A \( Q \) control abstraction \( \hat{\mathcal{H}} = (\Gamma(A_X), \hat{N}) \) of \( \mathcal{H} \) is an admissible control abstraction if, for all \( \hat{s} \in \Gamma(A_X), \hat{u} \in \Gamma(A_U) \) such that \( \hat{u} \in \text{Adm}(\hat{\mathcal{H}}, \hat{s}) \); (i) \( \hat{u} \) is \( Q \)-admissible in \( \hat{s} \); (ii) \( \forall s \in \Gamma^{-1}(\hat{s}) \forall u \in \Gamma^{-1}(\hat{u}) \exists y' \in D_X \exists y \in D_Y : N(s, u, y, y') \), that is, 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 control abstraction if it satisfies properties 1 and 3 of Definition 5.3, plus the following property (derived from property 2 of Definition 5.3): for all \( s, s' \in A_X, u \in A_U \) such that \( \exists y : N(s, u, y, y') \), if \( \Gamma(s) \neq \Gamma(s') \) then \( \hat{N}(\Gamma(s), \Gamma(u)) \).

**Example 5.7.** Let \( \mathcal{H} \) be as in Example 3.2, \( Q \) be as in Example 4.10. For all \( Q \) admissible control abstractions of \( \mathcal{H}, \hat{N}(1, 1, 1) = \hat{N}(-1, 1, -1) = 0 \), since action 1 is not \( Q \)-admissible either in \(-1 \) or in \( 1 \) (see Example 5.2). On the contrary, for all full \( Q \) control abstractions of \( \mathcal{H}, \hat{N}(1, 1, 1) = \hat{N}(-1, 1, -1) = 1 \). Thus, a control abstraction such that, \( \hat{N}(1, 1, 1) \oplus \hat{N}(-1, 1, -1) \) (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] \( Q \) control abstractions only differ in the number of self-loops. Moreover, the set of admissible [full] \( Q \) control abstractions is a finite lattice with respect to the LTS refinement relation (Section 5.2). This implies that such lattices have minimum (and maximum). Thus, it is easy to prove that the minimum admissible [full] \( Q \) control abstraction is the admissible [full] \( Q \) control abstraction with noneliminable self-loops only. Thus, the following proposition is a corollary of Proposition 5.5.

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

### 5.1. Maximum and Minimum Control Abstractions

By Theorem 4.16, we cannot hope for a constructive sufficient and necessary condition for the existence of a \( Q \) QFC solution to a DTLHS control problem, for a given \( Q \). Accordingly, our approach is able to determine (via a sufficient condition) if a \( Q \) QFC solution exists, and otherwise to state (via a necessary condition) if a \( Q \) QFC solution cannot exist. If both conditions are false, then our approach is not able to decide if a \( Q \) QFC solution exists or not. We base our sufficient [necessary] condition on computing a (close to) minimum admissible [full] \( Q \) control abstraction. Theorem 5.9 gives the foundations for such an approach. The proof of Theorem 5.9 follows from the definitions of admissible and full control abstractions and properties of strong and weak solutions (Section 5.2). In the following theorem we use the refinement order relation (denoted by \( \sqsubseteq \)) defined in Section 2.3.
**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 $\text{C}(\mathcal{H}, Q)$ the set of all $Q$ 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 $Q$ control abstractions of a DTLHS $\mathcal{H}$, with $Q = (A, \Gamma)$ quantization for $\mathcal{H}$. Then $\forall \hat{x}, \hat{x}' \in S$ such...
that $\hat{x} \neq \hat{x}'$, $\forall \hat{a} \in B[T_1(\hat{x}, \hat{a}, \hat{x}') \Rightarrow T_2(\hat{x}, \hat{a}, \hat{x}')]$. The same holds if $M_1, M_2$ are full $Q$ 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 $M_1$ is an admissible $Q$ control abstraction, this implies, by Definition 5.6, that $\hat{a}$ is $A$-admissible in $\hat{x}$. From point 1 of Definition 5.3 (for the admissible control abstraction case) or Definition 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 Definition 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 $H$ and a quantization $Q$, the set $(C(H, Q), \sqsubseteq)$ of $Q$ control abstractions of $H$ is a lattice. Moreover, the set of full $Q$ control abstractions of $H$ is a lattice.

**Proof.** By conditions 2 and 3 of Definition 5.3 all control abstractions do contain all admissible actions that have a concrete witness and all noneliminable self-loops.

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

Analogously, the set of full $Q$ control abstractions of $H$ 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, that is, the same arcs except from self-loops, as proved by Proposition 5.11. For ease of notation, given a state $x$ (respectively, an action $u$) we will often denote the corresponding abstract state $\Gamma(x)$ (respectively, action $\Gamma(u)$) with $\hat{x}$ (respectively $\hat{u}$). Analogously, we will often write $\hat{I}$ (respectively, $\hat{G}$) for $\Gamma(I)$ (respectively, $\Gamma(G)$).

In the following, $\mathcal{P} = (H, I, G, \mathcal{P})$ and $\hat{\mathcal{P}} = (\hat{H}, \Gamma(I), \Gamma(G))$, and $\hat{\mathcal{P}} = (\Gamma(Ax), \Gamma(Au), \hat{N})$.

**Proof of Point 1.** Applying the definition of solution to a DTLHS control problem (Definition 4.12), we have to show that if $\hat{K}$ is a strong solution to the LTS control problem $(\hat{H}, \hat{I}, \hat{G})$, then $K$ defined by $K(x, u) = (k(x) = \hat{u})$ is a strong solution to the LTS control problem $\text{LTS}(H), I, B_{\parallel\Gamma\parallel}(G)$, being $k$ a control law for $\hat{K}$.

Note that, since $\hat{H}$ is an admissible control abstraction, it contains admissible actions only. This implies that all actions enabled by $\hat{K}$ in $\hat{x}$ are $Q$-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 Definition 5.3, this implies that, for any transition $(x, u, x')$ of $\text{LTS}(H,K)$ such that $\hat{x} \neq \hat{x}'$, $(\hat{x}, \hat{u}, \hat{x}')$ is a (abstract) transition of $\hat{\mathcal{P}}$.

First of all, we prove that $I \subseteq \text{Dom}(K)$. 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 $\hat{I} \subseteq \text{Dom}(\hat{K})$, thus $\hat{x} \in \text{Dom}(\hat{K})$. Hence, there exists $\hat{u} \in \Gamma(A_K)$ such that $\hat{K}(\hat{x}, \hat{u})$ holds, which implies that $k(x, u)$ is defined. By definition of $K$, we have that for all $u \in \Gamma^{-1}(k(x))$ and for all $x \in \Gamma^{-1}(k(x)) K(x, u)$ holds, which means that $x \in \text{Dom}(K)$.

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

1. There exists a finite fullpath $\pi = x_0 u_0 x_1 u_1 \ldots x_n u_n$ in $\text{LTS}(H,K)$ such that $x_0 = x$, $\text{Adm}(\text{LTS}(H,K), x_n) = \emptyset$ and, for all $i \in [n], x_i \notin B_{\parallel\Gamma\parallel}(G)$. 

ACM Transactions on Software Engineering and Methodology, Vol. 23, No. 1, Article 6, Pub. date: February 2014.
There exists an infinite fullpath $\pi = x_0 u_0 x_1 u_1 \ldots x_n u_n \ldots$ in $LTS(\mathcal{H})^K$ such that $x_0 = x$ and, for all $i \in \mathbb{N}$, $x_i \not\in B|_{\Gamma}(\mathcal{G})$.

Let us deal with the finite fullpath case first (point 1 just given). Let $\hat{\pi} = \hat{x}_0 \hat{u}_0 \ldots \hat{x}_{n-1} \hat{u}_n$, and let $\rho$ 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 $\rho(i) = \hat{\pi}^{(\alpha(i))} = \Gamma(\pi^{(\alpha(i))})$, where the function $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ is recursively defined as follows.

- $\alpha(0) = 0$
- $\alpha(i + 1) = \begin{cases} \alpha(i) & \text{if } \alpha(i) \geq \pi_i \wedge \pi_i < 0 \\ \min \alpha(i) & \text{otherwise} \end{cases}$

By the fact (proved before) that if $\pi = G(\mathcal{H})^K$ with $\pi \neq \pi'$, then $\pi$ is a transition of $\mathcal{H}^K$, we have that $\rho$ is a run of $\mathcal{H}^K$. Let $m = |\rho| = \max_{i \in [n]} \alpha(i)$. Since $\hat{K}$ is a strong solution to $\hat{\pi}$, we have that $\hat{x}_m \in \text{Dom}(\hat{K})$. This implies that there exists $\hat{u} \in \Gamma(A_{\mathcal{K}})$ such that $K(\hat{x}_m, \hat{u})$ and $K(\hat{x}_m, \hat{u})$, thus there exists $\hat{u} \in \text{Adm}(\hat{\pi}(\hat{K}), \hat{x}_m)$. Thus by Definition 5.6 (and since $x_n = \Gamma^{-1}(\hat{x}_m)$) we have that $\text{Adm}(\pi, x_n) \geq \Gamma^{-1}(\hat{\pi}) \neq \emptyset$, which implies that $\pi$ cannot be a finite fullpath.

As for the infinite fullpath case (point 2 given earlier), we observe that in $\pi$ we cannot have an infinite sequence $x_m u_m x_{m+1} u_{m+1} \ldots$ such that for all $j \geq m$, $\pi_j = \pi(x_m)$ and $\pi_j = \pi(u_m)$. In fact, suppose by absurd that this is true, and let $m$ be the least $m$ for which this happens. Then $(\hat{x}_m, \hat{u}_m, \hat{x})$ is a noneliminable self-loop. Since $x_j \not\in B|_{\Gamma}(\mathcal{G})$ for all $j \geq m$, and thus $x_j \not\in \mathcal{G}$ for all $j \geq m$, we also have that $J_{\text{strong}}(\hat{\pi}(\hat{K}), \hat{G}, \hat{x}_m) = \infty$. By applying the same reasoning used for the finite fullpath case, we have that there is a path in $\hat{\pi}(\hat{K})$ leading from $\hat{x}$ to $\hat{x}_m$, which implies that $J_{\text{strong}}(\hat{\pi}(\hat{K}), \hat{G}, \hat{x}) = \infty$. Finally, this contradicts the fact that $\hat{K}$ is a strong solution to $\hat{\pi}$ and $\hat{x} \in \text{Dom}(\hat{K})$. 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 $x_m u_m x_{m+1} u_{m+1} \ldots$ such that for all $j \geq m$, $\pi_j = \pi(x_m)$.

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

- $\alpha(0) = 0$
- $\alpha(i + 1) = \min \{ j \mid \alpha(i) < j \wedge \pi_j \neq \pi(x_{(\alpha(i))}) \}$

By the fact (proved before) that if $\pi = G(\mathcal{H})^K$ with $\pi \neq \pi'$, then $\pi$ is a transition of $\mathcal{H}^K$, we have that $\rho$ is a run of $\mathcal{H}^K$. Moreover, since for all $i \in \mathbb{N}$, $x_i \not\in B|_{\Gamma}(\mathcal{G})$, then we have that for all $i \in \mathbb{N}$, $x_i \neq \hat{G}$. This contradicts the fact that $\hat{K}$ is a strong solution to $\hat{\pi}$ and $\hat{x} \in \text{Dom}(\hat{K})$.
Note that if \( \hat{x} \notin \hat{G} \) and \( \hat{K}(\hat{x}, \hat{u}) \) then \( J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) = \infty \) since there exists a \( \pi \in \text{Path}(\hat{H}_2^K, \hat{x}, \hat{u}) \) such that \( \pi(t) = \hat{x} \) and \( \pi(A)(t) = \hat{u} \) for all \( t \in \mathbb{N} \). As a consequence, if \( \hat{x} \notin \hat{G} \) then \( \hat{K}(\hat{x}, \hat{u}) \) does not hold. Moreover, suppose that \( \hat{x} \in \hat{G} \). Since \((\hat{x}, \hat{u}, \hat{x})\) is an eliminable self-loop of \( \hat{H}_2 \) and \( \hat{H}_2 \) is an admissible \( Q \) control abstraction, there exists a state \( \hat{x}' \neq \hat{x} \) such that \( T_2(\hat{x}, \hat{u}, \hat{x}') \).

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

As for the first point, we have to show that \( \hat{K}(\hat{x}, \hat{u}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}_1, \hat{x}) \) (Definition 4.1). Suppose by absurd that \( \hat{u} \notin \text{Adm}(\hat{H}_1, \hat{x}) \) for some \( \hat{x}, \hat{u} \). Since \( \hat{K}(\hat{x}, \hat{u}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}_2, \hat{x}) \), 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 \( \hat{u} \in \text{Adm}(\hat{H}_1, \hat{x}) \), which is absurd.

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

Suppose \( J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) = 1 \). Then, \( \emptyset \neq \text{Img}(\hat{H}_2^K, \hat{x}, \hat{u}) \subseteq \hat{G} \) for all \( \hat{u} \) such that \( \hat{K}(\hat{x}, \hat{u}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}_1, \hat{x}) \) for some \( \hat{x}, \hat{u} \). If for all \( \hat{u} \) such that \( \hat{K}(\hat{x}, \hat{u}) \) there exists a state \( \hat{x}' \neq \hat{x} \) such that \( \hat{x}' \in \text{Img}(\hat{H}_2^K, \hat{x}, \hat{u}) \), then we have that \( \hat{x}' \in \text{Img}(\hat{H}_1^K, \hat{x}, \hat{u}) \) by Fact 5.11, and since \( \emptyset \neq \text{Img}(\hat{H}_1^K, \hat{x}, \hat{u}) \subseteq \text{Img}(\hat{H}_2^K, \hat{x}, \hat{u}) \subseteq \hat{G} \) we have that \( J_{\text{strong}}(\hat{H}_1^K, \hat{G}, \hat{x}) = 1 = J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) \).

Otherwise, let \( \hat{u} \) be such that \( \hat{K}(\hat{x}, \hat{u}) \) and \( T_2(\hat{x}, \hat{u}, \hat{x}') \rightarrow \hat{x}' = \hat{x} \). 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 \( J_{\text{strong}}(\hat{H}_1^K, \hat{G}, \hat{x}) = 1 = J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) \). The other case, namely, \( (\hat{x}, \hat{u}, \hat{x}) \in B \), is impossible since, by the reasoning given before 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 \( \hat{x} \) such that \( J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) = n \), \( J_{\text{strong}}(\hat{H}_1^K, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) \). Let \( \hat{x} \in \text{Dom}(\hat{K}) \) be such that \( J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) = n + 1 \). If \( (\hat{x}, \hat{u}, \hat{x}) \notin B \) for any \( \hat{u} \), then \( \text{Img}(\hat{H}_2^K, \hat{x}, \hat{u}) = \text{Img}(\hat{H}_1^K, \hat{x}, \hat{u}) \) for all \( \hat{u} \), thus \( J_{\text{strong}}(\hat{H}_1^K, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) \) by induction hypothesis. Otherwise, let \( (\hat{x}, \hat{u}, \hat{x}) \in B \) for some \( \hat{u} \). By the reasoning given earlier, if \( \hat{x} \notin \hat{G} \) then \( \hat{K}(\hat{x}, \hat{u}) = 0 \), and again \( J_{\text{strong}}(\hat{H}_1^K, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) \) 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 \( J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) = n + 1 \), we must have \( J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}') \leq n \), thus again \( J_{\text{strong}}(\hat{H}_1^K, \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{H}_2^K, \hat{G}, \hat{x}) \) by inductive hypothesis.

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

**Proof of Point 3.** Applying the definition of DTLHS control problem (Definition 4.12), we will show that if \( \hat{K} \) is a weak solution to the LTS control problem (LTS(\( \hat{H} \), \( \hat{I} \), \( E_\|\Gamma\| \( (\hat{G}) \)),
and \( \hat{H} \) is any full \( Q \) control abstraction of \( H \) then there exists a weak solution \( \hat{K} \) to the control problem \((\hat{H}, \hat{I}, \hat{G})\).

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 \( Q \) control abstraction of \( H \).

Let \( \hat{H} \) be a full \( Q \) control abstraction of \( H \). First of all, we show that \( \hat{K} \) is a controller for \( \hat{H} \) (Definition 4.1), that is, that \( \hat{K}(\hat{x}, \hat{u}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}, \hat{x}) \). Suppose \( \hat{K}(\hat{x}, \hat{u}) \) holds: this implies that there exist \( x \in \Gamma^{-1}(\hat{x}), u \in \Gamma^{-1}(\hat{u}) \) such that \( K(x, u) \) and \( u \in \text{Adm}(H, x) \).

If there exists \( x' \in A_X \) such that \( x' \in \text{Img}(H, x, u) \) and \( \hat{x}' \neq \hat{x} \), then, being \( \hat{H} \) a full \( Q \) control abstraction of \( H \), we have that \((\hat{x}, \hat{u}, \hat{x}') \) is a transition of \( \hat{H} \), thus \( \hat{u} \in \text{Adm}(\hat{H}, \hat{x}) \). Otherwise, one of the following must hold:

- \( \text{Img}(H, x, u) = \emptyset \), which is impossible since \( K(x, u) \);
- for all \( x' \in A_X \) such that \( x' \in \text{Img}(H, x, u) \), we have that either \( x' \notin A_X \) or \( \hat{x}' = \hat{x} \).

Being \( K \) a weak controller for \( H \) 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 \( x' \in A_X \) such that \( x' \in \text{Img}(H, x, u) \) and \( \hat{x}' = \hat{x} \). If \( x = x' \), then there exists an infinite path inside \( \Gamma^{-1}(\hat{x}) \) with actions in \( \Gamma^{-1}(\hat{u}) \), that is, \((\hat{x}, \hat{u}, \hat{x})\) is a noneliminable self-loop. This implies that \( \hat{N}(\hat{x}, \hat{u}, \hat{x}) \) holds, thus \( \hat{u} \in \text{Adm}(\hat{H}, \hat{x}) \). Otherwise, that is, if \( x \neq x' \), then the 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 \( \hat{u} \in \text{Adm}(\hat{H}, \hat{x}) \), 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 noneliminable self-loop and \( \hat{N}(\hat{x}, \hat{u}, \hat{x}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}, \hat{x}) \).

We now have to prove that \( \hat{K} \) is a weak solution to \( \hat{H} \), where \( \hat{H} \) is a full \( Q \) control abstraction of \( H \). 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 \( \hat{K} \) is a weak solution to \( \hat{P} \), there exists \( u \in A_U \) such that \( K(x, u) \), thus by definition of \( \hat{K}, \hat{K}(\hat{x}, u) \) holds, and hence \( \hat{x} \in \text{Dom}(\hat{K}) \).

Now, we show that for all \( \hat{x} \in \text{Dom}(\hat{K}), J_{\text{weak}}(\hat{H}(\hat{K}), \hat{G}, \hat{x}) \) is finite. By definition of \( \hat{K} \), and since \( K \) is a weak solution to \( P \), there exists a finite path \( \pi = x_0u_0x_1u_1 \ldots u_{n-1}x_n \) such that \( x_0 \in \Gamma^{-1}(\hat{x}), x_i \in A_X \) for all \( 0 < i < n \) and \( x_n \in B_{\Gamma(X)}(G) \).

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

- let \( Z_\varepsilon = \{ j \mid j < n \wedge (\Gamma(x_j) \neq \Gamma(x_j)) \} \);
- \( \alpha(0) = 0 \);
- \( \alpha(i + 1) = \begin{cases} \alpha(i) & \text{if } Z_{\alpha(i)} = \emptyset \\ \min Z_{\alpha(i)} & \text{otherwise.} \end{cases} \)

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

**Proof of Point 4.** Analogously to the proof of point 2, let \( \hat{H}_1 = (\Gamma(A_X), \Gamma(A_U), T_1) \) and \( \hat{H}_2 = (\Gamma(A_X), \Gamma(A_U), T_2) \) be two full \( Q \) control abstractions of \( H \), with \( \hat{H}_1 \subseteq \hat{H}_2 \). If \( \hat{H}_1 = \hat{H}_2 \) the thesis is proved, thus let us suppose that \( \hat{H}_1 \neq \hat{H}_2 \). By Fact 5.11, the only difference between \( \hat{H}_1 \) and \( \hat{H}_2 \) may be in a finite number of eliminable self-loops which are in \( \hat{H}_2 \) only. Let \( B = (\hat{x}_1, \hat{u}_1, \hat{x}_1), \ldots, (\hat{x}_m, \hat{u}_m, \hat{x}_m) \) be the set of such self-loops. Let \( \hat{K} \) be the weak mgo to the LTS control problem \((\hat{H}_1, \hat{I}, \hat{G}) \) and let \((\hat{x}_i, \hat{u}_i, \hat{x}_i) \in B \).
Since we already know that \( \hat{I} \subseteq \text{Dom}(\hat{K}) \), we only have to prove that: (i) \( \hat{K} \) is a controller for \( \hat{\mathcal{H}} \) and that (ii) \( J_{\text{weak}}(\hat{\mathcal{H}}_2, \hat{G}, \hat{x}) < \infty \) for all \( \hat{x} \in \text{Dom}(\hat{K}) \).

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

As for the second one, it is sufficient to prove that \( J_{\text{weak}}(\hat{\mathcal{H}}_2, \hat{G}, \hat{x}) \leq J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) \).

This can be proved by induction on the value of \( J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) \).

Suppose \( J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) = 1 \). Then, \( \text{Img}(\hat{\mathcal{H}}_1, \hat{x}, \hat{u}) \cap \hat{G} \neq \emptyset \) for all \( \hat{u} \) such that \( \hat{K}(\hat{x}, \hat{u}) \). Then, \( \hat{K}_1 \) only adds self-loops to \( \hat{\mathcal{H}}_1 \), we have that \( \text{Img}(\hat{\mathcal{H}}_2, \hat{x}, \hat{u}) \cap \hat{G} \neq \emptyset \) for all \( \hat{u} \) such that \( \hat{K}(\hat{x}, \hat{u}) \), thus \( J_{\text{weak}}(\hat{\mathcal{H}}_2, \hat{G}, \hat{x}) = 1 \).

Suppose now that for all \( \hat{x} \) such that \( J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) = n \), \( J_{\text{weak}}(\hat{\mathcal{H}}_2, \hat{G}, \hat{x}) \leq J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) \).

Let \( \hat{x} \) be such that \( J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) = n + 1 \). If \( (\hat{x}, \hat{u}, \hat{x}) \in \mathcal{B} \) for any \( \hat{u} \), then \( \text{Img}(\hat{\mathcal{H}}_1, \hat{x}, \hat{u}) = \text{Img}(\hat{\mathcal{H}}_2, \hat{x}, \hat{u}) \) for all \( \hat{u} \), thus \( J_{\text{weak}}(\hat{\mathcal{H}}_2, \hat{G}, \hat{x}) \leq J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) \) by induction hypothesis. Otherwise, let \( (\hat{x}, \hat{u}, \hat{x}) \in \mathcal{B} \) for some \( \hat{u} \). If \( \hat{x} \notin \hat{G} \) we simply have that \( J_{\text{weak}}(\hat{\mathcal{H}}_2, \hat{G}, \hat{x}) \leq J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) \) by induction hypothesis. Otherwise, if \( \hat{x} \in \hat{G} \), let \( \hat{K}_1 \) be such that \( \hat{K}_1(\hat{x}, \hat{u}) = 0 \) and \( \hat{K}_1(\hat{x}, \hat{u}) = \hat{K}(\hat{x}, \hat{u}) \) for all \( \hat{u} \). Then, \( J_{\text{weak}}(\hat{\mathcal{H}}_2, \hat{G}, \hat{x}) \leq \max\{1, J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) \} \leq J_{\text{weak}}(\hat{\mathcal{H}}_1, \hat{G}, \hat{x}) \), thus the thesis is proved.

6. QUANTIZED CONTROLLER SYNTHESIS

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

**ALGORITHM 1: QFC synthesis**

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

**function** \( q\text{CtrSyn}(\mathcal{H}, Q, I, G) \)

1. \( I \leftarrow \Gamma(I), G \leftarrow \Gamma(G) \)
2. \( \hat{M} \leftarrow \text{minCtAbs}(\mathcal{H}, Q) \)
3. \( (b, \hat{D}, \hat{K}) \leftarrow \text{strongCt}(\hat{M}, I, \hat{G}) \)
4. **if** \( b \) **then return** \( (\text{Sol.}, \hat{D}, \hat{K}) \)
5. **end if**
6. **if exists WeakCt\( (\hat{W}, I, G) \) then return** \( (\text{Unk}, \hat{D}, \hat{K}) \)
7. **else return** \( (\text{NoSol.}, \hat{D}, \hat{K}) \)

Sections 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 \( (\hat{M}, I, \Gamma(I), \Gamma(G)) \). If \( \hat{K} \) exists, then a \( Q \) 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 \( Q \) control abstraction \( \mathcal{H} \). However, by point 2 of Theorem 5.9, \( \mathcal{H} \) must be lower than \( \hat{M} \) in the hierarchy lattice (see Figure 5). This suggests to compute \( \hat{M} \) as the minimum (admissible) \( Q \) control abstraction of \( \mathcal{H} \). Since by Proposition 5.8 we are
not able to compute the minimum $Q$ control abstraction, we compute $\hat{M}$ as a close to minimum admissible $Q$ control abstraction, that is, an admissible $Q$ control abstraction containing as few eliminable self-loops as possible (see Example 4.4).

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

6.1. QFC Synthesis Algorithm

Our QFC synthesis algorithm (function qCtrSyn outlined in Algorithm 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 $Q$ QFC (strong) solution to the control problem $(\mathcal{H}, \Gamma^{-1}(\hat{D}), G)$.

We represent boolean functions (e.g., the transition relation of $H$) 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.

Algorithm 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 Section 6.3).

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

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

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

If function existsWeakCtr returns $\text{FALSE}$, then a weak mgo to $\hat{P}'$ does not exist, and since the weak mgo is unique no weak solution exists to $\hat{P}'$. By Theorem 5.9 (point 3), no $Q$ 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 $\hat{P}$ for the (close to) minimum control abstraction is returned, together with its controlled region $\hat{D}$.

6.2. Synthesis Algorithm Correctness

The preceding 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, $Q = (A, \Gamma)$ be a quantization, and $(\mathcal{H}, I, G)$ be a DTLHS control problem. Then $qCtrSyn(\mathcal{H}, Q, I, G)$ returns a triple $(\mu, \hat{D}, \hat{K})$ such that: $\mu \in \{\text{Sol}, \text{NoSol}, \text{UNK}\}$, $\hat{D} = \text{Dom}(\hat{K})$ and, for all control laws $k$ for $\hat{K}$, $K(x, u) = (k(\Gamma(x)) = \Gamma(u))$ is a $Q$ QFC solution to the control problem $(\mathcal{H}, \Gamma^{-1}(\hat{D}), G)$.

Furthermore, the following holds: (i) if $\mu = \text{Sol}$ then $I \subseteq \Gamma^{-1}(\hat{D})$ and $K$ is a $Q$ QFC solution to the control problem $(\mathcal{H}, I, G)$; (ii) if $\mu = \text{NoSol}$ then there is no $Q$ QFC solution to the control problem $(\mathcal{H}, I, G)$.

Remark 6.2. Mazo and Tabuada [2011] describe a method for the automatic control software synthesis for continuous-time linear systems. Function $\text{strongCtr}$, as well as the approach in Mazo and Tabuada [2011], returns $\hat{K}$ as a (worst case) time optimal controller, that is, 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 ($\text{strongCtr}$ 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 $\text{strongCtr}$) 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 $\hat{x}_n, \hat{u}_n, \ldots, \hat{x}_m$ in every path of $\text{LTS}(\mathcal{H})$ into a single abstract transition $(\hat{x}_n, \hat{u}_n, \hat{x}_m)$ of $\hat{M}$ whenever $\hat{x}_n = \cdots = \hat{x}_{m-1}$ and $\hat{u}_n = \cdots = \hat{u}_{m-1}$. 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 $\text{minCtrAbs}$ decides that it is noneliminable. 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, $Q = (A, \Gamma)$ be a quantization for $\mathcal{H}$, and $(\mathcal{H}, I, G)$ be a DTLHS control problem.

In our approach we consider $\Gamma$ only in problems of type $P(W) \equiv (\max J(W), L(W) \land (\Gamma(W) = \hat{v}))$, 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 an MILP solver, we restrict ourselves to quantization functions $\gamma_{w_i}$ 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 $\Delta_w$ 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 Algorithm 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, $\hat{I} = \{\hat{x} | \text{feasible}(I(X) \land \Gamma(X) = \hat{x})\}$. Similarly, the quantization $\hat{G}$ of the goal region $G$ is $\hat{G} = \{\hat{x} | \text{feasible}(G(X) \land \Gamma(X) = \hat{x})\}$. 
Algorithm 2: Building control abstractions

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

Function minCtrAbs ($\mathcal{H}, Q$)

1: $\hat{N} \leftarrow \emptyset$
2: for all $\hat{x} \in \Gamma(\mathcal{A}_X)$ do
3: \hspace*{1em} for all $\hat{u} \in \Gamma(\mathcal{A}_U)$ do
4: \hspace*{2em} if $\neg Q$-admissible($\mathcal{H}, Q, \hat{x}, \hat{u}$) then continue
5: \hspace*{2em} if selfLoop($\mathcal{H}, Q, \hat{x}, \hat{u}$) then $\hat{N} \leftarrow \hat{N} \cup \{(\hat{x}, \hat{u}, \hat{x})\}$
6: \hspace*{2em} $O \leftarrow$ overImg($\mathcal{H}, Q, \hat{x}, \hat{u}$)
7: \hspace*{2em} for all $\hat{x}' \in \Gamma(O)$ do
8: \hspace*{3em} if $\hat{x}' \neq \hat{x}' \land \exists$ existsTrans($\mathcal{H}, Q, \hat{x}, \hat{u}, \hat{x}'$) then
9: \hspace*{3em} $\hat{N} \leftarrow \hat{N} \cup \{(\hat{x}, \hat{u}, \hat{x}')\}$
10: return $\hat{N}$

6.4. Computing Minimum Control Abstractions

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

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

For any pair $(\hat{x}, \hat{u})$ in $\Gamma(\mathcal{A}_X) \times \Gamma(\mathcal{A}_U)$ line 4 checks if $\hat{u}$ is $Q$-admissible in $\hat{x}$. This check is carried out by determining if the predicate $P(X, U, Y, X', \hat{x}, \hat{u}) \equiv N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land X' \notin \mathcal{A}_X$ is not feasible. If $\hat{u}$ is not $Q$-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 (Section 6.9).

If $\hat{u}$ is $Q$-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 Section 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 $O$, that is a quite tight overapproximation of the set of one-step reachable states from $\hat{x}$ via $\hat{u}$. $O$ is obtained by computing for each abstract variable $x_i$ the minimum and maximum possible values for the corresponding next state variable. Namely, $O = \prod_{i=1}^{\#x_i} [\gamma_x(m_i), \gamma_x(M_i)]$ where $m_i = \text{optimalValue} (\min, x_i', N(X, U, Y, X') \land A(X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u})$ and $M_i = \text{optimalValue} (\max, x_i', N(X, U, Y, X') \land A(X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u})$.

Finally, for each abstract state $x' \in \Gamma(O)$ 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 $N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x'}$.

Remark 6.5. From the nested loops in lines 2, 3, and 7 we have that minCtrAbs worst-case runtime is $O(|\Gamma(\mathcal{A}_X)|^2|\Gamma(\mathcal{A}_U)|)$. However, thanks to the heuristic implemented in function overImg, minCtrAbs typical runtime is about $O(|\Gamma(\mathcal{A}_X)||\Gamma(\mathcal{A}_U)|)$ as confirmed by our experimental results (see Section 8, Figure 7). The same holds for function minFullCtrAbs (see Section 6.4.1).

Remark 6.6. Function minCtrAbs is explicit in the (abstract) states and actions of $\mathcal{H}$ 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 \textit{minCtrAbs} can be easily modified in order to compute the close to minimum full control abstraction, thus obtaining function \textit{minFullCtrAbs} called in Algorithm 1, line 5. Function \textit{minFullCtrAbs} is obtained by removing the highlighted code (on grey background) from Algorithm 2, namely the admissibility check in line 4.

6.5. Self-Loop Elimination

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

\begin{algorithm}
\caption{Self-loop elimination}
\begin{algorithmic}[1]
\State \textbf{Input:} DTLHS \(H = (X, U, Y, N)\), quantization \(Q = (A, \Gamma)\), abstract state \(\hat{x}\), abstract action \(\hat{u}\).
\Function{selfLoop}{\(H, Q, \hat{x}, \hat{u}\)}
\If{\!
\exists\text{trans}(\hat{x}, \hat{u}, \hat{x})\}
\State \textbf{return} \textsc{False}
\EndIf
\For{\(x_i\in X\)}
\State \(w_i \leftarrow \text{optimalValue}(\min, x'_i - x_i, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})\)
\EndFor
\If{\(w_i < 0\)}
\State \textbf{return} \textsc{False}
\EndIf
\State \(W_i \leftarrow \text{optimalValue}(\max, x'_i - x_i, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})\)
\If{\(W_i < 0\)}
\State \textbf{return} \textsc{False}
\EndIf
\State \textbf{return} \textsc{True}
\EndFunction
\end{algorithmic}
\end{algorithm}

Function \textit{selfLoop} in Algorithm 3, whose correctness is proved in Section 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, \textit{selfLoop} returns \textsc{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 \textit{Zeno}-phenomena may arise, thus executing actions in \(\Gamma^{-1}(\hat{u})\) it is guaranteed that \(H\) will eventually leave the region \(\Gamma^{-1}(\hat{x})\). Otherwise, \textsc{True} is returned in line 7.

6.6. Proof of Function selfLoop Correctness

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

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

\textbf{Proof.} Suppose by absurd that the abstract self-loop \((\hat{x}, \hat{u}, \hat{x})\) has a concrete witness, \(\textit{selfLoop}(H, Q, \hat{x}, \hat{u})\) returns \textsc{False}, and \((\hat{x}, \hat{u}, \hat{x})\) is a noneliminable self-loop. Then there exists an infinite run \(\pi = x_0u_0x_1u_1\ldots\) such that for all \(t \in \forall x_i \in \Gamma^{-1}(\hat{x})\) and \(u_i \in \Gamma^{-1}(\hat{u})\).

For \(i \in [\lceil|X'|\rceil]\), let \(w_i \leq W_i\) be the values computed in lines 3 and 5 of Algorithm 3, that is, \(w_i = \text{optimalValue}(\min, x'_i - x_i, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})\) and \(W_i = \text{optimalValue}(\max, x'_i - x_i, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})\).
Since \( \text{selfLoop}(\mathcal{H}, Q, \hat{x}, \hat{u}) \) returns \( \text{FALSE} \), there exists at least an index \( j \in ||X'|| \) such that \( w_j > 0 \) or \( W_j < 0 \) (see lines 4 and 6 of Algorithm 3 respectively). Let us consider the former case (note that \( w_j > 0 \) implies \( W_j > 0 \)).

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

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

In both cases we have a contradiction, thus the thesis is proved. \( \square \)

6.7. Proof of Functions \( \text{minCtrAbs} \) and \( \text{minFullCtrAbs} \) Correctness

In this section we prove correctness of functions \( \text{minCtrAbs} \) (Algorithm 2) and \( \text{minFullCtrAbs} \) used in Algorithm 1. This section can be skipped at a first reading.

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

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

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

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

1. Each transition \((\hat{x}, \hat{u}, \hat{x}') \) is added to \( \hat{N} \) in line 5 or in line 9 of Algorithm 2. In both cases, it has been checked by function \( \text{existsTrans} \) 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 \( \text{selfLoop} \)).

2. Let \( x, x' \in A_X \) and \( u \in A_U \) be such that \( \exists y : N(x, u, y, x') \) and \( \Gamma(x) \neq \Gamma(x') \).

3. Note that condition 3 of Definition 5.3 may be rephrased as follows: if \((\hat{x}, \hat{u}, \hat{x}')\) is a noneliminable self-loop, then \( N(\hat{x}, \hat{u}, \hat{x}) \) must hold. That is, if \( N(\hat{x}, \hat{u}, \hat{x}) = 0 \) then 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 \( \text{selfLoop}(\mathcal{H}, Q, \hat{x}, \hat{u}) \) returns \( \text{FALSE} \) (respectively by line 1 of Algorithm 3 and by Proposition 6.7). Since a self-loop \((\hat{x}, \hat{u}, \hat{x})\) is not added to \( \hat{N} \) only if \( \text{selfLoop}(\mathcal{H}, Q, \hat{x}, \hat{u}) \) returns \( \text{FALSE} \) in line 5 of Algorithm 2, and since function \( \text{selfLoop}(\mathcal{H}, Q, \hat{x}, \hat{u}) \) is eventually invoked for all \( \hat{x} \in \Gamma(A_X) \) and \( \hat{u} \in \Gamma(A_U) \), the thesis is proved. \( \square \)

6.8. Proof of Synthesis Algorithm Correctness

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

**PROOF THEOREM 6.1.** If function \( q\text{CtrlSyn} \) returns \((\text{Sol}, \hat{D}, \hat{K})\), then function \( \text{minCtrAbs} \) has found an admissible \( Q \) control abstraction \( \hat{M} \) of \( M \) (see Proposition 6.8) and function \( \text{strongCtr} \) has found the strong mgc \( K \) to the control problem \((\mathcal{M}, \Gamma(I), \Gamma(G)) \). By
Theorem 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 Q QFC strong solution to the control problem $(\mathcal{H}, I, G)$.

If function $qCtrlSyn$ returns $(\text{NoSol}, \hat{D}, \hat{K})$, there is no weak solution to the control problem $(\mathcal{H}, I, G)$, where $\hat{V}$ is the close to minimum full control abstraction of $\mathcal{H}$ computed by function $\text{minFullCtrAbs}$ (Proposition 6.8). Therefore, by Theorem 5.9 (point 3) there is no Q QFC solution to the control problem $(\mathcal{H}, I, G)$. $\square$

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

Note that $X' \notin A_X$ is not a conjunctive predicate, thus feasibility of predicate $P(X, U, Y, X', \check{x}, \check{u})$ cannot be directly checked via function $\text{feasible}$. We implement such a check by calling $2|X|$ times function $\text{feasible}$ in the following way. For each $x' \in X'$, let $P_{x'}(X, U, Y, X', \check{x}, \check{u}) \equiv N(X, U, Y, X') \land \Gamma(X) = \check{x} \land \Gamma(U) = \check{u} \land x' \leq \inf X'$ and $P_{x'}(X, U, Y, X', \check{x}, \check{u}) \equiv N(X, U, Y, X') \land \Gamma(X) = \check{x} \land \Gamma(U) = \check{u} \land x' \geq \sup X'$. For each $x' \in X'$, we call function $\text{feasible}$ on $P_{x'}$ and $P_{x'}$ separately. If all such $2|X|$ calls return $\text{false}$, then $P$ is not feasible, otherwise $P$ is feasible.

Note that by Definition 5.3 we should also check that $\forall x \in \Gamma^{-1}(\check{x}) \forall u \in \Gamma^{-1}(\check{u}) \exists x' \in D_X \exists y \in D_Y : N(x, u, y, x')$. This cannot be checked via function $\text{feasible}$. 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 Algorithm 2, on the formula $\Phi(\check{x}, \check{u}) \equiv \exists x \in D_X \exists u \in D_U : \Gamma(X) = \check{x} \land \Gamma(U) = \check{u} \land [\exists x' \in D_X \exists y \in D_Y : N(x, u, y, x')]$. The output of Mjollnir is a formula $\Phi(\check{x}, \check{u})$ such that $\Phi(\check{x}, \check{u}) \equiv \Phi(\check{x}, \check{u})$ and $\Phi(\check{x}, \check{u})$ does not contain quantifiers (i.e., the only variables in $\Phi(\check{x}, \check{u})$ are $\check{x}$ and $\check{u}$). $\Phi(\check{x}, \check{u})$ is true if $\check{u}$ is not safe in $\check{x}$. Since $\Phi(\check{x}, \check{u})$ 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 $Q\text{-admissible}$ (line 4 of Algorithm 2) is invoked, it first checks if $(\check{x}, \check{u}) \in \hat{L}$. If this holds, then function $Q\text{-admissible}$ directly returns $\text{false}$. Otherwise, the previously described check (involving at most $2|X|$ calls to function $\text{feasible}$) is performed.

### 7. CONTROL SOFTWARE GENERATION

In this section we describe how we synthesize the actual control software (C functions $\text{ControlLaw}$ and $\text{ControllableRegion}$ in Section 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 $\text{obdd2c}(B)$ 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 $qCtrlSyn$ in Algorithm 1. We synthesize function $\text{ControllableRegion}$ by computing $\text{obdd2c}(\hat{D})$. As for function $\text{ControlLaw}$, let $r$ (respectively, $n$) be the number of bits used to represent plant actions (respectively states). We compute [Tronci 1998] a boolean function $F : \mathbb{B}^n \rightarrow \mathbb{B}$ that, for each quantized state $\check{x}$ in the controllable region $\hat{D}$, returns a quantized action $\hat{u}$ such that $\hat{K}(\check{x}, \hat{u})$ holds. Let $F : \mathbb{B}^n \rightarrow \mathbb{B}$ be the boolean function computing the $i$-th bit of $F$. That is, $F(\check{x}) = [F_1(\check{x}), \ldots, F_r(\check{x})]$. We take function $\text{ControlLaw}$ to be the (C implementation of) $\{\text{obdd2c}(F_1), \ldots, \text{obdd2c}(F_r)\}$.
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 Controlable Region [Control Law] is less than or equal to \( n \cdot T_B \). Thus, neglecting I/O times, each iteration of the control loop (see Figure 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 (Figure 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, beforehand, 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 name the resulting tool Quantized feedback Kontrol Synthesizer (QKS) (publicly available at QKS [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 Sections 8.1, 8.2, and 8.3 we present results for the buck DC-DC converter case study. In Sections 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 Section 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 Example 3.2 and Section 8.1).

8.1. Buck DC-DC Converter: Experimental Settings

In this section (and in Sections 8.2, 8.3) we present experimental results obtained by using QKS on a version of the buck DC-DC converter described in Section 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, 2012b]. We denote with \( \mathcal{H} = (X, U, Y, \tilde{N}) \) the DTLHS modeling such a converter, where \( X, U \) are as in Section 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 Section 3.1 as follows. As for variations in the power supply \( V_i \), they are modeled analogously to Example 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 \( A_{x_L} = [-4, 4] \) and
Table I. Buck DC-DC Converter (Section 3): Control Abstraction & Controller Synthesis Results. Part I

| 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 (Section 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  |

\[A_{v_0} = [-1, 7]\]. We define \(A = A_{iL} \times A_{v_0} \times A_v\). As for auxiliary variables, we use the following safety bounds: \(A_{iL} = A_{iD} = [-10^3, 10^3]\) and \(A_v = A_{v_0} = [-10^7, 10^7]\). As a result, we add 12 further constraints to \(\bar{N}\) stating that \(\bigwedge_{w \in \{i_L, v_0, i_D, n_s, n_v\}} w \in A_v\), thus obtaining a bounded DTLHS [Mari et al. 2011c].

Finally, the initial region \(I\) and goal region \(G\) are as in Example 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 \((i_L, v_0)\) into \(2^b\) equal intervals, where \(b\) is the number of bits used by AD conversion, thus with respect to Section 6.3 we have that \(\Delta_{iL} = \Delta_{v_0} = 2^b\). The resulting quantization is \(Q_b = (A, \Gamma_b)\), with \(\|\Gamma_b\| = 2^{3-b}\). Since we have two quantized variables \((i_L, v_0)\) 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 Algorithm 1, on the control problem \((\mathcal{H}, I, G)\) with quantization \(Q_b\). In the following, we will call \(\mathcal{M}_b\) the close to minimum (admissible) \(Q_b\) control abstraction for \(\mathcal{H}\), \(\mathcal{H}_b\) the maximum (full) \(Q_b\) control abstraction for \(\mathcal{H}\) (which we compute for statistical reasons also when Algorithm 1 returns Sol), \(\hat{K}_b\) the strong mgo for \(P_b = (\mathcal{M}_b, \varnothing, \Gamma_b(G))\), \(\hat{D}_b = \text{Dom}(\hat{K}_b)\) the controllable region of \(\hat{K}_b\), and \(K_b(s, u) = \hat{K}_b(\Gamma_b(s), \Gamma_b(u))\) the \(Q_b\) QFC solution to \(P_b = (\mathcal{H}, \Gamma_b^{-1}(\hat{D}_b), G)\). All our experiments have been carried out on a 3.0 GHz Intel hyperthreaded Quad Core Linux PC with 8GB 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 Section 6.

Tables I, II, III, and IV show our experimental results for QKS (and thus for Algorithm 1). Columns in Table I have the following meaning. Column \(b\) shows the number of AD bits. Columns labeled Control Abstraction show performance for Algorithm 2 (computation of \(\mathcal{M}_b\)) and they show running time (column CPU, in secs), memory usage (MEM, in bytes), the number of transitions in \(\mathcal{M}_b\) (Arcs), the number of self-loops in \(\mathcal{H}_b\) (MaxLoops), and the fraction of self-loops that are kept in \(\mathcal{M}_b\) with respect to the number of self-loops in \(\mathcal{H}_b\) (LoopFrac). Columns labeled Controller Synthesis show...
Table III. Buck DC-DC Converter: Number of MILPs and Time to Solve Them (secs). Part I

| MILP | $b = 8$ | $b = 9$ |
|------|---------|---------|
|      | 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

| MILP | $b = 10$ | $b = 11$ |
|------|----------|----------|
|      | 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+02 | 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 |

the computation time (column $CPU$, in secs) for the generation of $\hat{K}_d$, and the size of its OBDD representation ($|K|$, number of nodes). The latter is also the size (number of lines) of $\hat{K}_d$ C code synthesized implementation. Columns in Table 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 $\mu \in \{\text{SOL}, \text{NO\,SOL}, \text{UNK}\}$ of Algorithm 1.

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

8.2.1. MILP Problems Analysis. For each MILP problem solved in QKS, Tables 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 Section 6.3, that is, those computing the quantization for $I$ and $G$, MILP2 refers to MILP problems in function $SelfLoop$ (see Algorithm 3), MILP3 refers to the MILP problems used in function $overImg$ (line 6 of Algorithm 2), MILP4 refers to MILP problems used to check actions admissibility (line 8 of Algorithm 2), and MILP5 refers to MILP problems used to check transitions witnesses (line 4 of Algorithm 2). Columns in Tables 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), that is, the ratio between columns Time and Num.

CPU time standard deviation is always less than 0.003.

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

From Tables III and IV, column Avg, we see that the average time spent solving each MILP instance is small. Figure 8 graphically shows that MILP average computation time does not heavily depend on $b$. As observed in Remark 6.5, Figure 7 shows that the number of MILP4 invocations is much closer to $|\Gamma(\hat{A}_X)||\Gamma(\hat{A}_U)| = 2^{2b+1}$, rather than the
8.3. Buck DC-DC Converter: Control Software Performance

In this section we discuss the performance of the generated controller. Figure 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 (Theorem 6.1). This is the controllable region $\hat{D}$ returned by Algorithm 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. Figure 9 shows the controllable region $D_{10} = \Gamma^{-1}(\hat{D}_{10})$ 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 results in a theoretical worst-case running time $|\Gamma(A_X)|^2 |\Gamma(A_U)| = 2^{4b+1}$ of Algorithm 2. This shows effectiveness of function overimg heuristic.
Fig. 11. Controller performances for the Buck DC-DC converter: setup time and ripple.

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 approach, as discussed in Henzinger [2010] and 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. Figure 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 Figure 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. Figure 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 $V_{ref} = 5$ 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] and Yousefzadeh et al. [2008]).

8.4. Inverted Pendulum: Experimental Settings

In this section (and in Sections 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 Figure 12. The system is modeled by taking the angle $\theta$ 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 $\theta$ and $x_2$ is the angular speed $\dot{\theta}$.

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

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

Differently from Kreisselmeier and Birkhölder [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 Eqs. (11) and (12) as the Euler approximation with sampling time $T$, that is, the predicate $(x'_1 = x_1 + Tx_2) \land (x'_2 = x_2 + T \frac{g}{l} \sin x_1 + T \frac{1}{ml^2} Fu)$.

Since the system whose dynamics are in Eqs. (11) and (12) is not linear, we build a linear overapproximation of it as shown in Alimguzhin et al. [2012a]. The result is the DTLHS $\mathcal{H}$ defined in Example 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 Birkhölder [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 $D_{x_1} = [-1.1\pi, 1.1\pi]$ (we write $\pi$ for a rational approximation of it) and $D_{x_2} = [-4, 4]$ 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 $G(X) \equiv (-\rho \leq x_1 \leq \rho) \land (-\rho \leq x_2 \leq \rho)$, where $\rho \in \{0.05, 0.1\}$, and the initial region is defined by the predicate $I(X) \equiv (-\pi \leq x_1 \leq \pi) \land (-4 \leq x_2 \leq 4)$.

We run QKS on the control problem $(\mathcal{H}, I, G)$ for different values of the remaining parameters, that is, $\rho$ (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) and (12), thus simulating a path of
Table V. Inverted Pendulum: Control Abstraction & Controller Synthesis Results with $F = 0.5$

| $b$ | $T$ | $\rho$ | $|K|$ | 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  |

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

$H^{(K)}$. Such simulator also introduces random disturbances (up to 4%) in the next state computation to assess $K$ robustness with respect to nonmodeled disturbances. Finally, in the simulator Eqs. (11) and (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 $H$. 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 $H$. 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 @ 2.27 GHz, 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 nonobvious 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 $H$ with $F = 0.5$ (note that in Kreisselmeier and Birkhölzer [1994] $F = 0.7$). Results are in Table V, where each row corresponds to a QKS run, columns $b$, $T$, and $\rho$ 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 setup time. Figure 14(a) shows the simulations of $H^{(K_9)}$ and $H^{(K_{10})}$. As we can see, $K_{10}$ drives the system to the goal with a smarter trajectory, with one swing only. This has a significant impact on the setup time (the system stabilizes after about
8 seconds when controlled by $K_{10}$ instead of about 10 seconds required when controlled by $K_9$). Figure 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, Figure 14(b) shows the ripple of $x_1$ for $h^{(K_0)}$ inside the goal. Note that such ripple is very low (0.018 radians).

9. RELATED WORK

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

Section 9.1 compares our contribution with related work on control software synthesis from system-level formal specifications. For the sake of completeness, Section 9.2 expands such a comparison to recent results on (noncontrol) software synthesis from formal specifications, focusing on papers using techniques related to ours (constraint solving, OBDD, supervisory control [Ramadge and Wonham 1987]). Section 9.3 describes Table 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 Section 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 ours is Kreisselmeier and Birkholzer [1994] which studies the problem of control synthesis for discrete-time hybrid systems. However, while we present an automatic method, the approach in Kreisselmeier and Birkholzer [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] apply themselves 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] do not supply an effective method to generate control software (as we do in Section 6.1). As a consequence Mazo and Tabuada [2011] give no guarantee on WCET, an important issue since an SBCS is a hard real-time system.
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                      |     |               |
|                           |     | Piecewise affine              |     |               |
|                           |     | Linear hybrid sys.            |     |               |
|                           |     | Nonlinear                    |     |               |
|                           |     | Quantization                  |     |               |
|                           |     | Formally verified WCP        |     |               |
|                           |     | 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]       | •   | •                           | •••••|               |
| [Tronci 1996]            | •   | •                           | •••••|               |
| [Tronci 1997]            | •   | •                           | •••••|               |
| [Tronci 1998]            | •   | •                           | •••••|               |
| [Tronci 1999b]           | •   | •                           | •••••|               |
| [Tronci 1999c]           | •   | •                           | •••••|               |
| [Wong-Toi 1997]          | *   | •                           | ••••|               |

Girard et al. [2010] present a method to find an overapproximation 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 Figure 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 Figure 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], and Peter et al. [2011] and citations thereof. When the plant model is a Linear Hybrid Automaton (LHA) [Alur et al. 1995, 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], and 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 aforementioned 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 an 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] and Della Penna et al. [2008] present 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] present 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 Section 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 article and computing suitable overapproximations 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, 1998, 1999b, 1999a]. An automatic method for the generation of supervisory controllers for finite
state systems is presented in Tronci [1996]. Control software synthesis in nondeterministic 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 Section 6.1 we presented our QFC synthesis algorithm (Algorithm 1). Line 3 of Algorithm 1 calls function \texttt{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 nontime-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 \texttt{strongCtr}. Thus, Alimguzhin et al. [2012b] is not an improvement to the present article 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] and Jha et al. [2010, 2011] and citations thereof. Such approaches, by ignoring the quantization process, indeed focus on the control law design (see Section 1.1). However we note that Jha et al. [2010, 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 \textit{control abstraction}), which has been widely studied in a hybrid system formal verification context (e.g., see Alur et al. [2000, 2006], Tiwari [2008], and 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 \textit{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 a long time in many contexts. For examples, see Pnueli and Rosner [1989a, 1989b], Schewe and Finkbeiner [2006], and Girault and Rutten [2009]. We give a glimpse of recent results on (noncontrol) software synthesis approaches using techniques related to ours (constraint solving, OBDD, supervisory control).

Attie et al. [2004] show how to mechanically synthesize fault-tolerant concurrent programs for various fault classes. Srivastava et al. [2010] present 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] address the problem of synthesizing loop-free programs starting from logical relations between input and output variables. Srivastava et al. [2011] propose a synthesis technique and apply it to the inversion of imperative programs (e.g., such as insert/delete operations, compressors/decompressors). Cerný et al. [2011] present 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, 2010] and Gvero et al. [2011].

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

Table 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 Section 9.1 but the ones focusing on abstraction (since Section 9.2 results do not address control software synthesis).

Table 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 presented method. Note that Girard et al. [2010] and Pola et al. [2007] in Table 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).

ACRONYMS

AD. Analog-to-Digital
COBDD. OBDD with complemented edges
DA. Digital-to-Analog
DTLHS. Discrete-Time Linear Hybrid System  
DVFS. Dynamic Voltage and Frequency Scaling  
HIL. Hardware-In-the-Loop  
LHA. Linear Hybrid Automaton  
LLHA. Lazy Linear Hybrid Automaton  
LTS. Labeled Transition System  
MILP. Mixed Integer Linear Programming  
NDTCM. Non-Deterministic Two-Counter Machine  
OBDD. Ordered Binary Decision Diagram  
PWA-DTHS. Piecewise Affine Discrete Time Hybrid Systems  
QFC. Quantized Feedback Control  
QKS. Quantized feedback Kontrol Synthesizer  
SBCS. Software-Based Control System  
TA. Timed Automaton  
WCET. Worst-Case Execution Time

REFERENCES

M. Agrawal, F. Stephan, P. S. Thiagarajan, and S. Yang. 2006. Behavioural approximations for restricted linear differential hybrid automata. In Proceedings of the 9th International Workshop on Hybrid Systems: Computation and Control (HSCC'06). J. P. Hespanha and A. Tiwari, Eds., Lecture Notes in Computer Science, vol. 3927, Springer, 4–18.

M. Agrawal and P. S. Thiagarajan. 2005. The discrete time behavior of lazy linear hybrid automata. In Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05). M. Morari and L. Thiele, Eds., Lecture Notes in Computer Science, vol. 3414, Springer, 55–69.

V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci. 2012a. Automatic control software synthesis for quantized discrete time hybrid systems. In Proceedings of the Conference on Decision and Control (CDC’12). To appear. (A preliminary version can be found at http://arxiv.org/abs/1207.4098.)

V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci. 2012b. On model based synthesis of embedded control software. In Proceedings of the International Conference on Embedded Software (EMSOFT’12). To appear.

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

R. Alur, T. Dang, and F. Ivančić. 2006. Predicate abstraction for reachability analysis of hybrid systems. ACM Trans. Embed. Comput. Syst. 5, 1, 152–199.

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

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

R. Alur and P. Madhusudan. 2004. Decision problems for timed automata: A survey. In Proceedings of the International School on Formal Methods for the Design of Computer Communication. Lecture Notes in Computer Science, vol. 3185, Springer, 1–24.

E. Asarin and O. Maler. 1999. As soon as possible: Time optimal control for timed automata. In Proceedings of the International Workshop on Hybrid Systems: Computation and Control (HSCC’99). Lecture Notes in Computer Science, vol. 1569, Springer, 19–30.

P. C. Attie, A. Arora, and E. A. Emerson. 2004. Synthesis of fault-tolerant concurrent programs. ACM Trans. Program. Lang. Syst. 26, 1, 125–185.

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

A. Bemporad and N. Giorgiotti. 2004. A sat-based hybrid solver for optimal control of hybrid systems. In Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC’04). Lecture Notes in Computer Science, vol. 2993, Springer, 126–141.

ACM Transactions on Software Engineering and Methodology, Vol. 23, No. 1, Article 6, Pub. date: February 2014.
Model-Based Synthesis of Control Software from System-Level Formal Specifications

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

W. Kim, M. S. Gupta, G.-Y. Wei, and D. M. Brooks. 2007. Enabling on-chip switching regulators for multicore processors using current staggering. In Proceedings of the Workshop on Architectural Support for Gigascale Integration (ASGI'07).

G. Kreisselmeier and T. Birkholzer. 1994. Numerical nonlinear regulator design. IEEE Trans. Autom. Control 39, 1, 33–46.

V. Kuncak, M. Mayer, R. Piskac, and P. Suter. 2010. Comfy: A tool for complete functional synthesis. In Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10), T. Touili, B. Cook, and P. Jackson, Eds., Lecture Notes in Computer Science, vol. 6174, Springer, 430–433.

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

K. G. Larsen, P. Pettersson, and W. Yi. 1997. Uppaal: Status and developments. In Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97). Lecture Notes in Computer Science, vol. 1254, Springer, 456–459.

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

O. Maler, D. Nickovic, and A. Pnueli. 2007. On synthesizing controllers from bounded-response properties. In Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07). Lecture Notes in Computer Science, vol. 4590, Springer, 95–107.

F. Mari, I. Melatti, I. Salvo, and E. Tronci. 2010. Synthesis of quantized feedback control software for discrete time linear hybrid systems. In Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10). Lecture Notes in Computer Science, vol. 6174, Springer, 180–195.

F. Mari, I. Melatti, I. Salvo, and E. Tronci. 2011a. From boolean relations to control software. In Proceedings of the 6th International Conference on Software Engineering Advances (ICSEA’11).

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

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

F. Mari, I. Melatti, I. Salvo, and E. Tronci. 2012a. Control software visualization. In Proceedings of the 2nd International Conference on Advanced Communications and Computation (INFOCOMP’12). ThinkMind, 15–20.

F. Mari, I. Melatti, I. Salvo, and E. Tronci. 2012b. Linear constraints as a modeling language for discrete time hybrid systems. In Proceedings of the 7th International Conference on Software Engineering Advances. ThinkMind, 664–671.

F. Mari, I. Melatti, I. Salvo, and E. Tronci. 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’12). A. Roychoudhury and M. D’Souza, Eds., Lecture Notes in Computer Science, vol. 7521, Springer, 243–258.

M. Mazo, A. Davitian, and P. Tabuada. 2010. Pessoa: A tool for embedded controller synthesis. In Proceedings of the 22nd International Conference on Computer Aided Verification (CAV’10). Lecture Notes in Computer Science, vol. 6174, Springer, 566–569.

M. J. Mazo and P. Tabuada. 2011. Symbolic approximate time-optimal control. Syst. Control Lett. 60, 4, 256–263.

D. Monniaux. 2010. Quantifier elimination by lazy model enumeration. In Proceedings of the 22nd International Conference on Computer Aided Verification (CAV’10). Lecture Notes in Computer Science, vol. 6174, Springer, 585–599.

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

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

A. Platzer and E. M. Clarke. 2009. Formal verification of curved flight collision avoidance maneuvers: A case study. In Proceedings of the 2nd World Congress on Formal Methods (FM’09). A. Cavalcanti and D. Dams, Eds., Lecture Notes in Computer Science, vol. 5850, Springer, 547–562.

A. Pnueli and R. Rosner. 1989a. On the synthesis of a reactive module. In Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL’89). ACM Press, New York, 179–190.

ACM Transactions on Software Engineering and Methodology, Vol. 23, No. 1, Article 6, Pub. date: February 2014.
A. Pnueli and R. Rosner. 1989b. On the synthesis of an asynchronous reactive module. In Proceedings of the Conference on Automata, Languages and Programming (ICALP’89). G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, Eds., Lecture Notes in Computer Science, vol. 372, Springer, 652–671.

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

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

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

S. Sankaranarayanan and A. Tiwari. 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.estereotechnologies.com/products/scade-system/.

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

E. Tronci. 1996. Optimal finite state supervisory control. In Proceedings of the 35th IEEE Conference on Decision and Control (CDC’96).

E. Tronci. 1997. On computing optimal controllers for finite state systems. In Proceedings of the 36th IEEE Conference on Decision and Control (CDC’97). Vol. 4, 3592–3593.

E. Tronci. 1998. Automatic synthesis of controllers from formal specifications. In Proceedings of the 2nd International Conference on Normal Engineering Methods (ICFEM’98). 134–143.

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

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