Synthesis of Quantized Feedback Control Software for Discrete Time Linear Hybrid Systems

Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci

Dip. di Informatica Sapienza Università di Roma,
Via Salaria 113, 00198 Roma, Italy
{mari, melatti, salvo, tronci}@di.uniroma1.it

Abstract. We present an algorithm that given a Discrete Time Linear Hybrid System $H$ returns a correct-by-construction software implementation $K$ for a (near time optimal) robust quantized feedback controller for $H$ along with the set of states on which $K$ is guaranteed to work correctly (controllable region). Furthermore, $K$ has a Worst Case Execution Time linear in the number of bits of the quantization schema.

1 Introduction

Software generation from models and formal specifications forms the core of model based design of embedded software [18]. This approach is particularly interesting for control systems since in such a case system specifications are much easier to define than the control software behavior itself.

A control system consists of two subsystems (forming the closed loop system): the controller and the plant. In an endless loop the controller measures outputs from and sends commands to the plant in order to drive it towards a given goal. In our setting the controller consists of software implementing the control law. System requirements are typically given as specifications for the closed loop system. Control engineering techniques are used to design the control law (i.e. the functional specifications for the control software) from the closed loop system specifications. Software engineering techniques are then used to design control software implementing a given control law.

Unfortunately, when the plant model is a hybrid system [4, 1, 3] existence of a control law is undecidable (e.g. see [17]) even for linear hybrid automata. This scenario is further complicated by the quantization process always present in software based control systems. Namely, measures from sensors go through an AD (analog-to-digital) conversion before being sent to the control software and commands from the control software go through a DA (digital-to-analog) conversion before being sent to plant actuators. Furthermore, typically a robust control is desired, that is, one that meets the given closed loop requirements notwithstanding (nondeterministic) variations in the plant parameters.

As for hybrid systems, no approach is available for the automatic synthesis of robust quantized feedback control laws and of their software implementation. This motivates the focus of our paper.
Our Main Contributions A Discrete Time Linear Hybrid System (DTLHS) is a discrete time hybrid system whose dynamics is defined as the logical conjunction of linear constraints on its continuous as well as discrete variables.

We present an effective algorithm that, given a DTLHS model $\mathcal{H}$ for the plant and a quantization scheme (i.e., how many bits we use for AD conversion), returns a pair $(K, R)$, where: $K$ is a correct-by-construction software implementation (C language in our case) of a (near time optimal) Quantized Feedback Controller (QFC) for $\mathcal{H}$ and $R$ is an OBDD [10] representation of the set of states (controllable region) on which $K$ is guaranteed to meet the closed loop requirements. Furthermore, $K$ is robust with respect to nondeterministic variations in the plant parameters and has a Worst Case Execution Time (WCET) guaranteed to be linear in the number of bits of the quantization schema.

We implemented our algorithm on top of the CUDD package and of the GLPK Mixed Integer Linear Programming (MILP) solver and present experimental results on using our tool to synthesize robust QFCs for a widely used mixed-mode analog circuit: the buck DC-DC converter (e.g., see [26]).

Analog DC-DC converters are a vital part of many mission (e.g. satellites) or safety (e.g., aircrafts) critical applications. However, the ever increasing demand for energy efficiency and easy reconfigurability makes fully software based switching converters (e.g., as in [26]) a very attractive alternative to analog ones. Unfortunately, lack of formal reliability assessment (in order to bound the failure probability to $10^{-5}$) limits the deployment of software based switching converters in safety critical applications. Reliability analysis for switching converters using an analog control schema has been studied in [13]. For software based converters, carrying out such a reliability analysis entails formal verification of the control law as well as of its software implementation. The above considerations make the buck DC-DC converter a very interesting (and challenging) example for automatic synthesis of correct-by-construction control software.

Our experimental results show that within about 20 hours of CPU time and within 200MB of RAM we can synthesize $(K, R)$ as above for a 10 bit quantized buck DC-DC converter.

Related Work Synthesis of Quantized Feedback Control Laws for linear systems has been widely studied in control engineering (e.g. see [14]). However, to the best of our knowledge, no previously published paper addresses synthesis of Quantized Feedback Control Software for DTLHSs. Indeed, our work differs from previously published ones in the following aspects: (1) we provide a tool for automatic synthesis of correct-by-construction control software (rather than design methodologies for the control law); (2) we synthesize robust control software (thus encompassing quantization) whereas robust control law design techniques do not take into account the software implementation; (3) in order to generate provably correct software, we assume a nondeterministic (malicious) model for quantization errors rather than a stochastic one, as usually done in control engineering; (4) our synthesis tool also returns the controllable region, that is the set of states on which the synthesized control software is guaranteed to work correctly (this is very important for Fault Detection Isolation and Recovery, FDIR, e.g. see [21]). In the following we discuss some related literature.
Quantization can be seen as a form of abstraction, where the abstract state space and transitions are defined by the number of bits of AD conversion. Abstraction for hybrid systems has been widely studied. For example, see [25, 2, 20, 19] and citations thereof. Note however that all published literature on abstraction focuses on designing abstractions to support verification or control law design. In our case instead, the abstraction is fully defined by the AD conversion schema and our focus is on devising techniques to effectively remove abstract transitions in order to counteract the nondeterminism (information loss) stemming from the quantization process.

Control synthesis for Timed Automata (TA) [4], Linear Hybrid Automata (LHA) [1, 3] as well as nonlinear hybrid systems has been extensively studied. Examples are in [22, 11, 6, 30, 16, 28, 5, 9] and citations thereof. We note however that all above papers address design of control laws and do not take into account the quantization process, that is, they assume exact (i.e. real valued) state measures. Here instead we address design of quantized feedback control software.

Correct-by-construction software synthesis in a finite state context has been studied in [7, 29, 27, 12]. The above approaches cannot be directly used in our context since they do not account for continuous state variables.

2 Background

Unless otherwise stated each variable \( x \) ranges on a known bounded interval \( \mathcal{D}_x \) either of the reals or of the integers (discrete variables). We denote with \( \sup(x) \) (\( \inf(x) \)) the sup (inf) of \( \mathcal{D}_x \). Boolean variables are discrete variables ranging on the set \( \mathcal{B} = \{0, 1\} \). We denote with \( X = [x_1, \ldots, x_n] \) a finite sequence (list) of variables, with \( \cup \) list concatenation and with \( \mathcal{D}_X = \prod_{x \in X} \mathcal{D}_x \) the domain of \( X \).

A valuation \( X^* \in \mathcal{D}_X \) over a list of variables \( X \) is a function \( v \) that maps each variable \( x \in X \) to a value \( v(x) \in \mathcal{D}_x \). We may use the same notation to denote a variable (a syntactic object) and one of its valuations. The intended meaning will be always clear from the context. To clarify that a variable \( x \) valued \( v \) is real (integer, boolean) valued we may write \( x^r(x^d, x^b) \). Analogously \( X^r(X^d, X^b) \) denotes the sequence of real (integer, boolean) variables [valuations] in \( X \).

If \( x \) is a boolean variable [valuation] we write \( \bar{x} \) for \((1-x)\).

A linear expression (over \( X \)) is a linear combination with real coefficients of variables in \( X \). A constraint (over \( X \)) is an expression of the form \( \alpha \bowtie b \) where \( \alpha \) is a linear expression over \( X \), \( \bowtie \) is one of \( \leq, \geq \), and \( b \) is a real constant.

A constraint is a predicate on \( X \). If \( A(X) \) and \( B(X) \) are predicates on \( X \), then \( A(X) \land B(X) \) and \( A(X) \lor B(X) \) are predicates on \( X \). A conjunctive predicate is just a conjunction of linear constraints. A satisfying assignment to \( P(X) \) is a valuation \( X^* \) such that \( P(X^*) = 1 \). Abusing notation we may denote with \( P \) the set of satisfying assignments to \( P(X) \). Given a predicate \( P(X) \) and a fresh boolean variable \( y \notin X \), the if-then predicate \( y \rightarrow P(X) [\bar{y} \rightarrow P(X)] \) denotes the predicate \( ((y = 0) \lor P(X)) [((y = 1) \lor P(X))] \). In our setting (bounded variables), for any predicate \( P(X) \) there exists a sequence \( Z \) of fresh boolean variables and a conjunctive predicate \( Q(Z, X) \) s.t. \( \forall X[P(X) \iff \exists Z Q(Z, X)] \) (see [23] for details). Thus, any if-then predicate can be transformed into a conjunctive predicate. Accordingly, we will regard and use if-then predicates as conjunctive predicates.
A Mixed Integer Linear Programming (MILP) problem with decision variables $X$ is a tuple $(\text{max}(J(X), A(X)))$ 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 $(\text{max}(J(X), A(X)))$ is a valuation $X^*$ s.t. $A(X^*)$ holds and, for any valuation $\Xi$, $(A(\Xi) \rightarrow (J(\Xi) \leq J(X^*)$). We write $(\text{min}(J(X), A(X)))$ for $(\text{max}, -J(X), A(X))$. A feasibility problem is a MILP problem of the form $(\text{max}, 0, A(X))$. We write also $A(X)$ for $(\text{max}, 0, A(X))$.

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, $T : S \times A \times S \rightarrow \mathbb{B}$ is the transition relation of $S$. Let $s \in S$, $a \in A$. 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 $\text{Img}(S, s, a) = \{s' \in S \mid T(s, a, s')\}$. A run or path for $S$ 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 run $\pi$ is the number of actions in $\pi$. We denote with $\pi^{(s)}(t)$ the $t$-th state element of $\pi$, and with $\pi^{(a)}(t)$ the $t$-th action element of $\pi$. That is $\pi^{(s)}(t) = s(t)$, and $\pi^{(a)}(t) = a(t)$.

3 Discrete Time Linear Hybrid Systems

In this section we introduce Discrete Time Linear Hybrid Systems (DTLHS).

**Definition 1.** A Discrete Time Linear Hybrid System (DTLHS) is a tuple $H = (X, U, Y, N)$ where:
- $X = X^r \cup X^d$ is a finite sequence of real ($X^r$) and discrete ($X^d$) present state variables. We denote with $X'$ the sequence of next state variables obtained by deconstructing with all variables in $X$.
- $U = U^r \cup U^d$ is a finite sequence of input variables.
- $Y = Y^r \cup Y^d$ is a finite sequence of auxiliary variables. Auxiliary variables are typically used to model modes (e.g., from switching elements such as diodes) or uncontrollable inputs (e.g., disturbances).
- $N(X, U, Y, X')$ is a conjunctive predicate over $X \cup U \cup Y \cup X'$ defining the transition relation (next state) of the system.

Note that in our setting (bounded variables) any predicate can be transformed into a conjunctive predicate (Sect. 2). Accordingly, in Def. 1, without loss of generality we focused on conjunctive predicates in order to simplify our exposition.

The dynamics of a DTLHS $H = (X, U, Y, N)$ is defined by $\text{LTS}(H) = (D_X, D_U, N)$ where: $N : D_X \times D_U \times D_X \rightarrow \mathbb{B}$ is a function s.t. $N(s, a, s') = \exists y \in D_Y N(s, a, y, s')$. A state for $H$ is a state for $\text{LTS}(H)$ and a run (or path) for $H$ is a run for $\text{LTS}(H)$ (Sect. 2).

**Example 1.** Let $H = (\{x\}, \{u\}, \emptyset, N)$ with $D_x = [-2.5, 2.5]$, $D_u = \{0, 1\}$, and $N(x, u, x') = [u \rightarrow x' = \alpha x] \land [u \rightarrow x' = \beta x]$ with $\alpha = \frac{1}{3}$ and $\beta = \frac{2}{3}$. When $Y = \emptyset$ (as here) for the sake of simplicity we omit it from $N$ arguments.

Adding nondeterminism to $H$ allows us to synthesize robust controllers. For example, variations in the parameter $\alpha$ can be modelled with a tolerance $\rho \in [0, 1]$ (e.g., $\rho = 0.5$) for $\alpha$. This replaces $N$ with: $N^{\rho} = [u \rightarrow x' \leq (1 + \rho)\alpha x] \land [u \rightarrow x' \geq (1 - \rho)\alpha x] \land [u \rightarrow x' = \beta x]$. Suitable control synthesis on $H^{\rho} = (\{x\}, \{u\}, \emptyset, N^{\rho})$ will yield a robust (up to $\rho$) controller for $H$. 
Example 2. The buck DC-DC converter (right part of Fig. 1) is a mixed-mode analog circuit converting the DC input voltage \((V_i\) in Fig. 1) to a desired DC output voltage \((V_o\) in Fig. 1). The typical software based approach (e.g., see [26]) is to control the switch \(u\) in Fig. 1 (typically implemented with a MOSFET) with a microcontroller. Designing the software to run on the microcontroller to properly actuate the switch is the control design problem for the buck DC-DC converter in our context. The circuit in Fig. 1 can be modeled as a DTLHS \(\mathcal{H} = (X, U, Y, N)\) with: \(X = X^r = [i_L, v_o]\), \(U = U^d = [u]\), \(Y = Y^r \cup Y^d\) with \(Y^r = [i_u, v_u, i_D, v_D]\) and \(Y^d = [q]\). \(H\) auxiliary variables \(Y\) stem from the constitutive equations of the switching elements (i.e., the switch \(u\) and the diode \(D\) in Fig. 1). The transition relation \(N(X, U, Y, X')\) for \(\mathcal{H}\) is shown in Fig. 1 (left), where we use a discrete time model with sampling time \(T\) (writing \(x'\) for \(x(t + 1)\)) and model a tolerance \(\rho = 0.25\) (25%) on \(V_i\) values. In Fig. 1 (left), constants \(a_{i,j}, b_{i,j}\) depend on the circuit parameters \(R, R_L, R_C, L, C\) and algebraic constraints (i.e., constraints not involving next state variables) stem from the constitutive equations of the switching elements (see [23] for details).

\[
N(X, U, Y, X') = ((i_L') = (1 + T a_{1,1})i_L + T a_{1,2}v_o + T b_{1,1}v_D) + v_{i_u} \\
&\wedge (v_o' = T a_{2,1}i_L + (1 + T a_{2,2})v_o + T b_{2,1}v_D) \\
&\wedge (v_u - v_D \leq (1 + \rho)V_i) \wedge (v_u - v_D \geq (1 - \rho)V_i) \\
&\wedge (i_D = i_L - i_u) \wedge (q \rightarrow v_D = 0) \wedge (q \rightarrow v_D = 0) \\
&\wedge (u \rightarrow v_u = 0) \wedge (u \rightarrow v_u = 0) \\
&\wedge (u \rightarrow v_u = 0) \wedge (u \rightarrow v_u = 0)
\]

![Fig. 1. Buck DC-DC converter](image-url)

4 Quantized Feedback Control Problem for DTLHS

We define the Feedback Control Problem for LTs (Def. 2) and for DTLHSs (Def. 4). On such a base we define the Quantized Feedback Control Problem for DTLHSs (Def. 6).

We begin by extending to (possibly infinite) LTs the definitions in [29, 12] for finite LTs. In what follows, let \(S = (S, A, T)\) be an LTS, \(I, G \subseteq S\) be, respectively, the initial and goal sets of \(S\).

**Definition 2.** A controller for \(S\) is a function \(K : S \times A \rightarrow \mathbb{B}\) s.t. \(\forall s \in S, \forall a \in A\), if \(K(s, a)\) then \(3s' T(s, a, s')\). Dom\((K)\) denotes the set of states for which at least a control action is enabled. Formally, Dom\((K) = \{s \in S \mid \exists a K(s, a)\}\). \(S^{(K)}\) denotes the closed loop system, that is the LTS \((S, A, T^{(K)})\), where \(T^{(K)}(s, a, s') = T(s, a, s') \wedge K(s, a).\) A control problem for \(S\) is a triple \((S, I, G)\).

A controller for \(S\) (Def. 2) is used to restrict \(S\) behavior so that all states in the initial region \((I)\) will reach in one or more steps the goal region \((G)\). In the following, we formalize such a concept by defining strong and weak solutions to an LTS control problem.

We call a path \(\pi\) fullpath [7] if either it is infinite or its last state \(\pi^{(S)}(|\pi|)\) has no successors. We denote with Path\((s)\) the set of fullpaths starting in state \(s\), i.e., the set of fullpaths \(\pi\) such that \(\pi^{(S)}(0) = s\). Observe that Path\((s)\) is never empty, since it contains at least the path of length 0 containing only state \(s\).

Given a path \(\pi\) in \(S\), \(J(S, \pi, G)\) denotes the unique \(n > 0\), if it exists, s.t. \(\pi^{(S)}(n) \in G\) \(\wedge \forall 0 < i < n, \pi^{(S)}(i) \notin G\), +\(\infty\) otherwise. 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. The worst case distance (pessimistic view) of a state \( s \) from the goal region \( G \) is \( J_{\text{strong}}(S, G, s) = \sup \{ J(S, \pi, G) \mid \pi \in \text{Path}(s) \} \). The best case distance (optimistic view) of a state \( s \) from the goal region \( G \) is \( J_{\text{weak}}(S, G, s) = \inf \{ J(S, \pi, G) \mid \pi \in \text{Path}(s) \} \).

**Definition 3.** A strong [weak] solution to a control problem \( \mathcal{P} = (S, I, G) \) is a controller \( K \) for \( S \), such that \( I \subseteq \text{Dom}(K) \) and for all \( s \in \text{Dom}(K) \), \( J_{\text{strong}}(S^{(K)}, G, s) \) [\( J_{\text{weak}}(S^{(K)}, G, s) \)] is finite.

**Example 3.** Let \( S_0 [S_1] \) be the LTS which transition relation consists of the continuous [all] arrows in Fig. 2 (left). Let \( I = \{-1, 0, 1\} \) and \( G = \{0\} \). Then, \( K(s, \hat{u}) \equiv [s \neq 0 \Rightarrow \hat{u} = 0] \) is a strong solution to the control problem \((S_0, \hat{I}, G)\) and a weak solution to \((S_1, \hat{I}, G)\).

**Remark 1.** Note that if \( K \) is a strong solution to \((S, I, G)\) and \( G \subseteq I \) (as it is usually the case in control problems) then all paths starting from \( S_0 (\subseteq I) \) will touch \( G \) infinitely often (stability).

A DTLHS control problem is a triple \((\mathcal{H}, I, G)\) where \( \mathcal{H} \) is a DTLHS and \((\text{LTS}(\mathcal{H}), I, G)\) is an LTS control problem. For DTLHSs we restrict ourselves to control problems where \( I \) and \( G \) can be represented as conjunctive predicates. From [17] it is easy to show that DTLHS control problems are undecidable [23]. For DTLHS control problems usually robust controllers are desired. That is, controllers that, notwithstanding nondeterminism in the plant (e.g. due to parameter variations), drive the plant state to the goal region. For this reason, and to counteract the nondeterminism stemming from the quantization process, we focus on strong solutions. Furthermore, to accommodate quantization errors, always present in software based controllers, it is useful to relax the notion of control solution by tolerating an (arbitrarily small) error \( \varepsilon \) on the continuous variables. This leads to the definition of \( \varepsilon \)-solution. Let \( \varepsilon \) be a nonnegative real number, \( W^r = \prod_{i=1}^m W_i^r \subseteq \mathcal{X}^r \), \( W^d = \prod_{i=1}^m W_i^d \subseteq \mathcal{X}^d \) and \( W = W^r \times W^d \subseteq \mathcal{X}^r \times \mathcal{X}^d \). The \( \varepsilon \)-relaxation of \( W \) is the set (ball of radius \( \varepsilon \)) \( B_{\varepsilon}(W) = \{(z_1, \ldots, z_n) \mid (q_1, \ldots, q_m) \in W^d \) and \( \forall i \in \{1, \ldots, n\} \exists x_i \in W_i^r \) s.t. \( |z_i - x_i| \leq \varepsilon \} \).

**Definition 4.** Let \((\mathcal{H}, I, G)\) be a DTLHS control problem and \( \varepsilon \) be a nonnegative real number. An \( \varepsilon \)-solution to \((\mathcal{H}, I, G)\) is a strong solution to the LTS control problem \((\text{LTS}(\mathcal{H}), I, B_{\varepsilon}(G))\).

**Example 4.** Let \( \mathcal{P} = (\mathcal{H}, I, G) \) as in Ex. 1, \( I = \mathcal{X} \) and \( G = \{0\} \) (represented by conjunctive predicate \( x = 0 \)). Control problem \( \mathcal{P} \) has no solution (because of the Zeno phenomenon), but for all \( \varepsilon > 0 \) it has the \( \varepsilon \)-solution \( K \) s.t. \( \forall x \in I \), \( K(x, 0) = 1 \).

**Example 5.** The typical goal of a controller for the buck DC-DC converter in Ex. 2 is keeping the output voltage \( v_o \) close enough to a given reference value \( V_{ref} \). This leads to the control problem \( \mathcal{P} = (\mathcal{H}, I, G) \) where: \( \mathcal{H} \) is defined in Ex. 2, \( I = (|i_L| \leq 2) \land (0 \leq v_o \leq 6.5) \), \( G = (|v_o - V_{ref}| \leq \theta) \land (|i_L| \leq 2) \) and \( \theta = 0.01 \) is the desired converter precision.

In order to define quantized feedback control problems for DTLHSs (Def. 6) we introduce quantizations (Def. 5). Let \( x \) be a real valued variable ranging on a bounded interval of reals \( \mathcal{X} = [a_x, b_x] \). A quantization for \( x \) is a function \( \gamma \)
from \( D_x \) to a bounded interval of integers \( \gamma(D_x) = [a_x, b_x] \). For ease of notation we extend quantizations to integer variables ranging on a bounded interval of integers by stipulating that the only quantization \( \gamma \) for such variables is the identity function (i.e. \( \gamma(x) = x \)). The width \( \|\gamma^{-1}(v)\| \) of \( v \in \gamma(D_x) \) in \( \gamma \) is defined as follows: \( \|\gamma^{-1}(v)\| = \sup \{ |w - z| \mid w, z \in D_x \land \gamma(w) = \gamma(z) = v \} \). The quantization step \( \|\gamma\| \) is defined as follows: \( \|\gamma\| = \max \{ \|\gamma^{-1}(v)\| \mid v \in \gamma(D_x) \} \).

**Definition 5.** Let \( \mathcal{H} = (X, U, Y, N) \) be a DTLHS. A quantization \( \Gamma \) for \( \mathcal{H} \) is a set of maps \( \Gamma = \{ \gamma_w \mid \gamma_w \text{ is a quantization for } w \in X \cup U \} \). Let \( W = [w_1, \ldots, w_k] \subseteq X \cup U \) and \( v = [v_1, \ldots, v_k] \in D_W \). We write \( \Gamma(v) \) for the tuple \( [\gamma_{w_1}(v_1), \ldots, \gamma_{w_k}(v_k)] \) and \( \Gamma(D_W) \) for the set of tuples \( \{\Gamma(v) \mid v \in D_W \} \). Finally, the quantization step \( \|\Gamma\| \) for \( \Gamma \) is defined as: \( \|\Gamma\| = \max \{ \|\gamma\| \mid \gamma \in \Gamma \} \).

A control problem admits a quantized solution if control decisions can be made by just looking at quantized values. This enables a software implementation for a controller.

**Definition 6.** Let \( \mathcal{H} = (X, U, Y, N) \) be a DTLHS, \( \Gamma \) be a quantization for \( \mathcal{H} \) and \( \mathcal{P} = (\mathcal{H}, I, \mathcal{G}) \) be a control problem. A \( \Gamma \)-Quantized Feedback Control (QFC) solution to \( \mathcal{P} \) is a \( \|\Gamma\| \)-solution \( K(x, u) \) to \( \mathcal{P} \) such that \( K(x, u) = \hat{K}(\Gamma(x), \Gamma(u)) \) where \( \hat{K} : \Gamma(D_X) \times \Gamma(D_U) \to \mathbb{B} \).

![Fig. 2. \( \Gamma \) control abstraction for DTLHSs in Exs. 3 and 8 (left) and Ex. 9 (right)](image)

**Example 6.** Let \( \mathcal{P} \) be as in Ex. 4, \( \Gamma(x) = \text{round}(x/2) \) (where \( \text{round}(x) = \lfloor x \rfloor + \lfloor 2(x - \lfloor x \rfloor) \rfloor \) is the usual rounding function) and \( \hat{K} \) as in Ex. 3. Then, \( \|\Gamma\| = 2 \) and \( K(x, u) = \hat{K}(\Gamma(x), \Gamma(u)) \) is a \( \Gamma \) QFC solution to \( \mathcal{P} \).

## 5 Control Abstraction

The AD process maps intervals of state values into discrete state values. As a result the control software sees the controlled system (plant) as a nondeterministic finite automaton. Of course we want our control software to work notwithstanding such a nondeterminism (strong solution, Def. 3). To this end we should try to limit such a nondeterminism as much as possible. This leads to the notion of control abstraction (Def. 8), the main focus of this section.

Since QFC (Def. 6) 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 safe action (Def. 7). Intuitively, an action is safe in a state if it never drives the system outside of its state bounds.

**Definition 7.** Let \( \mathcal{H} = (X, U, Y, N) \) be a DTLHS and \( \Gamma \) be a quantization.

1. We say that action \( u \in D_U \) is safe for \( s \in D_X \) (in \( \mathcal{H} \)) if for all \( s' \), \( \exists y \in D_Y \) \( N(s, u, y, s') \) implies \( s' \in D_X \).
2. We say that action \( \hat{u} \in \Gamma(D_U) \) is \( \Gamma \)-safe in state \( \hat{s} \in \Gamma(D_X) \) if for all \( s \in \Gamma^{-1}(\hat{s}) \), \( u \in \Gamma^{-1}(\hat{u}) \), \( u \) is safe for \( s \) in \( H \).

Note that, in general, not all actions \( u \in D_U \) are safe in \( H \) since Def. 1 only asks \( N \) to be a conjunctive predicate.

Example 7. Let \( H \) be as in Ex. 1. Then action \( u = 1 \) is not safe in state \( s = 2 \) since we have \( N(2,1,3) \), and \( s' = 3 \) is outside \( H \) state bounds.

A control abstraction (Def. 8) is a finite state automaton modelling how a DTLHS is seen from the control software because of AD conversion.

Definition 8. Let \( H = (X, U, Y, N) \) be a DTLHS and \( \Gamma \) be a quantization for \( H \). We say that the LTS \( \hat{H} = (\hat{\Gamma}(D_X), \hat{\Gamma}(D_U), \hat{N}) \) is a \( \Gamma \) control abstraction of \( H \) if its transition relation \( \hat{N} \) satisfies the following conditions.

1. Each abstract transition stems from a concrete transition. Formally: for all \( \hat{s}, \hat{s}' \in \hat{\Gamma}(D_X), \hat{u} \in \hat{\Gamma}(D_U), \) if \( \hat{N}(\hat{s}, \hat{u}, \hat{s}') \) then there exist \( s \in \Gamma^{-1}(\hat{s}), u \in \Gamma^{-1}(\hat{u}), s' \in \Gamma^{-1}(\hat{s}') \), \( y \in \tilde{D}_Y \) s.t. \( N(s, u, y, s') \).

2. If an abstract action is safe then all its possible concrete effects (besides self-loops) are faithfully represented in the abstract system. Formally: for all \( \hat{s} \in \hat{\Gamma}(D_X), \hat{u} \) \( \Gamma \)-safe in \( \hat{s} \), \( s \in \Gamma^{-1}(\hat{s}), u \in \Gamma^{-1}(\hat{u}) \), \( s' \in D_X \), if \( \exists y \in \tilde{D}_Y \) \( N(s, u, y, s') \) then \( \hat{N}(\hat{s}, \hat{u}, \hat{s}') \).

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 \( \hat{s} \in \hat{\Gamma}(D_X), \hat{u} \in \hat{\Gamma}(D_U), \) if \( \forall k \exists x(0), \ldots, x(k+1) \in \Gamma^{-1}(\hat{s}) \) \( \exists u(0), \ldots, u(k) \in \Gamma^{-1}(\hat{u}) \exists y(0), \ldots, y(k) \in \tilde{D}_Y \left[ \bigwedge_{t=0}^{k} N(x(t), u(t), y(t), x(t + 1)) \right] \) then \( \hat{N}(\hat{s}, \hat{u}, \hat{s}) \).

We say that \( \hat{H} \) is a control abstraction of \( H \) if \( \hat{H} \) is a \( \Gamma \) control abstraction of \( H \) for some quantization \( \Gamma \). Finally, we denote with \( A_{\Gamma}(H) \) the set of all \( \Gamma \) control abstractions on \( H \).

Note that any abstraction (e.g. see [2]) is also a control abstraction. However, the converse is false since some concrete transition (e.g. a self loop or an unsafe action) may have no abstract image. Let \( S_1 = (S, A, T_1) \) and \( S_2 = (S, A, T_2) \) be LTSs. We say that \( S_1 \) refines \( S_2 \) (notation \( S_1 \sqsubseteq S_2 \)) iff for each \( s, s' \in S, a \in A, T_1(s, a, s') \) implies \( T_2(s, a, s') \). The binary relation \( \sqsubseteq \) is a partial order.

Moreover, the poset \( (A_{\Gamma}(H), \subseteq) \) of an abstraction \( \hat{H} \) is a lattice. Furthermore, since \( A_{\Gamma}(H) \) is a finite set, the poset \( (A_{\Gamma}(H), \subseteq) \) has unique maximum and unique minimum elements.

Example 8. Let \( H \) be as in Ex. 1 and \( \Gamma \) be as in Ex. 6. Each \( \Gamma \) control abstraction of \( H \) has the form \( \hat{H} = (\{-1,0,1\},\{0,1\},\hat{N}) \), where the set of transitions in \( \hat{N} \) is any subset, containing all continuous arrows, of the set of transitions of the automaton depicted in Fig. 2 (left). In particular, a control abstraction may omit some self loops (namely, those with dotted arrows in Fig. 2). Transitions \( \hat{N}(0,0,0) \) and \( \hat{N}(0,1,0) \) must belong to all \( \Gamma \) control abstractions, because of condition 3 in Def. 8. In fact all paths starting in 0 will remain in 0 forever. The transition relation defined in Fig. 2 (left) by continuous arrows is the minimum \( \Gamma \) control abstraction \( \hat{H}_{\text{min}} \) of \( \hat{H} \) whereas the transition relation defined by all arrows is the maximum \( \Gamma \) control abstraction \( \hat{H}_{\text{max}} \) of \( \hat{H} \). Note that there is no controller (strongly) driving all states of \( \hat{H}_{\text{max}} \) to state 0. In fact, because of self-loops, action 0 from state 1 may lead to state 0 as well as to state 1 (self-loop). On the other hand the controller \( \hat{K} \) enabling only action 0 in any
state will (weakly) drive all states of $\hat{H}_{\text{max}}$ to 0 since each state in $\hat{H}_{\text{max}}$ has at least a 0-labelled transition leading to state 0. Controller $\hat{K}$ will also (strongly and thus weakly) drive all states of $\hat{H}_{\text{min}}$ (including 0) to state 0.

**Remark 2.** Example 8 suggests that we should focus on minimum control abstractions in order to increase our chances of finding a strong controller. Correctness of such an intuition will be shown in Theor. 1. As for computing the minimum control abstraction we note that this entails deciding if a given self-loop can be eliminated according to condition 3 in Def. 8. Unfortunately it is easy to show that such a problem comes down to solve a reachability problem on linear hybrid systems, that, by [17], is undecidable. Thus, self-loop eliminability is undecidable too in our context. As a result, in general, we cannot hope to compute exactly the minimum control abstraction.

## 6 Synthesis of Quantized Feedback Control Software

We outline our synthesis algorithm QKS (Quantized feedback Kontrol Synthesis) and give its properties (Theor. 1). Details are in [23]. QKS takes as input a tuple $(\Gamma, \mathcal{H}, I, G)$, where: $\mathcal{H} = (X, U, Y, N)$ is a DTLHS, $\Gamma$ is a quantization for $\mathcal{H}$ and $(\mathcal{H}, I, G)$ is a control problem. QKS returns a tuple $(\mu, \hat{D}, \hat{K})$, where: $\mu \in \{\text{Sol}, \text{NoSol}, \text{Unk}\}$, $K(x, u) = \bar{K}(\Gamma(x), \Gamma(u))$ is a $\Gamma$ QFC solution for $\mathcal{H}$ (Def. 6), $\hat{D} = \text{Dom}(\hat{K})$ and $D = \Gamma^{-1}(\hat{D}) = \text{Dom}(K)$ is $K$ controllable region.

We compute QKS output as follows. As a first step we compute a $\Gamma$ control abstraction $\mathcal{Q} = (\Gamma(P_X), \Gamma(P_U), N)$ of $\mathcal{H}$ as close as we can (see Remark 2) to the minimum one. Sect. 6.1 (function \texttt{minCtrlAbs} in Alg. 1) outlines how $\mathcal{Q}$ can be computed. Let $\hat{I} = \Gamma(I), \hat{G} = \Gamma(G)$ and $\hat{K}$ be the most general optimal (mgo) strong solution to the (LTS) control problem $(\mathcal{Q}, \emptyset, \hat{G})$. Intuitively, the mgo strong solution $\hat{K}$ to a control problem $(\mathcal{Q}, \emptyset, \hat{G})$ is the unique strong solution that, disallowing as few actions as possible, drives as many states as possible to a state in $\hat{G}$ along a shortest path. We compute (the OBDD representation for) $\hat{K}$ by implementing a suitable variant of the algorithm in [12]. Finally, we define: $K(x, u) = \bar{K}(\Gamma(x), \Gamma(u)), \hat{D} = \text{Dom}(\hat{K})$, and $D = \Gamma^{-1}(\hat{D}) = \text{Dom}(K)$.

If $\hat{I} \subseteq \hat{D}$ then QKS returns $\mu = \text{Sol}$. Note that in such a case, from the construction in [12], $\hat{K}$ is time optimal for the control problem $(\mathcal{Q}, I, \hat{G})$, thus $K$ will typically move along a shortest path to $\hat{G}$ (i.e., $\hat{K}$ is near time-optimal). If $\hat{I} \not\subseteq \hat{D}$ then we compute the maximum $\Gamma$ control abstraction $\mathcal{W}$ of $\mathcal{H}$ and use the algorithm in [29] to check if there exists a weak solution to $(\mathcal{W}, I, \hat{G})$. If that is the case QKS returns $\mu = \text{Unk}$, otherwise QKS returns $\mu = \text{NoSol}$. Note that the maximum control abstraction may contain also (possibly) unsafe transitions (condition 2 of Def. 8). Thus a weak solution for $\mathcal{W}$ may exist even when no weak solution for $\mathcal{Q}$ exists. Using the above notations Theor. 1 summarizes the main properties of QKS.

**Theorem 1.** Let $\mathcal{H}$ be a DTLHS, $\Gamma$ be a quantization and $(\mathcal{H}, I, G)$ be a control problem. Then QKS($\Gamma, \mathcal{H}, I, G$) returns a triple $(\mu, \hat{D}, \hat{K})$ s.t.: $\mu \in \{\text{Sol}, \text{NoSol}, \text{Unk}\}$, $\hat{D} = \text{Dom}(\hat{K}), D = \Gamma^{-1}(\hat{D})$ and $K = \bar{K}(\Gamma(x), \Gamma(u))$ is a $\Gamma$ QFC solution to the control problem $(\mathcal{H}, D, G)$. Furthermore, the following holds.
1. If \( \mu = \text{Sol} \) then \( I \subseteq D \) and \( K \) is a \( \Gamma \) QFC solution to the control problem \((\mathcal{H}, I, G)\).

2. If \( \mu = \text{NoSol} \) then there is no \( \Gamma \) QFC solution to the control problem \((\mathcal{H}, I, G)\).

3. If \( \mu = \text{Unk} \) then QKS is inconclusive, that is \((\mathcal{H}, I, G)\) may or may not have a \( \Gamma \) QFC solution.

Note that the AD conversion hardware is modelled by \( \Gamma \) and that from the OBDDs for \( \hat{K} \) above we get a C program (Section 6.2). Thus \( \hat{K} \) as described above defines indeed the control software we are looking for. Finally, note that case 3 in Theor. 1 stems from undecidability of the QFC problem [17].

Example 9. Let \( \mathcal{P} = (\mathcal{H}, I, G) \) be as in Ex. 4 and \( I \) be as in Ex. 8. For all \( \Gamma \) control abstractions \( \mathcal{H} \) (and thus for the minimum one shown in Ex. 8) not containing the self loops \( \hat{N}(-1, 0, -1) \) and \( \hat{N}(1, 0, 1) \), \( \hat{K} \) as in Ex. 3 is the mg strong solution to \((\hat{H}, \varnothing, \Gamma(G))\). Thus, \( K(s, u) \) as in Ex. 6 is a \( \Gamma \) QFC solution to \( \mathcal{P} \). Weak solutions to \((\hat{H}, \Gamma(I), \Gamma(G))\) exist for all \( \Gamma \) control abstractions \( \mathcal{H} \).

Note that existence of a \( \Gamma \) QFC solution to a control problem depends on \( \Gamma \). Let us consider the quantization \( \Gamma'(x) = \lfloor x/2 \rfloor \) for \( \mathcal{H} \). Then the maximum \( \Gamma' \) control abstraction of \( \mathcal{H} \) is \( \mathcal{L} = \{\{-2, -1, 0, 1\}, \{0, 1\}, \hat{N}\} \), where the transition \( \hat{N} \) is depicted in Fig. 2 (right). Clearly \((\mathcal{L}, \Gamma'(I), \Gamma'(G))\) has no weak solution since there is no path to the goal \( \Gamma'(G) = \{0\} \) from any of the states \(-2, -1\). Thus \( \mathcal{P} \) has no \( \Gamma'' \) QFC solution.

6.1 Computing Control Abstractions

Function \( \text{minCtrAbs} \) in Alg. 1 computes a close to minimum \( \Gamma \) control abstraction (Def. 8) \( \hat{Q} = (\Gamma(D_X), \Gamma(D_U), N) \) of \( \mathcal{H} = (X, U, Y, N) \) as well as \( I = \Gamma(I) \) and \( \hat{G} = \Gamma(G) \).

Line 6 initializes (the OBDDs for) \( \hat{N}, \hat{I}, \hat{G} \) to \( \varnothing \) (i.e. the boolean function identically 0). Line 2 loops through all \( |\Gamma(D_X)| \) states \( \hat{s} \) of \( \mathcal{H} \). Line 3 [line 4] add state \( \hat{s} \) to \( \hat{I} \) [\( \hat{G} \)] if \( \hat{s} \) is the image of a concrete state in \( I \) [\( G \)]. Line 5 loops through all \( |\Gamma(D_U)| \) actions \( \hat{u} \) of \( \mathcal{H} \). Line 13 checks if action \( \hat{u} \) is \( \Gamma \)-safe in \( \hat{s} \) (see Def. 7.2 and Def. 8.2). Function \( \text{SelfLoop} \) in line 7 returns 0 when, accordingly to Def. 8.3 a self-loop need not to be in \( \hat{N} \). An exact check is undecidable (Remark 2), however our gradient based \( \text{SelfLoop} \) function typically turns off (Tab. 1 in Sect. 7) to be a quite tight overapproximation of the sets of (strictly needed) self-loops.

We compute \( \text{SelfLoop}(\hat{s}, \hat{u}) \) as follows. For each real valued state component \( x_i \), let \( w_i \| W_i \| = \{\min, \max, x'_i - x_i, N(X, U, Y, X') \wedge \Gamma'(X) = \hat{s} \wedge \Gamma(U) = \hat{u}\} \). If for some \( \hat{u} [w_i \neq 0 \wedge W_i \neq 0 \wedge (w_i \text{ and } W_i \text{ have the same sign})] \) then \( \text{SelfLoop} \) returns 0 (since any long enough sequence of concrete actions in \( \Gamma^{-1}(\hat{u}) \) will drive state component \( x_i \), outside of \( \Gamma^{-1}(\hat{s}) \)), otherwise \( \text{SelfLoop} \) returns 1. Lines 9, 10, 11 compute a quite tight overapproximation (\( \text{OverImg} \)) of the set of states reachable in one step from \( \hat{s} \). Line 12 loops on all \( \text{OverImg} \) abstract next states \( \hat{s}' \) that may be reachable with the abstract outgoing transition \( (\hat{s}, \hat{u}) \) under consideration. Line 13 checks if there exists a concrete transition realizing the abstract transition \( (\hat{s}, \hat{u}, \hat{s}') \) when \( \hat{s} \neq \hat{s}' \) (no self-loop) and if so adds the
abstract transition \((\bar{s}, \bar{u}, \bar{s}')\) to \(\tilde{N}\) (line 14). Finally, line 15 returns the (transition relation for) the control abstraction along with \(\tilde{I}\) and \(\tilde{G}\).

**Remark 3.** From the loops in lines 2, 5, 12 we see that the worst case runtime for Alg. 1 is \(O(|\Gamma(D_X)|^2|\Gamma(D_U)|))\). However, thanks to the heuristic in lines 9–11, Alg. 1 typical runtime is about \(O(|\Gamma(D_X)||\Gamma(D_U)|)\) as confirmed by our experimental results (Sect. 7, Fig. 3(b)).

**Remark 4.** Alg. 1 is explicit in the (abstract) states and actions of \(\tilde{\mathcal{H}}\) and symbolic with respect to the auxiliary variables (modes) in the transition relation \(\mathcal{N}\) 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.

**Algorithm 1 Building control abstractions**

**Input:** A quantization \(\Gamma\); a DTLHS \(\mathcal{H} = (X, U, Y, N)\), a control problem \((\mathcal{H}, \tilde{I}, \tilde{G})\).

**Function** \(\text{minCtrlAbs}(\Gamma, \mathcal{H}, \tilde{I}, \tilde{G})\):

1. \(\tilde{N} \leftarrow \emptyset, \tilde{I} \leftarrow \emptyset, \tilde{G} \leftarrow \emptyset\), let \(X = [x_1, \ldots, x_n]\), \(X' = [x_1', \ldots, x_n']\)
2. for all \(\hat{s} \in \Gamma(D_X)\) do
3.   if (MLP (min, 0, \(I(X) \wedge \Gamma(X) = \hat{s}\)) is feasible) then \(\tilde{I} \leftarrow \tilde{I} \cup \{\hat{s}\}\)
4.   if (MLP (min, 0, \(G(X) \wedge \Gamma(X) = \hat{s}\)) is feasible) then \(\tilde{G} \leftarrow \tilde{G} \cup \{\hat{s}\}\)
5. for all \(\hat{u} \in \Gamma(D_U)\) do
6.   if (MLP (min, 0, \(N(X, U, Y, X') \wedge \Gamma(X) = \hat{s} \wedge \Gamma(U) = \hat{u} \wedge X' \notin D_X\)) is feasible) then continue
7.   if SelfLoop(\(\hat{s}, \hat{u}\)) then \(\tilde{N} \leftarrow \tilde{N} \cup \{\hat{s}, \hat{u}, \hat{s}\}\)
8.   for all \(i = 1, \ldots, n\) do
9.     \(m_i \leftarrow x_i^*, \text{ where } X^* = [x_1^*, \ldots, x_n^*]\) is a solution to the MILP (min, \(x_i^*\), \(N(X, U, Y, X') \wedge \Gamma(X) = \hat{s} \wedge \Gamma(U) = \hat{u}\))
10. \(M_i \leftarrow x_i^*\), \(\text{ where } X'^* = [x_1^*, \ldots, x_n^*]\) is a solution to the MILP (max, \(x_i^*\), \(N(X, U, Y, X') \wedge \Gamma(X) = \hat{s} \wedge \Gamma(U) = \hat{u}\))
11. let \(\text{Over}_{\text{lng}}(\hat{s}, \hat{u}) = \prod_{i=1}^{n} [\gamma_{e_i}(m_i), \gamma_{e_i}(M_i)]\)
12. for all \(\hat{s}' \in \text{Over}_{\text{lng}}(\hat{s}, \hat{u})\) do
13.   if \(\hat{s} \neq \hat{s}' \wedge (\text{MLP (min, 0, } N(X, U, Y, X') \wedge \Gamma(X) = \hat{s} \wedge \Gamma(U) = \hat{u} \wedge \Gamma(X') = \hat{s}')\) is feasible) then
14.     \(\tilde{N} \leftarrow \tilde{N} \cup \{\hat{s}, \hat{u}, \hat{s}'\}\)
15. return \((\tilde{N}, \tilde{I}, \tilde{G})\)

### 6.2 Control Software With a Guaranteed WCET

From controller \(\hat{K}\) computed by QKS (see Sect. 6) we generate our correct-by-construction control software (obdd2c(\(\hat{K}\))). This is done (function obdd2c) by translating the OBDD representing \(\hat{K}\) into C code along the lines of [29]. From such a construction we can easily compute the Worst Case Execution Time (WCET) for our controller. We have: \(WCET = nrT_B\), where \(r [n]\) is the number of bits used to represent plant actions [states] and \(T_B\) is the time needed to execute the C instructions modelling the if-then-else semantics of OBDD nodes as well as edge complementation (since we use the CUDD package).

Let \(T\) be the chosen sampling time. Then it must be \(WCET \leq T\). That is, \(nrT_B \leq T\). This equation allows us to know, before hand, the realizability (e.g. with respect to schedulability constraints) of the (to be designed) control software. For example, let \(T_B = 10^{-7}\text{sec}s\), \(n = 10\) and \(r = 1\). Then, the for the system sampling time we have: \(T \geq 10^{-6} = WCET\).
7 Experimental Results

We implemented QKS (Sect. 6) in C, using GLPK to solve MILP problems and the CUDD package for OBDD based computations.

Our experiments aim at evaluating effectiveness of: control abstraction (\(\hat{Q}\), Sect. 6.1) generation, synthesis of OBDD representation of control law (\(\hat{K}\), Sect. 6), control software (obdd2c(\(\hat{K}\)), Sect. 6.2) size and guaranteed operational ranges (i.e. controllable region). Note that control software reaction time (WCET) is known a priori from Sect. 6.2 and its robustness to parameter variations in the controlled system (\(\mathcal{H}\)) as well as enforcement of safety bounds on state variables are an input to our synthesis algorithm (e.g. see Ex. 1, 2).

We present experimental results obtained by using QKS on the buck DC-DC converter described in Ex. 2. We denote with \(\mathcal{H}\) the DTLHS modeling such a converter. We set the parameters of \(\mathcal{H}\) as follows: \(T = 10^{-6}\) secs, \(L = 2 \cdot 10^{-4}\) H, \(r_L = 0.1\) Ω, \(r_C = 0.1\) Ω, \(R = 5 \pm 25\%\) Ω, \(C = 5 \cdot 10^{-5}\) F, \(V_i = 15 \pm 25\%\) V and require our controller to be robust to foreseen variations (25%) in the load (\(R\)) and in the power supply (\(V_i\)).

The model in Ex. 2 already accounts for variations in the power supply. Variations in the load \(R\) can be taken into account along the same lines, however much more work is needed (along the lines of [15]) since \(\mathcal{H}\) dynamics is not linear in \(R\). This adds 11 auxiliary boolean variables to the model in Ex. 2. Details are in [23]. For converters, safety (as well as physical) considerations set requirements on admissible values for state variables. We set: \(D_{\nu_0} = [-4, 4]\), \(D_{\nu_p} = [-1, 7]\). Note that robustness requires that, notwithstanding nondeterministic variations (within the given tolerances) for power supply and load, the synthesized controller always keeps state variables within their admissible regions. We use the following bounds for auxiliary variables: \(D_{\nu_0} = D_{\nu_p} = [-10^{3}, 10^{3}]\) and \(D_{\nu_1} = D_{\nu_2} = [-10^{7}, 10^{7}]\). The initial region \(I\) and goal region \(G\) are as in Ex. 5. Finally, 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.

Table 1. Buck DC-DC converter (Sect. 3): control abstraction and controller synthesis results. Experiments run on an Intel 3.0 GHz Dual Core Linux PC with 4 GB of RAM.

<table>
<thead>
<tr>
<th>(b)</th>
<th>CPU</th>
<th>Arcs</th>
<th>MaxLoops</th>
<th>LoopFac</th>
<th>Controller Abstraction</th>
<th>(\nu_0)</th>
<th>OBDD</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2.50e+03</td>
<td>1.54e+06</td>
<td>2.54e+04</td>
<td>0.00323</td>
<td>6.00e+00</td>
<td>1.07e+02</td>
<td>2.50e+03</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.33e+04</td>
<td>7.72e+05</td>
<td>1.58e+04</td>
<td>0.00440</td>
<td>1.00e+02</td>
<td>1.24e+03</td>
<td>1.14e+04</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6.34e+04</td>
<td>5.16e+06</td>
<td>2.09e+04</td>
<td>0.00781</td>
<td>1.00e+02</td>
<td>2.75e+03</td>
<td>7.01e+04</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6.86e+05</td>
<td>4.24e+08</td>
<td>2.29e+04</td>
<td>0.01417</td>
<td>5.00e+03</td>
<td>7.00e+03</td>
<td>4.13e+05</td>
<td></td>
</tr>
</tbody>
</table>

We use a uniform quantization dividing the domain of each state variable \((\nu_0, \nu_p)\) into \(2^b\) equal intervals, where \(b\) is the number of bits used by AD conversion. We call the resulting quantization \(\Gamma_b\). The quantization step is \(\|\Gamma_b\| = 2^{3-b}\).

For each value of interest for \(b\), following Sect. 6, we compute: (1) a (close to minimum) \(\Gamma_b\) control abstraction \(\hat{\nu}_b\) for \(\mathcal{H}\), (2) the mgo strong solution \(\hat{\nu}_b\) for \(\hat{P}_b = (\hat{\nu}, \emptyset, \Gamma_b(G))\), (3) \(\hat{\nu}_b\) controllable region \(\hat{D}_b = \text{Dom}(\hat{\nu}_b)\), (4) a \(\Gamma_b\) QFC
solution $K^b(s, u) = \hat{K}^b(\Gamma_b(s), \Gamma_b(u))$ to the control problem $P^b = (H, \Gamma_b^{-1}(\hat{D}^b), G)$. Note that, since we have two quantized variables ($i_L, v_Q$) each one with $b$ bits, the number of states in the control abstraction is exactly $2^{2b}$.

Tab. 1 shows our experimental results. Columns in Tab. 1 have the following meaning. Column $b$ shows the number of AD bits. Columns labelled Control Abstraction show performances for Alg. 1. Column CPU shows Alg. 1 time (in secs) to compute \(\hat{H}^b\). Column Areas shows the number of transitions in \(\hat{H}^b\). In order to assess effectiveness of function SelfLoop (Sect. 6.1) column MaxLoops shows the number of loops in the maximum \(\Gamma_b\) control abstraction for \(H\), while column \(\text{LoopFrac}\) shows the fraction of such loops in \(\hat{H}^b\). Columns labelled Controller Synthesis show the computation time in secs (CPU) for the generation of \(\hat{K}^b\), and the size of its OBDD representation (OBDD). The latter is also the size (number of lines) of the C code for our synthesized implementation of \(\hat{K}^b\) (obdd2c(\(\hat{K}^b\))). Finally, column Total shows the total computation time in secs (CPU) for the whole process (i.e., control abstraction plus controller source code generation). All computations were completed using no more than 200MB. As for the value of \(\mu\) (see Thero. 1), we have that \(\mu = \text{UNK}\) for $b = 8$, and \(\mu = \text{SOL}\) in all other cases.

From Tab. 1 we see that computing control abstractions (i.e. Alg. 1) is the most expensive operation in QKS (see Sect. 6) and that thanks to function SelfLoop \(\hat{K}^b\) contains no more than 2% of the loops in the maximum \(\Gamma_b\) control abstraction for \(H\).

For each MILP problem in Alg. 1, Fig. 3(b) shows (as a function of $b$) the number of MILP instances solved while Fig. 3(a) shows (as a function of $b$) the average CPU time (in seconds) spent solving a single MILP problem instance. CPU time standard deviation is always less than 0.003. The correspondence between the curves in Figs. 3(b), 3(a) and Alg. 1 is the following. MILP1 refers to line 3 (and represents also the data for the twin MILP in line 4). MILP2 refers to MILP problems in function SelfLoop (line 7). MILP3 refers to line 9 (and represents also the data for the twin MILP in line 10). MILP4 refers to line 13 and MILP5 refers to line 6.

From Fig. 3(a) we see that the average time spent solving each MILP instance is small. The lower [upper] bound to the number of times MILP4 (i.e. the most called MILP in Alg. 1) is called (#MILP4) is \(|\Gamma(D_X)||\Gamma(D_U)| = 2^{2b+1} \quad |\Gamma(D_X)|^2|\Gamma(D_U)| = 2^{4b+1}\) (see Remark 3). From Fig. 3(b) we see that #MILP4 is quite close to \(|\Gamma(D_X)||\Gamma(D_U)| = 2^{2b+1}\). This shows effectiveness of our heuristic to tightly overapproximate \text{Over Img} (lines 9-11 of Alg. 1).

One of the most important features of our approach is that it returns the guaranteed operational range (precondition) of the synthesized software (Thore. 1). This is the controllable region $D$ returned by QKS in Sect. 6. Fig. 3(c) shows the controllable region $D$ for $K^{10}$ along with some trajectories (with time increasing counterclockwise) for the closed loop system. Since for $b = 10$ we have $\mu = \text{SOL}$, we have that $I \subseteq D$ (see also Fig. 3(c)). Thus we know (on a formal ground) that 10 bit AD ($\|I_{10}\| = 2^{-7}$) conversion suffices for our purposes. The controllable region for $K^{11}$ turns out to be only slightly larger than the one for $K^{10}$.\(\)
8 Conclusion

We presented an effective algorithm that given a DTLHS $\mathcal{H}$ and a quantization schema returns a correct-by-construction robust control software $K$ for $\mathcal{H}$ along with the controllable region $R$ for $K$. Furthermore, our control software has a WCET linear in the number of bits of the quantization schema. We have implemented our algorithm and shown feasibility of our approach by presenting experimental results on using it to synthesize C controllers for the buck DC-DC converter. Our approach is explicit in the quantized state variables and symbolic in the system modes. Accordingly, it works well with systems with a small number of (continuous) state variables and possibly many modes. Many hybrid systems fall in this category.

Future research may investigate fully symbolic approaches, e.g., based on Fourier-Motzkin (FM) variable elimination, to compute control abstractions. Since FM tools typically work on rational numbers this would also have the effect of avoiding possible numerical errors of MILP solvers [24].

Acknowledgments We are grateful to our anonymous referees for their helpful comments. Our work has been partially supported by: MIUR project DM24283 (TRAMP) and by the EC FP7 project GA218815 (ULISSE).

References