Quantized Feedback Control Software Synthesis from System Level Formal Specifications

Federico Mari, Igor Melatti, Ivano Salvo, Enrico Tronci

Department of Computer Science
Sapienza University of Rome
via Salaria 113, 00198 Rome
email: {mari,melatti,salvo,tronci}@di.uniroma1.it

September 12, 2011

Abstract

Many Embedded Systems are indeed Software Based Control Systems (SBCSs), 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 SBCS 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, DTLHS) of the controlled system (plant), implementation specifications (that is, number of bits in the Analog-to-Digital, AD, conversion) and System Level Formal Specifications (that is, safety and liveness requirements for the closed loop system) returns correct-by-construction control software that has a Worst Case Execution Time (WCET) linear in the number of AD bits and meets the given specifications. We show feasibility of our approach by presenting experimental results on using it to synthesize control software for a buck DC-DC converter, a widely used mixed-mode analog circuit.
1. Every $T$ seconds (sampling time) do
2. Read AD conversion $\hat{x}$ of plant sensor outputs $x$
3. If ($\hat{x}$ is not in the Controllable Region)
4. Then // Exception (Fault Detected):
5. Start Fault Isolation and Recovery (FDIR)
6. Else // Nominal case:
7. Compute (Control Law) command $\hat{u}$ from $\hat{x}$
8. Send DA conversion $u$ of $\hat{u}$ to plant actuators

Figure 1: A typical control loop skeleton

1 Introduction

Many Embedded Systems are indeed Software Based Control Systems (SBCSs). 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. 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).

Software generation from models and formal specifications forms the core of Model Based Design of embedded software [21]. This approach is particularly interesting for SBCSs since in such a case system level (formal) specifications are much easier to define than the control software behavior itself.

Fig. 1 shows the typical control loop skeleton for an SBCS. Measures from plant sensors go through an AD (analog-to-digital) conversion (quantization) before being processed (line 2) and commands from the control software go through a DA (digital-to-analog) 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).
For SBCSs system level specifications are typically given with respect to the desired behaviour 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 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 [6, 1, 19, 5] (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, the control software may have a WCET greater than the sampling time (line 1 in Fig. 1). This invalidates the schedulability analysis (typically carried out before the control software is completed) and may trigger redesign of the software or even of its functional specifications (in order to simplify its design).

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

The previous considerations motivate research on methods and tools that from the plant model (as a hybrid system), from formal specifications for the closed loop system behaviour (System Level Formal Specifications) and from implementation specifications (that is, number of bits used in the quantization process) can generate correct-by-construction control software satisfying
the given specifications. This is the focus of the present paper.

1.1 Our Main Contributions

We model the controlled system (plant) as a *Discrete Time Linear Hybrid System* (DTLHS), that is a discrete time hybrid system whose dynamics is defined as a *linear predicate* (i.e., a boolean combination of linear constraints) on its variables. We model system level safety as well as liveness specifications as set 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 [22, 23] it is possible to 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-algorithms.

We present constructive algorithms defining, respectively, a sufficient and a necessary condition for existence of a solution to our control software synthesis problem. Given a DTLHS model $\mathcal{H}$ for the plant, a quantization schema (i.e. how many bits we use for AD conversion) and system level formal specifications, our algorithms will return 1 if a solution exists (respectively, does not exist) and 0 when unable to decide (unavoidable case since our problem is undecidable). Furthermore, when our sufficient condition is satisfied, we return a pair of C functions $\text{Control}_\text{Law}$, $\text{Controllable}_\text{Region}$ such that: function $\text{Control}_\text{Law}$ implements a (near time-optimal) *Quantized Feedback Controller* (QFC) for $\mathcal{H}$ meeting the given system level formal specifications and function $\text{Controllable}_\text{Region}$ computes the set of states on which $\text{Control}_\text{Law}$ is guaranteed to work correctly (*controllable region*). Both functions have a *Worst Case Execution Time* (WCET) guaranteed to be linear in the number of bits of the state quantization schema. Furthermore, function $\text{Control}_\text{Law}$ 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 and present experimental results on using our tool QKS to synthesize robust control software for a widely used mixed-mode analog circuit: the buck DC-DC converter (e.g. see [40]). This is an interesting and challenging example (e.g., see [16] and Sect. 1 of [30]) for automatic synthesis of correct-by-construction control
software from system level formal specifications.

Our experimental results 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.

1.2 Related Work

This paper is a journal version of [30] which is extended hereby providing omitted proofs and algorithms.

Control software synthesis for continuous time linear systems (no switching) has been studied in [34] (and citations thereof), and in [10] (for piecewise affine systems). Such works do not account for state feedback quantization. Thus (formal) system level correctness of the generated software is not addressed. Of course Quantized Feedback Control has been widely studied in control engineering (e.g. see [18]). However such research does not address hybrid systems (our case) 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.

When the plant model is a Timed Automaton (TA) [6] the reachability and control law synthesis problems have both been widely studied. Examples are in [27, 13, 29] and citations thereof. When the plant model is a Linear Hybrid Automaton (LHA) [1, 5] reachability and existence of a control law are both undecidable problems [22, 23]. This, of course, has not prevented devising effective (semi) algorithms for such problems. Examples are in [5, 19, 17, 44]. Much in the same spirit here we give a necessary condition along with a constructive sufficient condition for control software existence. Note that none of the above mentioned papers address control software synthesis since they all assume exact (i.e. real valued) state measures (that is, state feedback quantization is not considered).

Finite horizon control of Piecewise Affine Discrete Time Hybrid Systems (PWA-DTHS) has been studied using a MILP based approach. See, for example, [11]. Explicit finite horizon control synthesis algorithms for discrete time (possibly non-linear) hybrid systems have been studied in [15] and ci-
tations thereof. Such approaches cannot be directly used in our context since they address synthesis of finite horizon controllers and do not account for quantization. Correct-by-construction software synthesis in a finite state setting has been studied, for example, in [8, 43, 41, 14]. Such approaches cannot be directly used in our context since they cannot handle continuous state variables.

Quantization can be seen as a sort of abstraction, which has been widely studied in a hybrid system formal verification context (e.g., see [4, 2]). Note however that in a verification context abstractions are designed so as to ease the verification task whereas in our setting quantization is a design requirement since it models a hardware component (AD converter) which is part of the specification of the control software synthesis problem. Indeed, in our setting, we have to design a controller notwithstanding the nondeterminism stemming from the quantization process. As a result, the techniques used to devise clever abstractions in a verification setting cannot be directly used in our synthesis setting where quantization is given.

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

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 \(x^r\). Similarly, to clarify that a variable \(x\) is discrete (i.e. integer valued) we may write \(x^d\). Boolean variables are discrete variables ranging on the set \(B = \{0, 1\}\). We may write \(x^b\) to denote a boolean variable. Analogously \(X^r\) \((X^d, X^b)\) denotes the sequence of real (integer, boolean) variables in \(X\). Unless otherwise stated, we suppose \(D_{X^r} = \mathbb{R}^{\mid X^r\mid}\) and \(D_{X^d} = \mathbb{Z}^{\mid X^d\mid}\). Finally, if \(x\) is a boolean variable we write \(\bar{x}\) for \((1 - x)\).
2.1 Predicates

A *linear expression* over a list of variables $X$ is a linear combination of variables in $X$ with rational coefficients. 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 $x \geq a \land x \leq b$, being $x \in X$.

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 $X^* \in D_X$ the sequence of values $[v(x_1), \ldots, v(x_n)]$. By abuse of language, we call valuation also the sequence of values $X^*$. A *satisfying assignment* to a predicate $P$ over $X$ is a valuation $X^*$ such that $P(X^*)$ holds. If a satisfying assignment to a predicate $P$ over $X$ exists, we say that $P$ is *feasible*. Abusing notation, we may denote with $P$ the set of satisfying assignments to the predicate $P(X)$. Two predicates $P$ and $Q$ over $X$ are *equivalent*, notation $P \equiv Q$, 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* $y \rightarrow C(X)$ (if $y$ then $C(X)$) denotes the predicate $((y = 0) \lor C(X))$. Similarly, we use $\bar{y} \rightarrow C(X)$ (if not $y$ then $C(X)$) to denote the predicate $((y = 1) \lor C(X))$. A *guarded predicate* is a conjunction of either constraints or guarded constraints. When a guarded predicate is bounded, it can be easily transformed into a (bounded) conjunctive predicate, as stated by the following proposition (details are in App. [A.1]).

**Proposition 1.** For each bounded guarded predicate $P(X)$, it is possible to compute an equivalent bounded conjunctive predicate $Q(X)$. 
2.2 Mixed Integer Linear Programming

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 \( X^* \) 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 \((\text{max}, 0, A(X))\). We write also \( A(X) \) for \((\text{max}, 0, A(X))\). We write \((\text{max}, -J(X), A(X))\).

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 \( \text{optimalValue}(\text{max}, J(X), A(X)) \) that returns either the optimal value of the MILP problem \((\text{max}, J(X), A(X))\) or \( +\infty \) if such MILP problem is unbounded or unfeasible.

2.3 Labeled Transition Systems

A Labeled Transition System (LTS) is a tuple \( \mathcal{S} = (S, A, T) \) where \( S \) is a (possibly infinite) set of states, \( A \) is a (possibly infinite) set of actions, and \( T : S \times A \times S \rightarrow \mathbb{B} \) is the transition relation of \( \mathcal{S} \). We say that \( T \) (and \( \mathcal{S} \)) is deterministic if \( T(s, a, s') \land T(s, a, s'') \) implies \( s' = s'' \), and nondeterministic otherwise. Let \( s \in S \) and \( a \in A \). We denote with \( \text{Adm}(\mathcal{S}, s) \) the set of actions admissible in \( s \), that is \( \text{Adm}(\mathcal{S}, s) = \{ a \in A \mid \exists s' T(s, a, s') \} \) and with \( \text{Img}(\mathcal{S}, s, a) \) the set of next states from \( s \) via \( a \), that is \( \text{Img}(\mathcal{S}, s, a) = \{ s' \in S \mid T(s, a, s') \} \). We call transition a triple \((s, a, s') \in S \times A \times S\), and self loop a transition \((s, a, s)\). A transition \((s, a, s') \) [self loop \((s, a, s)\)] is a transition [self loop] of \( \mathcal{S} \) iff \( T(s, a, s') \) \([T(s, a, s)]\).

A run or path for an LTS \( \mathcal{S} \) is a sequence \( \pi = s_0, a_0, s_1, a_1, s_2, a_2, \ldots \) of states \( s_t \) and actions \( a_t \) such that \( \forall t \geq 0 \ T(s_t, a_t, s_{t+1}) \). The length \( |\pi| \) of a finite run \( \pi \) 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 \( \pi^{(A)}(t) = a_t \).

Given two LTSs \( \mathcal{S}_1 = (S, A, T_1) \) and \( \mathcal{S}_2 = (S, A, T_2) \), we say that \( \mathcal{S}_1 \) refines \( \mathcal{S}_2 \) (notation \( \mathcal{S}_1 \sqsubseteq \mathcal{S}_2 \)) iff \( T_1(s, a, s') \) implies \( T_2(s, a, s') \) for each state

\[1\]

Note that, with this definition, deterministic LTSs are not a special case of nondeterministic ones, as it usually is.
$s, s' \in S$ and action $a \in A$. The refinement relation is a partial order on LTSs.

## 3 Discrete Time Linear Hybrid Systems

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

**Definition 1.** A Discrete Time Linear Hybrid System is a tuple $\mathcal{H} = (X, U, Y, N)$ where:

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

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

By Prop. 1, any bounded guarded predicate can be transformed into a conjunctive predicate. 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.

Note that DTLHSs can effectively model linear algebraic constraints involving both continuous as well as discrete variables. Therefore many embedded control systems may be modeled as DTLHSs.
Example 1. Let $x$ be a continuous variable, $u$ be a boolean variable, and $N(x, u, x') \equiv [u \rightarrow x' = \alpha x] \land [u \rightarrow x' = \beta x]$ be a guarded predicate with $\alpha = \frac{1}{2}$ and $\beta = \frac{3}{2}$. Then $\mathcal{H} = \langle \{x\}, \{u\}, \emptyset, N \rangle$ is a DTLHS.

Note that $\mathcal{H}$ is deterministic. Adding nondeterminism to $\mathcal{H}$ allows us to address the problem of (bounded) variations in the DTLHS parameters. For example, variations in the parameter $\alpha$ can be modelled with a tolerance $\rho \in [0, 1]$ for $\alpha$. This replaces $N$ with: $N(\rho) \equiv [u \rightarrow x' \leq (1 + \rho)\alpha x] \land [u \rightarrow x' \geq (1 - \rho)\alpha x] \land [u \rightarrow x' = \beta x]$. We have that $\mathcal{H}(\rho) = \langle \{x\}, \{u\}, \emptyset, N(\rho) \rangle$, for $\rho \in (0, 1]$, is a nondeterministic DTLHS. Note that, as expected, $\mathcal{H}(0) = \mathcal{H}$.

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

**Definition 2.** Let $\mathcal{H} = \langle X, U, Y, N \rangle$ be a DTLHS. The dynamics of $\mathcal{H}$ is defined by the Labeled Transition System $\text{LTS}(\mathcal{H}) = \langle D_X, D_U, \tilde{N} \rangle$ where: $\tilde{N} : D_X \times D_U \times D_X \rightarrow \mathbb{B}$ is a function s.t. $\tilde{N}(x, u, x') \equiv \exists y \in D_Y N(x, u, y, x')$. A state $x$ for $\mathcal{H}$ is a state $x$ for $\text{LTS}(\mathcal{H})$ and a run (or path) for $\mathcal{H}$ is a run for $\text{LTS}(\mathcal{H})$ (Sect. 2.3).

Note that a DTLHS $\mathcal{H}$ is deterministic iff $\text{LTS}(\mathcal{H})$ is deterministic.

**Example 2.** Let $\mathcal{H} = \langle \{x\}, \{u\}, \emptyset, N \rangle$ be the DTLHS of Ex. 1. Then a sequence $\pi$ is a run on $\mathcal{H}$ iff $\pi^{(S)}(t) = \pi^{(S)}(0) \frac{\alpha^{i}}{\alpha^{i}}$, being $i = \sum_{j=0}^{t-1} \pi^{(A)}(j)$ (taking 0 the summation on an empty set).

3.1 Buck DC-DC Converter as a DTLHS

The buck DC-DC converter (Fig. 2) is a mixed-mode analog circuit converting the DC input voltage ($V_i$ in Fig. 2) to a desired DC output voltage ($v_O$ in Fig. 2). 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. [40]) as well as on-chip to support Dynamic Voltage and Frequency Scaling (DVFS) in multicore processors (e.g. [26, 39]). Because of its widespread use, control schemas for buck DC-DC converters have been widely studied (e.g. see [26, 39, 40, 45]). The typical software based approach (e.g. see [40]) is to control the switch $u$ in Fig. 2 (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 Fig. 2 can be modeled as a DTLHS $H = (X, U, Y, N)$. The circuit state variables are $i_L$ and $v_C$. However we can also use the pair $i_L, v_O$ as state variables in $H$ model since there is a linear relationship between $i_L, v_C$ and $v_O$, namely:

$$v_O = \frac{r_C}{r_C + R}i_L + \frac{R}{r_C + R}v_C.$$  

Such considerations lead to use the following sets of variables to model $H$:

$$X = \begin{bmatrix} i_L, \\ v_O \end{bmatrix},$$

$$U = \begin{bmatrix} u \end{bmatrix},$$

$$Y_r = \begin{bmatrix} i_u, v_u, i_D, v_D \end{bmatrix}$$

and

$$Y_b = \begin{bmatrix} q \end{bmatrix}.$$  

Note how $H$ auxiliary variables $Y$ stem from the constitutive equations of the switching elements (i.e. the switch $u$ and the diode D in Fig. 2). From a simple circuit analysis (e.g. see [28]) we have the following equations:

$$\dot{i}_L = a_{1,1}i_L + a_{1,2}v_O + a_{1,3}v_D$$

(1)

$$\dot{v}_O = a_{2,1}i_L + a_{2,2}v_O + a_{2,3}v_D$$

(2)

where the coefficients $a_{i,j}$ depend on the circuit parameters $R, r_L, r_C, L$ and $C$ in the following way:

$$a_{1,1} = -\frac{r_C}{L}, \quad a_{1,2} = -\frac{1}{L}, \quad a_{1,3} = -\frac{1}{C}, \quad a_{2,1} = \frac{R}{r_C + R}[-\frac{r_C}{L} + \frac{1}{C}], \quad a_{2,2} = \frac{1}{r_C + R}[\frac{1}{L} + \frac{1}{C}], \quad a_{2,3} = -\frac{1}{L} \frac{r_C}{r_C + R}.$$  

Using a discrete time model with sampling time $T$ (writing $x'$ for $x(t+1)$) we have:

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

(3)

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

(4)

The algebraic constraints stemming from the constitutive equations of the switching elements are the following:

$$q \rightarrow v_D = 0 \quad (5) \quad \bar{q} \rightarrow v_D \leq 0 \quad (9)$$

$$q \rightarrow i_D \geq 0 \quad (6) \quad \bar{q} \rightarrow v_D = R_{off}i_D \quad (10)$$

$$u \rightarrow v_u = 0 \quad (7) \quad \bar{u} \rightarrow v_u = R_{off}i_u \quad (11)$$

$$i_D = i_L - i_u \quad (8) \quad v_D = v_u - V_i \quad (12)$$

The transition relation $N$ of $H$ is given by the conjunction of the constraints in Eqs. (5)–(12).
4 Quantized Feedback Control

In this section, we formally define the Quantized Feedback Control Problem for DTLHSs (Sect. 4.3). To do this, first we give the definition of Feedback Control Problem for LTSs (Sect. 4.1), and then for DTLHSs (Sect. 4.2).

4.1 Feedback Control Problem for LTSs

We begin by extending to possibly infinite LTSs the definitions in [43, 14] for finite LTSs. In what follows, let $S = (S, A, T)$ be an LTS, $I, G \subseteq S$ be, respectively, the initial and goal regions.

Definition 3. A controller for an LTS $S$ is a function $K : S \times A \rightarrow B$ such that $\forall s \in S, \forall a \in A$, if $K(s, a)$ 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, $\text{Dom}(K) = \{ s \in S | \exists a K(s, a) \}$. $S^k$ denotes the closed loop system, that is the LTS $(S, A, T^k)$, where $T^k(s, a, s') = T(s, a, s') \land K(s, a)$. An LTS control problem is a triple $(S, I, G)$.

Example 3. 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 Fig. 3 and let $S_1$ be the LTS $(S, A, T_1)$ where the transition relation $T_1$ consists of all arrows in Fig. 3. Any function $K : S \times A \rightarrow B$ is a controller for $S_1$, since for all states $s \in S$ $\text{Adm}(S_1, s) = A$. On the other hand, a function $K$ is a controller for $S_0$ iff $s \neq 0 \rightarrow K(s, 1) = 0$.

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 [8] if either it is infinite or its last state $\pi(|\pi|)$ has no successors (i.e. $\text{Adm}(S, \pi(|\pi|)) = \emptyset$). We denote with $\text{Path}(S, s, a)$ the set of fullpaths of $S$ starting in state $s$ with action $a$, i.e. the set of fullpaths $\pi$ such that $\pi(0) = s$ 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 $\pi(0)$ to the goal on $\pi$. That is, if there exists $n > 0$ s.t. $\pi(n) \in G$, then $J(S, G, \pi) = \min\{ n \mid n > 0 \land \pi(n) \in G \}$. 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 $\sup \emptyset = +\infty$ and $\inf \emptyset = -\infty$, the worst case distance (pessimistic view) of a state
Definition 5. \( J_{\text{strong}}(\mathcal{S}, G, s) = \sup \{ J(\mathcal{S}, G, s, a) \mid a \in \text{Adm}(\mathcal{S}, s) \} \), where: \( J(\mathcal{S}, G, s, a) = \sup \{ J(\mathcal{S}, G, \pi) \mid \pi \in \text{Path}(\mathcal{S}, s, a) \} \). The best case distance (optimistic view) of a state \( s \) from the goal region \( G \) is \( J_{\text{weak}}(\mathcal{S}, G, s) = \sup \{ J(\mathcal{S}, G, s, a) \mid a \in \text{Adm}(\mathcal{S}, s) \} \), where: \( J(\mathcal{S}, G, s, a) = \inf \{ J(\mathcal{S}, G, \pi) \mid \pi \in \text{Path}(\mathcal{S}, s, a) \} \).

**Definition 4.** Let \( \mathcal{P} = (\mathcal{S}, I, G) \) be an LTS control problem and \( K \) be a controller for \( \mathcal{S} \) such that \( I \subseteq \text{Dom}(K) \). 

\( K \) is a strong solution to \( \mathcal{P} \) if for all \( s \in \text{Dom}(K) \), \( J_{\text{strong}}(\mathcal{S}^{(K)}, G, s) \) is finite. \( K \) is a weak solution to \( \mathcal{P} \) if for all \( s \in \text{Dom}(K) \), \( J_{\text{weak}}(\mathcal{S}^{(K)}, G, s) \) is finite.

An optimal strong [weak] solution to \( \mathcal{P} \) is a strong [weak] solution \( K^* \) to \( \mathcal{P} \) such that for all strong [weak] solutions \( K \) to \( \mathcal{P} \), for all \( s \in \mathcal{S} \) we have: \( J_{\text{strong}}(\mathcal{S}^{(K^*)}, G, s) \leq J_{\text{strong}}(\mathcal{S}^{(K)}, G, s) \) \( J_{\text{weak}}(\mathcal{S}^{(K^*)}, G, s) \leq J_{\text{weak}}(\mathcal{S}^{(K)}, G, s) \).

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

**Example 4.** Let \( \mathcal{S}_0, \mathcal{S}_1 \) be the LTSs in Ex. 3. Let \( \mathcal{P}_0 = (\mathcal{S}_0, I, G) \) and \( \mathcal{P}_1 = (\mathcal{S}_1, I, G) \) be two control problems, where \( I = \{-1, 0, 1\} \) and \( G = \{0\} \). The controller \( K(s, a) \equiv [s \neq 0 \rightarrow a = 0] \) is a strong solution to the control problem \( \mathcal{P}_0 \). Observe that \( K \) is not optimal. Indeed, let us consider \( \tilde{K}(s, a) \equiv a = 0 \). Since \( K \) enables action 1 in state 0, we have that \( J_{\text{strong}}(\mathcal{S}_0^{(K)}, G, 0) = 2 \). Since \( \tilde{K} \) enables action 0 only, the path \( \pi \) s.t. \( \pi^{(S)}(t) = 0 \) and \( \pi^{(A)}(t) = 0 \) for all \( t \geq 0 \) is the unique fullpath of \( \mathcal{S}_0^{(K)} \) starting from state 0, and therefore \( J_{\text{strong}}(\mathcal{S}_0^{(K)}, G, 0) = 1 \).

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 \( J_{\text{strong}}(\mathcal{S}_1^{(K)}, G, 1) = J_{\text{strong}}(\mathcal{S}_1^{(K)}, G, -1) = \infty \) because of the dotted self loops \( (1, 0, 1) \) and \( (-1, 0, -1) \) of \( T_1 \).

**Definition 5.** The most general optimal strong [weak] solution to \( \mathcal{P} \) (strong [weak] mgo in the following) is an optimal strong [weak] solution \( \bar{K} \) to \( \mathcal{P} \) such
that for all other optimal strong [weak] solutions $K$ to $\mathcal{P}$, for all $s \in S$, for all $a \in A$ we have that $K(s,a) \rightarrow \tilde{K}(s,a)$.

The definition of most general optimal controller is well posed as stated by the following proposition (details are in App. A.2).

**Proposition 2.** An LTS control problem $(S, \emptyset, G)$ has always an unique strong [weak] mgo $K^*$. Moreover, for all $I \subseteq S$, we have:

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

Our control synthesis algorithm (Alg. 1 in Sect. 6.1) makes use of a variant of the symbolic (i.e. OBDD based) algorithm in [14] for the computation of mgos (function `strongCtr`, line 3), and a variant of the algorithm in [43] to verify the existence of a weak solution (function `existsWeakCtr`, line 6). The proof of Prop. 2 is essentially also a correctness proof for function `strongCtr`. It can be easily adapted to prove the uniqueness of the weak most general optimal solution, and thus to prove the correctness of function `existsWeakCtr` (details are in App. A.3).

**Example 5.** Let $\mathcal{P}_0, \mathcal{P}_1, K$ and $\tilde{K}$ be as in Ex. 4. $K$ is the weak mgo for $\mathcal{P}_1$. $\tilde{K}$ is the strong mgo for $\mathcal{P}_0$.

**Remark 1.** Note that if $K$ is a strong solution to $(S, I, G)$ 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 Def. 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.
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 6.** 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 $(\text{LTS}(\mathcal{H}), I, G)$. Thus, a controller $K : D_X \times D_U \to B$ is a strong [weak] solution to $(\mathcal{H}, I, G)$ iff it is a strong [weak] solution to $(\text{LTS}(\mathcal{H}), I, G)$.

For DTLHS control problems, usually robust controllers are desired. That is, controllers that, notwithstanding nondeterminism in the plant (e.g., due to parameter variations, see Ex. 1), 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., capacitor voltage and inductor current in Sect. 3.1) and will not measure auxiliary variables (e.g., diode state in Sect. 3.1).

**Example 6.** The typical goal of a controller for the buck DC-DC converter in Sect. 3.1 is keeping the output voltage $v_O$ close enough to a given reference value $V_{\text{ref}}$. This leads to the control problem $\mathcal{P} = (\mathcal{H}, I, G)$ where $\mathcal{H}$ is defined in Sect. 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 [18] 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 7.** A quantization function $\gamma$ for a real interval $I = [a, b]$ is a non-decreasing function $\gamma : [a, b] \to \hat{I}$, where $\hat{I}$ is a bounded integer interval $[\gamma(a), \gamma(b)] \subseteq \mathbb{Z}$. The quantization step of $\gamma$, notation $\|\gamma\|$, is defined as $\sup\{|w - z| \mid w, z \in I \land \gamma(w) = \gamma(z)\}$. 

15
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 8.** 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 $a_w, b_w \in \mathcal{D}_w$ (that is, $A$ explicitly bounds each variable in $W$). For each $w \in W$, we define $A_w = \{v \in \mathcal{D}_w \mid a_w \leq v \leq b_w\}$ as the admissible region for variable $w$. Moreover, we define $A_V = \prod_{v \in V} A_v$, with $V \subseteq W$, as the admissible region for variables in $V$.

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

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

Note that $\Gamma$ is univocally defined by its quantizations $\gamma_w$ where $w$ is a real valued variable since discrete variables are not affected by the quantization process. 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 7.** Let $\mathcal{P}$ be the DTLHS control problem defined in Ex. 8. Let us consider the quantization $Q = (A, \Gamma)$, where $A(x, u) \equiv -2.5 \leq x \leq 2.5 \land 0 \leq u \leq 1$. $A$ defines the admissible regions $A_x = A_X = [-2.5, 2.5]$ for $X$ and $A_u = A_U = \{0, 1\}$ for $U$. Let $\Gamma = \{\gamma_x, \gamma_u\}$, with $\gamma_x(x) = \text{round}(x/2)$ (where round$(x) = \lfloor x \rfloor + \lfloor 2(x - \lfloor x \rfloor) \rfloor$ is the usual rounding function) and $\gamma_u(u) = u$. Note that $\gamma_x(x) = -1$ for all $x \in [2.5, -1]$, $\gamma_x(x) = 0$ for all $x \in (-1, 1)$ and $\gamma_x(x) = 1$ for all $x \in [1, 2.5]$. Thus, we have that $\Gamma(A_x) = \{-1, 0, 1\}$, $\Gamma(A_u) = \{0, 1\}$ and $\|\Gamma\| = 1$.

Quantization, i.e. representing reals with integers, unavoidably introduces errors in reading real-valued plant sensors in the control software. We address this problem in the following way. First, we introduce the definition of $\varepsilon$-solution. Essentially, we require that the controller drives the plant “near enough” (up to a given error $\varepsilon$) to the goal region $G$. To this end, we define the $\varepsilon$-relaxation of a set in $\mathbb{R}^n \times \mathbb{Z}^m$. 


Definition 9. Let \( \varepsilon \) be a nonnegative real number, \( W \subseteq \mathbb{R}^n \times \mathbb{Z}^m \). 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 10. 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 8. Let \( \mathcal{H} \) be the DTLHS described in Ex. 7. We consider the control problem defined by the initial region \( I = \mathbb{R} \times (-2.5, 2.5) \) 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 11. 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 \( Q \)-Quantized Feedback Control (QFC) strong [weak] solution to \( \mathcal{P} \) is a strong [weak] \( \|\Gamma\| \)-solution \( K : D_X \times D_U \to B \) 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 \( \hat{K} : \Gamma(A_X) \times \Gamma(A_U) \to B \).

Note that by Def. 11 a necessary condition for the existence of a \( Q \)-QFC (strong as well as weak) solution is that \( G, I \subseteq A_X \). This is indeed the case in real-world systems and in all our examples. Furthermore, note that a \( Q \)-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 behaviour inside the admissible region.

Example 9. Let \( \mathcal{P} \) be the DTLHS control problem defined in Ex. 8 and \( Q = (A, \Gamma) \) be the quantization defined in Ex. 7. Let \( \hat{K} \) be defined by \( \hat{K}(\hat{x}, \hat{u}) \equiv [\hat{x} \neq 0 \to \hat{u} = 0] \). For any \( \varepsilon > 0 \), the quantized controller \( K(x, u) = \hat{K}(\Gamma(x), \Gamma(u)) \) is an \( \varepsilon \)-solution to \( \mathcal{P} \), and hence it is a \( Q \)-QFC solution.
Along the same lines of similar undecidability proofs [23], it is possible to show that existence of a \( \mathcal{Q} \) QFC solution to a DTLHS control problem (DTLHS quantized control problem) is undecidable (details are in App. A.4).

**Theorem 3.** The DTLHS quantized control problem is undecidable.

## 5 Control Abstraction

A quantization naturally induces an abstraction of a DTLHS. Motivated by finding QFC solutions in the abstract model, in this paper we introduce a novel notion of abstraction, namely *control abstraction*.

Control abstraction (Def. 13) models how a DTLHS \( \mathcal{H} \) is *seen* from the control software after AD/DA conversions. Since QFC control rests on AD conversion we must be careful not to drive the plant outside the bounds in which AD conversion works correctly. This leads to the definition of admissible action (Def. 12). Intuitively, an action is admissible in a state if it never drives the system outside of its admissible region.

**Definition 12.** Let \( \mathcal{H} = (X, U, Y, N) \) be a DTLHS and \( \mathcal{Q} = (A, \Gamma) \) be a quantization for \( \mathcal{H} \).

An action \( u \in A_U \) is \( A \)-admissible in \( s \in A_X \) if for all \( s' \), \( \exists y \in A_Y \ N(s, u, y, s') \) implies \( s' \in A_X \).

An action \( \hat{u} \in \Gamma(A_U) \) is \( \mathcal{Q} \)-admissible in \( \hat{s} \in \Gamma(A_X) \) if for all \( s \in \Gamma^{-1}(\hat{s}) \), \( u \in \Gamma^{-1}(\hat{u}) \), \( u \) is \( A \)-admissible for \( s \) in \( \mathcal{H} \).

**Example 10.** Let \( \mathcal{H} \) be the DTLHS defined in Ex. 7 and \( \mathcal{Q} \) be the quantization defined in Ex. 7. Then the action \( u = 1 \) is not \( A \)-admissible in the state \( s = 2 \) since we have \( N(2, 1, 3) \), and \( s' = 3 \) is outside the admissible region \( A_X \). As a consequence, quantization the action \( \hat{u} = 1 \) is not \( \mathcal{Q} \)-admissible in the state \( \hat{s} = 1 \), since \( 2 \in \Gamma^{-1}(1) \). Analogously, \( u = 1 \) is not \( A \)-admissible in \( s = -2 \) since we have \( N(-2, 1, 3) \). Thus \( \hat{u} = 1 \) is not \( \mathcal{Q} \)-admissible in \( \hat{s} = -1 \), since \( -2 \in \Gamma^{-1}(-1) \). It is easy to see that no other \( \hat{u} \in \Gamma(A_U), \hat{s} \in \Gamma(A_X) \) exist s.t. \( \hat{u} \) is not \( \mathcal{Q} \)-admissible in \( \hat{s} \).

**Definition 13.** Let \( \mathcal{H} = (X, U, Y, N) \) be a DTLHS and \( \mathcal{Q} = (A, \Gamma) \) be a quantization for \( \mathcal{H} \). We say that the LTS \( \hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N}) \) is a \( \mathcal{Q} \) control abstraction of \( \mathcal{H} \) if its transition relation \( \hat{N} \) satisfies the following conditions:
1. Each abstract transition stems from a concrete transition. Formally: for all \( \hat{s}, \hat{s}' \in \Gamma(A_X) \), \( \hat{u} \in \Gamma(A_U) \), if \( \hat{N}(\hat{s}, \hat{u}, \hat{s}') \) then there exist \( s \in \Gamma^{-1}(\hat{s}) \), \( u \in \Gamma^{-1}(\hat{u}) \), \( s' \in \Gamma^{-1}(\hat{s}') \), \( y \in A_Y \) such that \( N(s, u, y, s') \).

2. Each concrete transition is faithfully represented by an abstract transition, whenever it is not a self loop and its corresponding abstract action is \( Q \)-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 \( \hat{N}(\Gamma(s), \Gamma(u), \Gamma(s')) \).

3. If there is no upper bound to the length of concrete paths inside the counter-image of an abstract state then there is an abstract self loop. Formally: for all \( \hat{s} \in \Gamma(A_X) \), \( \hat{u} \in \Gamma(A_U) \), if it exists an infinite run \( \pi \) in \( H \) 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 \( (\hat{s}, \hat{u}, \hat{s}) \) for which there exists an infinite run \( \pi \) in \( H \) such that \( \forall t \in \mathbb{N} \pi^{(s)}(t) \in \Gamma^{-1}(\hat{s}) \) and \( \pi^{(A)}(t) \in \Gamma^{-1}(\hat{u}) \) is said to be a non-eliminable self loop. A self loop \( (\hat{s}, \hat{u}, \hat{s}) \) such that \( (\hat{s}, \hat{u}, \hat{s}) \) is not a non-eliminable self loop, but for which a concrete witness exists (i.e., there exist \( s, s' \in \Gamma^{-1}(\hat{s}) \), \( u \in \Gamma^{-1}(\hat{u}) \), \( y \in A_Y \) such that \( N(s, u, y, s') \)) is said to be an eliminable self loop.

We say that \( \hat{H} \) is a control abstraction of \( H \) if \( \hat{H} \) is a \( Q \) control abstraction of \( H \) for some quantization \( Q \). We denote with \( C(H, Q) \) the set of all \( Q \) control abstractions of \( H \).

**Example 11.** Let \( H \) be as in Ex. 7 and \( Q = (A, \Gamma) \) be as in Ex. 8. \( Q \) control abstractions of \( H \) are depicted in Fig. 8. Any \( Q \) control abstraction \( \hat{H} \) of \( H \) has the form \( \{\{-1, 0, 1\}, \{0, 1\}, \hat{N}\} \) where the set \( \hat{N} \) of transitions always contains at least all continuous arrows in the automaton depicted in Fig. 8 and some dotted arrows. By condition 5 in Def. 13, self loops \( (0, 0, 0) \) and \( (0, 1, 0) \) are non-eliminable, thus they must belong to all \( Q \) control abstractions. In fact all paths starting from 0 will remain in 0 forever. All other self loops are eliminable. Note that, despite \( H \) being deterministic, all \( Q \) control abstractions of \( H \) are nondeterministic.

Along the same lines of similar undecidability proofs [23, 3], it is possible to show that we cannot algorithmically state if a self loop is eliminable or non-eliminable (details are in App. A.5).

**Proposition 4.** Given a DTLHS \( H \) and a quantization \( Q \), it is undecidable to determine if a self loop is non-eliminable.
Note that if in Def. 13 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 [2] and citations thereof). As a result, any abstraction is also a control abstraction whereas a control abstraction in general is not an abstraction since some self loops or some non admissible actions may be missing.

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

**Definition 14.** Let $H = (X, U, Y, N)$ be a DTLHS and $Q = (A, \Gamma)$ be a quantization for $H$. A $Q$ control abstraction $\hat{H} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$ of $H$ is an admissible $Q$ control abstraction iff, for all $\hat{s} \in \Gamma(A_X), \hat{u} \in \Gamma(A_U)$ s.t. $\hat{u} \in \text{Adm}(\hat{H}, \hat{s})$:

1. $\hat{u}$ is $Q$-admissible in $\hat{s}$, i.e. each abstract transition contains an admissible action;

2. $\forall s \in \Gamma^{-1}(\hat{s}) \forall u \in \Gamma^{-1}(\hat{u}) \exists s' \in D_X \exists y \in D_Y N(s, u, y, s')$, i.e. each concrete state in $\Gamma^{-1}(\hat{s})$ has a successor for all concrete actions in $\Gamma^{-1}(\hat{u})$.

We say $\hat{H}$ is a full $Q$ control abstraction if it satisfies properties 4 and 5 of Def. 13, plus the following property (derived from property 2 of Def. 13): for all $s, s' \in A_X, u \in A_U$ such that $\exists y.N(s, u, y, s')$, if $\Gamma(s) \neq \Gamma(s')$ then $\hat{N}(\Gamma(s), \Gamma(u), \Gamma(s'))$.

We denote with $C_a(H, Q)$ $|C_f(H, Q)|$ the set of all admissible [full] $Q$ control abstractions of $H$.

It is easy to show that, if all actions are $Q$-admissible in all states, then a full $Q$ control abstraction is also an admissible $Q$ control abstraction and vice versa. Otherwise, a $Q$ control abstraction cannot be admissible and full at the same time. Moreover, if exactly one (abstract state, abstract action) pair $\hat{s}, \hat{u}$ exists s.t. $\hat{u}$ is not $Q$-admissible in $\hat{s}$, then any $Q$ control abstraction is either admissible or full (i.e., full and admissible $Q$ control abstractions are a partition of $C(H, Q)$). Otherwise, there will be a $Q$ control abstraction which is neither full nor admissible. Finally note that, if $H_1, H_2$ are $Q$ control abstractions of $H$ s.t. $H_1$ is admissible and not full and $H_2$ is full and not admissible, then $H_2 \not\sqsupseteq H_1$.

**Example 12.** Let $H$ be as in Ex. 4, $Q = (A, \Gamma)$ be as in Ex. 7 and $\hat{H} = (\{-1, 0, 1\}, \{0, 1\}, \hat{N})$ be a control abstraction of $H$. For all $Q$ admissible...
control abstractions, \( \hat{N}(1, 1, 1) = \hat{N}(-1, 1, -1) = 0 \), since action 1 is not \( Q \)-admissible neither in \(-1\) nor in 1 (see Ex. 10). On the contrary, for all full \( Q \) control abstractions, \( \hat{N}(1, 1, 1) = \hat{N}(-1, 1, -1) = 1 \). Thus, a control abstraction s.t. \( \hat{N}(1, 1, 1) \oplus \hat{N}(-1, 1, -1) \) (being \( \oplus \) the logical XOR) is neither full nor admissible.

By the definition of quantization, a control abstraction is a finite LTS. Moreover, two different admissible [full] \( Q \) control abstractions only differ in the number of self loops (Fact 5) and the set of control abstractions is a finite lattice with respect to the LTS refinement relation (Fact 6) (details are in App. A.6). This implies that \((C(H, Q), \sqsubseteq)\) has maximum and minimum. It is easy to show that the same holds for the lattice of admissible and full control abstractions.

**Fact 5.** Let \( M_1 = (S, A, T_1) \) and \( M_2 = (S, A, T_2) \) be two admissible \( Q \) control abstractions of a DTLHS \( H \), with \( Q \) quantization for \( H \). Then \( \forall \hat{s}, \hat{s}' \in S \text{ s. t. } \hat{s} \neq \hat{s}', \forall \hat{a} \in A \left[ T_1(\hat{s}, \hat{a}, \hat{s}') \iff T_2(\hat{s}, \hat{a}, \hat{s}') \right] \). The same holds if \( M_1, M_2 \) are full \( Q \) control abstractions.

**Fact 6.** 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 admissible [full] \( Q \) control abstractions of \( H \) \((C_a(H, Q), \sqsubseteq) \) \( (C_f(H, Q), \sqsubseteq) \) is a lattice.

**Example 13.** Let \( H \) be as in Ex. 1, \( Q = (A, \Gamma) \) be as in Ex. 7, and \( \hat{H} = (\{-1, 0, 1\}, \{0, 1\}, \hat{N}) \) be a control abstraction of \( H \). The transition relation consisting of continuous arrows only w.r.t. Fig. \( \text{[4]} \) (i.e. the LTS \( S_0 \) in Ex. 3) is the minimum \( Q \) control abstraction of \( H \), whereas the transition relation consisting of all arrows (i.e. the LTS \( S_1 \) in Ex. 3) is the maximum \( Q \) control abstraction of \( H \).

By Facts 5 and 6, the minimum \( Q \) control abstraction is the admissible \( Q \) control abstraction with non-eliminable self loops only. Analogously, the minimum full \( Q \) control abstraction is the full \( Q \) control abstraction with non-eliminable self loops only. Thus, the following proposition is a simple corollary of Prop. 4.

**Proposition 7.** Given a DTLHS \( H \) and a quantization \( Q \), it is undecidable to state if: i) a \( Q \) control abstraction for \( H \) is the minimum \( Q \) control abstraction for \( H \), and ii) a full \( Q \) control abstraction for \( H \) is the minimum full \( Q \) control abstraction for \( H \).
Finally, note that the minimum $Q$ control abstraction is always an admissible $Q$ control abstraction, whilst the maximum $Q$ control abstraction is always a full $Q$ control abstraction.

### 5.1 Maximum and Minimum Control Abstractions

Since finding a solution to a DTLHS quantized control problem is undecidable (Theor. 3), we cannot hope for a constructive sufficient and necessary condition for the existence of a $Q$ QFC solution, 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. Note that both the sufficient and the necessary conditions might be false. In such a case our approach is not able to decide if a $Q$ QFC solution exists or not.

We base our sufficient condition on computing a (close to) minimum admissible $Q$ control abstraction, and our necessary condition on computing a (close to) minimum full $Q$ control abstraction. Theor. 8 gives the foundations for such an approach (details are in App. A.6).

**Theorem 8.** Let $H$ be a DTLHS, $Q = (A, \Gamma)$ be a quantization for $H$, and $(H, I, G)$ be a control problem.

1. If $\hat{H}$ is an admissible $Q$ control abstraction and $\hat{K}$ is a strong solution to the LTS control problem $(\hat{H}, \Gamma(I), \Gamma(G))$ then $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $Q$ QFC strong solution to the DTLHS control problem $(H, I, G)$.

2. If $\hat{H}_1, \hat{H}_2$ are two admissible $Q$ control abstractions of $H$ s.t. $\hat{H}_1 \subseteq \hat{H}_2$, and $\hat{K}$ is a strong solution to the LTS control problem $(\hat{H}_2, \Gamma(I), \Gamma(G))$, then $\hat{K}$ is a strong solution to the LTS control problem $(\hat{H}_1, \Gamma(I), \Gamma(G))$.
3. If $\hat{H}$ is a full $Q$ control abstraction and the LTS control problem $(\hat{H}, \Gamma(I), \Gamma(G))$ does not have a weak solution then there exists no $Q$ QFC (weak as well as strong) solution to the DTLHS control problem $(H, I, G)$.

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

Fig. 5 graphically represents a sketch of the correspondence between a concrete DTLHS $H$ and its control abstractions $\hat{H}$ lattices (in the case that at least two non-admissible actions from given states exist).

Example 14. Let $P = (H, I, G)$ be as in Ex. 8 and $Q = (A, \Gamma)$ be as in Ex. 11. For all $Q$ control abstractions $\hat{H}$ in Ex. 11 (and thus for the admissible ones shown in Ex. 12) not containing the eliminable self loops $(-1,0,-1)$ and $(1,0,1)$, $\hat{K}(\hat{x}, \hat{u}) \equiv [\hat{x} \neq 0 \rightarrow \hat{u} = 0]$ (see Ex. 9) is the strong mgo for $(\hat{H}, \Gamma(I), \Gamma(G))$. Thus, $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $Q$ QFC solution to $P$. Weak solutions to $(\hat{H}, \Gamma(I), \Gamma(G))$ exist for all (full) $Q$ control abstractions $\hat{H}$. Note that existence of a $Q$ QFC solution to a control problem depends on $\Gamma$. Let us consider the quantization $Q' = (A, \Gamma')$, where $\Gamma'(w) = \lfloor w/2 \rfloor$. A full $Q'$ control abstraction of $H$ is $L = \{-2, -1, 0, 1\}, \{0, 1\}, \hat{N})$, where the transition $\hat{N}$ is depicted in Fig. 4. $(L, \Gamma'(I), \Gamma'(G))$ has no weak solution.
since there is no path to the goal $\Gamma'(G) = \{0\}$ from states $-2$ and $-1$. Thus $\mathcal{P}$ has no $Q'$ QFC solution.

6 Quantized Controller Synthesis

In this section, we present the quantized controller synthesis algorithm (function $qCtrSyn$ in Alg. 1). Function $qCtrSyn$ takes as input a DTLHS control problem $\mathcal{P} = (\mathcal{H}, I, G)$ and a quantization $Q$. Then, resting on Theor. 8, $qCtrSyn$ computes an admissible $Q$ control abrasion $\hat{\mathcal{M}}$ in order to find a $Q$ QFC strong solution to $\mathcal{P}$ (our sufficient condition), and a full $Q$ control abrasion $\hat{\mathcal{W}}$ to determine if such a solution does not exist (our necessary condition).

Namely, as for the sufficient condition, we compute the strong mgo $\hat{K}$ for the LTS control problem $(\hat{\mathcal{M}}, \Gamma(I), \Gamma(G))$. If $\hat{K}$ exists, then a $Q$ 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 abrasion $\hat{\mathcal{H}}$. However, by point 2 of Theor. 8, $\hat{\mathcal{H}}$ must be lower than $\hat{\mathcal{M}}$ in the hierarchy lattice (see Fig. 5). This suggests to compute $\hat{\mathcal{M}}$ as the minimum (admissible) $Q$ control abrasion of $\mathcal{H}$. Since by Prop. 7 we are not able to compute the minimum $Q$ control abrasion, we compute $\hat{\mathcal{M}}$ as a close to minimum admissible $Q$ control abrasion containing as few eliminable self loops as possible (see Ex. 4).

As for the necessary condition, we compute the weak mgo $\hat{K}$ for the LTS control problem $(\hat{\mathcal{W}}, \Gamma(I), \Gamma(G))$. If $\hat{K}$ does not exists, then a $Q$ QFC (weak as well as strong) solution to $\mathcal{P}$ cannot exist. Note that, if $\hat{K}$ exists, a weak mgo may not exist for some other full $Q$ control abrasion $\hat{\mathcal{H}}$. However, by point 4 of Theor. 8, $\hat{\mathcal{H}}$ must be lower than $\hat{\mathcal{W}}$ in the hierarchy lattice (see Fig. 5). Hence, again by Prop. 7 we compute $\hat{\mathcal{W}}$ as the close to minimum full $Q$ control abrasion, i.e. the full $Q$ control abrasion containing as few eliminable self loops as possible.

6.1 QFC Synthesis Algorithm

Our QFC synthesis algorithm (function $qCtrSyn$ outlined in Alg. 1) takes as input a DTLHS $\mathcal{H} = (X, U, Y, N)$, a quantization $Q = (A, \Gamma)$, 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 $(\mathcal{H}, \Gamma^{-1}(\hat{D}), G)$.

Algorithm 1 QFC Synthesis algorithm $qCtrSyn$

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

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

1. $\hat{I} \leftarrow \Gamma(I)$, $\hat{G} \leftarrow \Gamma(G)$
2. $\hat{M} \leftarrow \text{minCtrlAbs}(\mathcal{H}, Q)$
3. $(b, \hat{D}, \hat{K}) \leftarrow \text{strongCtr}(\hat{M}, \hat{I}, \hat{G})$
4. **if** $b$ **then** return $(\text{Sol}, \hat{D}, \hat{K})$
5. $\hat{W} \leftarrow \text{minFullCtrlAbs}(\mathcal{H}, Q)$
6. **if** existsWeakCtr($\hat{W}, \hat{I}, \hat{G}$) **then** return $(\text{Unk}, \hat{D}, \hat{K})$
7. **else** return $(\text{NoSol}, \hat{D}, \hat{K})$

We represent boolean functions (e.g. the transition relation of $\hat{\mathcal{H}}$) using OBDDs [12] and sets by using their characteristic functions. For the sake of clarity, however, we will present our algorithms using a set theoretic notation for sets and predicates over sets.

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

As said in Sect. 6, we want to compute a close to minimum $Q$ control abstraction $\hat{M}$ of $\mathcal{H}$. That is, we want to compute an admissible $Q$ control abstraction with as few eliminable self loops as possible, given that we cannot hope to rule out all of them by Prop. 4. This is done by function $\text{minCtrlAbs}$ in line 2 which implements an effective heuristic to build $\hat{M}$ (see Sect. 6.4 for further details about $\text{minCtrlAbs}$).

Line 3 determines if a strong mgo to the LTS control problem $\hat{\mathcal{P}} = (\hat{M}, \hat{I}, \hat{G})$ exists by calling function $\text{strongCtr}$ that implements a variant of the algorithm in [14] (details are in App. A.3). Function $\text{strongCtr}$ returns a triple $(b, \hat{D}, \hat{K})$ such that $\hat{K}$ is the strong mgo the LTS control problem $(\hat{M}, \emptyset, \hat{G})$ and $\hat{D} = \text{Dom}(\hat{K})$ is the maximum region of controllable states. If $b$ is True then $\hat{K}$ is a strong mgo for $\mathcal{P}$ (i.e. $\hat{I} \subseteq \hat{D}$) and $qCtrSyn$ returns the tuple $(\text{Sol}, \hat{D}, \hat{K})$ (line 4). By Theor. 8 (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 3-7 $qCtrSyn$ tries to establish if such a solution may exist or not.

Function $\text{minFullCtrAbs}$ in line 5 computes the close to minimum full $Q$ control abstraction $\hat{W}$ of $\mathcal{H}$ (see Sect. 6.5 for further details about $\text{minFullCtrAbs}$). Line 6 checks if a weak mgo for the LTS control problem $\hat{P}' = (\hat{W}, \hat{I}, \hat{G})$ exists by calling function $\text{existsWeakCtrl}$, which is based on the algorithm in [43] (details are in App. A.3).

If function $\text{existsWeakCtrl}$ returns False, then a weak mgo to the LTS control problem $\hat{P}'$ does not exist, and by Prop. 2 no weak solution exists to $\hat{P}'$. By Theor. 8 (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 control led region $\hat{D}$.

### 6.2 Synthesis Algorithm Correctness

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

**Theorem 9.** Let $\mathcal{H}$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization, and $(\mathcal{H}, I, G)$ be a DTLHS control problem. Then $qCtrSyn(\mathcal{H}, \mathcal{Q}, I, G)$ returns a triple $(\mu, \hat{D}, \hat{K})$ such that: $\mu \in \{\text{Sol}, \text{NoSol}, \text{Unk}\}$, $\hat{D} = \text{Dom}(\hat{K})$ and $\hat{K}$, defined as $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $Q$ QFC solution to the control problem $(\mathcal{H}, \Gamma^{-1}(\hat{D}), G)$. Furthermore, the following holds.

1. If $\mu = \text{Sol}$ then $I \subseteq \Gamma^{-1}(\hat{D})$ and $K$ is a $Q$ QFC solution to the control problem $(\mathcal{H}, I, G)$.

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

**Proof.** If function $qCtrSyn$ returns (Sol, $\hat{D}$, $\hat{K}$), then function $\text{minCtrAbs}$ has found an admissible $Q$ control abstraction $\hat{M}$ of $\mathcal{H}$ (see Prop. 10) and function $\text{strongCtr}$ has found the strong mgo $\hat{K}$ to the control problem $(\hat{M}, \Gamma(I), \Gamma(G))$. By Theor. 8 (point 5) the controller $K$, defined by $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $Q$ QFC strong solution to the control problem $(\mathcal{H}, I, G)$.

If function $qCtrSyn$ returns (NoSol, $\hat{D}$, $\hat{K}$), there is no weak solution to the control problem $(\hat{W}, \Gamma(I), \Gamma(G))$, where $W$ is the close to minimum full...
control abstraction of $\mathcal{H}$ computed by function $\text{minFullCtrAbs}$ (Prop. 10). Therefore, by Theor. 8 (point 3) there is no $Q$ QFC solution to the control problem $(\mathcal{H}, I, G)$.

Note that if $\mu = \text{Unk}$ then function $qCtrSyn$ is inconclusive, that is $(\mathcal{H}, I, G)$ may or may not have a $Q$ QFC solution. This case stems from undecidability of the QFC problem (Theor. 3).

Remark 2. We note that function $\text{strongCtr}$ (see Alg. 4) returns a (worst case) time optimal controller, that is a controller that in each state enables the actions leading to a goal state in the least number of transitions. However the controllers we generate may not be time optimal for the real plant. In fact, self loops elimination shrinks all concrete sequences of the form $x_n, u_n, x_{n+1}, u_{n+1}, \ldots, x_{m-1}, u_{m-1}, x_m$ in every path of $\text{LTS}(\mathcal{H})$ into a single abstract transition $(\Gamma(x_n), \Gamma(u_n), \Gamma(x_m))$ of $\hat{\mathcal{M}}$ whenever $\Gamma(x_n) = \Gamma(x_{n+1}) = \ldots = \Gamma(x_{m-1})$ and $\Gamma(u_n) = \Gamma(u_{n+1}) = \ldots = \Gamma(u_{m-1})$. This leads to mismatches between the length of paths in the plant model and those in the control abstraction used for the synthesis. 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 $\hat{u}$ because of an eliminable self loop $(\hat{x}, \hat{u}, \hat{x})$ on a non-goal state $\hat{x}$. For these reasons we refer to our controller as a near time optimal controller.

6.3 Quantization

As usual in the following $\mathcal{H} = (X, U, Y, N)$ is a DTLHS, $Q = (A, \Gamma)$ is a quantization for $\mathcal{H}$, and $(\mathcal{H}, I, G)$ is a DTLHS control problem. The control abstraction to be built is $\mathcal{H} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$.

In our algorithms, we consider $\Gamma$ only in equality tests of type $(\Gamma(W) = \hat{v}) \equiv \bigwedge_{i \in |W|} (\gamma_{w_i}(w_i) = \hat{v}_i)$, where $W = [w_1, \ldots, w_{|W|}]$ may be $X, X'$ or $U$. More in detail, in our algorithms we have to solve problems of type $P(W) \equiv (\max, J(W), L(W) \land (\Gamma(W) = \hat{v}))$, being $J(W)$ a linear expression and $L(W)$ a conjunctive predicate. If also $(\Gamma(W) = \hat{v})$ is a conjunctive predicate, we have that $P(W)$ is a MILP problem, thus allowing us to use a MILP solver on $P(W)$. To this aim, we restrict ourselves to quantization functions $\gamma_{w_i}$ for which equality tests can be represented by using conjunctive
predicates. For \( w \in X \cup U \), a typical example is 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.

**Remark 3.** Note that, strictly speaking, the conjunctive predicate \( P_{\gamma_w}(w, \hat{z}) \) represents a relaxation of \( \gamma_w(w) = \hat{z} \). In fact, for all \( k \in [1, \Delta_w - 1] \) we have that \( P_{\gamma_w}(\inf A_w + \delta_w k, k - 1) \land P_{\gamma_w}(\inf A_w + \delta_w k, k) \). This may introduce spurious transitions: such transitions may increase nondeterminism in the control abstraction, but do not affect soundness of our algorithm.

We may now explain how \( \hat{I}, \hat{G} \) are effectively computed in line 1 of Alg. 1. Since the initial region \( I \) is represented as a conjunctive predicate, its quantization \( \hat{I} \) is computed by solving \( |\Gamma(A_X)| \) feasibility problems. More precisely, \( \hat{I} = \{ \hat{x} | \text{feasible}(I(X) \land \Gamma(X) = \hat{x}) \} \). 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:** A DTLHS \( H = (X, U, Y, N) \) and a quantization \( Q = (A, \Gamma) \).

**function** \( \text{minCtrAbs} \) (\( H, Q \))

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

**6.4 Computing Minimum Control Abstractions**

In this section, we present in Alg. 2 function \( \text{minCtrAbs} \), which effectively computes a close to minimum \( Q \) control abstraction for a given DTLHS.

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

For any pair \((\hat{x}, \hat{u})\) in \(\Gamma(A_X) \times \Gamma(A_U)\) line 4 checks if \(\hat{u}\) is \(\mathcal{Q}\)-admissible in \(\hat{x}\). This check is carried out by determining if the predicate \(P(X, U, Y, X', \hat{x}, \hat{u}) \equiv N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land X' \not\in A_X\) is not feasible. If \(\hat{u}\) is not \(\mathcal{Q}\)-admissible in \(\hat{x}\) (i.e., if \(P(X, U, Y, X', \hat{x}, \hat{u})\) is feasible), no transition of the form \((\hat{x}, \hat{u}, \hat{x}')\) is added to \(\hat{N}\). Note that \(X' \not\in A_X\) is not a conjunctive predicate, thus feasibility of predicate \(P(X, U, Y, X', \hat{x}, \hat{u})\) cannot be directly checked via function \textit{feasible}. We implement such a check by calling \(2|X|\) times function \textit{feasible} in the following way. For each \(x' \in X'\), let \(P_x^-(X, U, Y, X', \hat{x}, \hat{u}) \equiv N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land x' \leq \inf A_x\) and \(P_x^+(X, U, Y, X', \hat{x}, \hat{u}) \equiv N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land x' \geq \sup A_x\). For each \(x' \in X'\), we call function \textit{feasible} on \(P_x^+\) and \(P_x^-\) separately. If all such \(2|X|\) calls return \texttt{FALSE}, then \(P\) is not feasible, otherwise \(P\) is feasible. Note that by Def. 13 we should also check that \(\forall x \in \Gamma^{-1}(\hat{x}) \forall u \in \Gamma^{-1}(\hat{u}) \exists x' \in D_X \exists y \in D_Y N(x, u, y, x')\). This cannot be checked via function \textit{feasible}. We therefore perform such a check by using a tool for quantifier elimination, namely Mjollnir [30].

If \(\hat{u}\) is \(\mathcal{Q}\)-admissible in \(\hat{x}\), line 5 checks if the self loop \((\hat{x}, \hat{u}, \hat{x})\) has to be added to \(\hat{N}\). To this aim, we employ a function \textit{selfLoop} which takes a (state, action) pair \((\hat{x}, \hat{u})\) and returns \texttt{FALSE} if, accordingly to Def. 13, the self loop \((\hat{x}, \hat{u}, \hat{x})\) is eliminateable (i.e., it need not to be in \(\hat{N}\)). Details about our \textit{gradient based} heuristic implemented in function \textit{selfLoop} are given in Sect. 6.6.

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

Finally, for each abstract state \(\hat{x}' \in \Gamma(O)\) 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 \textit{existsTrans} solves the MILP problem \(N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x}'\).

\textbf{Remark 4.} \textit{From the nested loops in lines \ref{line:feasible} \ref{line:selfLoop} \ref{line:overImg} we have that \texttt{minCtrAbs worst case runtime is }\mathcal{O}(|\Gamma(A_X)|^2|\Gamma(A_U)|). However, thanks to the heuris-}
tic implemented in function overImg, minCtrAbs typical runtime is about $O(|\Gamma(A_X)||\Gamma(A_U)|)$ as confirmed by our experimental results (see Sect. 8, Fig. 6). The same holds for function minFullCtrAbs (see Sect. 6.5).

**Remark 5.** Function minCtrAbs is explicit in the (abstract) states and actions of $\hat{\mathcal{H}}$ and symbolic with respect to the auxiliary variables (modes) in the transition relation $N$ of $\mathcal{H}$. As a result our approach will work well with systems with just a few state variables and many modes, our target here.

### 6.5 Computing Minimum Full Control Abstraction

Function minCtrAbs can be easily modified in order to compute the close to minimum full $Q$ control abstraction, thus obtaining function minFullCtrAbs called in Alg. 1 line 5.

More precisely, function minFullCtrAbs is obtained by removing the highlighted code (on grey background) from Alg. 2 namely the admissibility check in line 4. Correctness of functions minCtrAbs and minFullCtrAbs is stated by the following proposition (details are in App. A.8).

**Proposition 10.** Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS and $Q = (A, \Gamma)$ be a quantization for $\mathcal{H}$.

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

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

### 6.6 Self Loop Elimination

In order to exactly get the minimum control abstraction, function selfLoop should return TRUE iff the given self loop is non-eliminable. This entails checking condition 3 of Def. 13 namely: $P(\hat{x}, \hat{u}) \equiv \exists \pi \forall t \in \mathbb{N} \pi^{(S)}(t) \in \Gamma^{-1}(\hat{x}) \land \pi^{(A)}(t) \in \Gamma^{-1}(\hat{u})$, which is undecidable by Prop. 9. Function selfLoop, outlined in Alg. 3 checks a sufficient condition for self loop elimination that in practice turns out to be very effective. That is, function selfLoop returns FALSE when a self loop is eliminable (or there is not a concrete witness for it). On the other hand, if function selfLoop returns TRUE, then the self loop under consideration may be non-eliminable as well as eliminable. In a conservative way, we assume self loops for which function selfLoop returns
TRUE to be non-eliminable (i.e. they are added to the close to minimum control abstraction, see line 5 of Alg. 2).

**Algorithm 3** Self loop elimination

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

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

1. if $\neg\exists\text{Trans}(\hat{x}, \hat{u}, \hat{x})$ then return FALSE
2. for $i = 1$ to $|X|$ do
3. $w_i \leftarrow \text{optimalValue}(\min, x_i' - x_i, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})$
4. if $w_i > 0$ then return FALSE
5. $W_i \leftarrow \text{optimalValue}(\max, x_i' - x_i, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})$
6. if $W_i < 0$ then return FALSE
7. return TRUE

Function **selfLoop** in Alg. 3 works as follows. First of all it checks if there is a concrete witness for the self loop under consideration. If it is not the case, **selfLoop** returns TRUE (line 1). Otherwise, for each real variable $x_i$, it tries to establish if $x_i$ is either always increasing or always decreasing inside $\Gamma^{-1}(\hat{x})$ by performing actions in $\Gamma^{-1}(\hat{u})$.

The minimum (resp. maximum) variation $w_i$ (resp. $W_i$) of the variable $x_i$ caused by performing an action $u \in \Gamma^{-1}(\hat{u})$ in $\Gamma^{-1}(\hat{x})$ is computed by solving the MILP problem in line 3 (resp. 5). If for some $i$, $w_i$ is strictly positive, $x_i' - x_i$ is strictly positive and then $x_i$ is always increasing inside $\Gamma^{-1}(\hat{x})$ by performing actions in $\Gamma^{-1}(\hat{u})$. Since $\Gamma^{-1}(\hat{x})$ is a compact set, no Zeno-phenomena may arise and hence, executing actions in $\Gamma^{-1}(\hat{u})$, it is guaranteed that $\mathcal{H}$ will eventually leave the region $\Gamma^{-1}(\hat{x})$ (line 4). Similarly, if for some $i$, $W_i$ is strictly negative, $x_i' - x_i$ is strictly negative and then $x_i$ is always decreasing inside $\Gamma^{-1}(\hat{x})$ by performing actions in $\Gamma^{-1}(\hat{u})$ (line 6).

Correctness of function **selfLoop** follows from the considerations given above, and is stated by the following proposition (details are in App. A.7).

**Proposition 11.** Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$, $\hat{x} \in \Gamma(A_X)$, and $\hat{u} \in \Gamma(A_U)$. If the abstract self loop $(\hat{x}, \hat{u}, \hat{x})$ has a concrete witness and **selfLoop**($\mathcal{H}, \mathcal{Q}, \hat{x}, \hat{u}$) returns FALSE, then $(\hat{x}, \hat{u}, \hat{x})$ is an eliminable self loop.
Note that if $\text{selfLoop}(\hat{x}, \hat{u})$ returns $\text{true}$, $P(\hat{x}, \hat{u})$ may be false, thus by Prop. 11 $P(\hat{x}, \hat{u}) \rightarrow \text{selfLoop}(\hat{x}, \hat{u})$. This can also be formulated as follows: if $(\hat{x}, \hat{u}, \hat{x})$ is a non-eliminable self loop, then $\text{selfLoop}(\hat{x}, \hat{u})$ returns $\text{true}$ (but the vice versa does not hold). The heuristic implemented in function $\text{selfLoop}$ in practice leads to a control abstraction very close to the minimum one as shown by our experimental results (see Tab. 1 in Sect. 5).

7 Control Software Generation

In this section we describe how we synthesize the actual control software (C functions $\text{Control\_Law}$ and $\text{Controllable\_Region}$ in Sect. 4) and show how we compute its WCET. More details are given in [32, 31] (details are in App. C).

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. When multiple OBDDs are translated via $\text{obdd2c}$, sharing between such OBDDs may be taken into account by maintaining an hash table of already translated OBDD nodes.

Let $(\mu, \hat{D}, \hat{K})$ be the output of function $q\text{CtrSyn}$ in Alg. 1. We synthesize function $\text{Controllable\_Region}$ by computing $\text{obdd2c}(\hat{D})$.

Let $r$ (resp. $n$) be the number of bits used to represent plant actions (resp. states). Let $F : \mathbb{B}^n \rightarrow \mathbb{B}^r$ be any boolean function such that, for each quantized state $\hat{x}$, if $\hat{D}(\hat{x})$ holds then also $\hat{K}(\hat{x}, F(\hat{x}))$ holds. In other words, $F$ is a boolean function returning, for each quantized state $\hat{x}$ in the controllable region $\hat{D} = \text{Dom}(\hat{K})$, a quantized action $\hat{u}$ such that $\hat{K}(\hat{x}, \hat{u})$ holds. In a hardware synthesis setting, techniques to compute $F$ satisfying the above functional equation have been widely studied (e.g. see [9]). In our software synthesis setting we follow the approach presented in [43] to compute such an $F$. Let $F_i : \mathbb{B}^n \rightarrow \mathbb{B}$ be the boolean function computing the $i$-th bit of $F$. That is, $F(\hat{x}) = [F_1(\hat{x}), \ldots, F_r(\hat{x})]$. Then, we take function $\text{Control\_Law}$ to be (the C implementation of) $[\text{obdd2c}(F_1), \ldots, \text{obdd2c}(F_r)]$. 

32
7.1 Control Software WCET

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

8 Experimental Results

We implemented our QFC synthesis algorithm in C programming language, using GLPK to solve MILP problems and the CUDD package for OBDD based computations. We name the resulting tool QKS (Quantized feedback Kontrol Synthesizer).

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). Note that control software reaction time (WCET) is known a priori from Sect. 7.1 and its robustness to parameter variations in the controlled system as well as enforcement of safety bounds on state variables are an input to our synthesis algorithm (see Ex. 1 and Sect. 8.1).
8.1 Experimental Settings

We present experimental results obtained by using QKS on a version of the buck DC-DC converter described in Sect. 3.1. We denote with $\mathcal{H} = (X, U, \bar{Y}, \bar{N})$ the DTLHS modeling such a converter, where $X, U$ are as in Sect. 3.1 while $\bar{Y}$ and $\bar{N}$ extends $Y$ and $N$ given in Sect. 3.1 as described in the following. $\bar{N}$ is the conjunction of Eqs. (3)–(11) of Sect. 3.1 and other additional contraints defined in the following.

We set the parameters of $\mathcal{H}$ as follows:

\[
\begin{align*}
\tau_L &= 0.1 \Omega \\
L &= 2 \cdot 10^{-4} \text{H} \\
T &= 10^{-6} \text{secs} \\
R &= 5 \pm 25\% \Omega \\
\tau_C &= 0.1 \Omega \\
C &= 5 \cdot 10^{-5} \text{F} \\
V_i &= 15 \pm 25\% \text{V}
\end{align*}
\]

and require our controller to be robust to foreseen variations (25%) in the load ($R$) and in the power supply ($V_i$). Variations in the power supply are modeled by adding Eqs. (13) and (14) to $\bar{N}$ (note that such constraints replace Eq. (12) in Sect. 3.1, see also Ex. 1):

\[
\begin{align*}
v_D \leq v_u - V_i (1 - \rho_{V_i}) & \quad (13) \\
v_D \geq v_u - V_i (1 + \rho_{V_i}) & \quad (14)
\end{align*}
\]

being the tolerance $\rho_{V_i} = 25\%$. Along the same lines, $\mathcal{H}$ models also variations in the load $R$. However, since $\mathcal{H}$ dynamics is not linear in $R$, much more work is needed [33]. 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 $\bar{Y}$, and 15 (guarded) constraints to be added to $\bar{N}$ (details are in App. E).

For converters, safety (as well as physical) considerations set requirements on admissible values for state variables (admissible regions). We set $A_{i_L} = [-4, 4]$ and $A_{v_O} = [-1, 7]$. Since the action variable $u$ is boolean, we have $A_u = \mathbb{B}$. We define $A = A_{i_L} \times A_{v_O} \times A_u$. 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 region $A$. As for auxiliary variables, we use the following safety bounds: $A_{i_u} = A_{i_D} = [-10^3, 10^3]$ and $A_{v_u} = A_{v_D} = [-10^7, 10^7]$. As a result, we add 12 further constraints to $\bar{N}$ stating that $\bigwedge_{w \in \{i_L, v_O, i_u, i_D, v_u, v_D\}} w \in A_w$ [33] (details are in App. E).

Finally, the initial region $I$ and goal region $G$ are as in Ex. 6 thus the DTLHS control problem we consider is $P = (\mathcal{H}, I, G)$. Note that no (formally proved) robust control software is available for buck DC-DC converters.
We use a uniform quantization dividing the domain of each state variable \((i_L, v_O)\) into \(2^b\) equal intervals, where \(b\) is the number of bits used by AD conversion (thus w.r.t. Sect. 6.3 we have that \(\Delta_{i_L} = \Delta_{v_O} = 2^b\)). We call the resulting set of quantization functions \(\Gamma_b = \{\gamma_{i_L}, \gamma_{v_O}, \gamma_u\}\), where \(\gamma_u\) is the identity function. The resulting quantization is \(Q_b = (A, \Gamma_b)\). The quantization step for \(\Gamma_b\) is \(\|\Gamma_b\| = \|\gamma_{i_L}\| = \|\gamma_{v_O}\| = |A_{i_L}|/2^b = |A_{v_O}|/2^b = 2^{3-b}\). Since we have two quantized variables \((i_L, v_O)\) each one with \(b\) bits, the number of states in the control abstraction is exactly \(2^{2b}\). Note that the quantization \(Q\) and the transition relation of \(H\) are s.t. \(\forall s \in \Gamma(A_X) \forall \hat{u} \in \Gamma(A_U)\) \(\forall s \in \Gamma^{-1}(\hat{s}) \forall u \in \Gamma^{-1}(\hat{u}) \exists s' \in D_X \exists y \in D_Y N(s, u, y, s')\). This allows us to skip the Mjollnir check described in Sect. 6.4.

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

### 8.2 QKS Performance

In this section we will show the performance (in terms of computation time and memory) of algorithms discussed in Sect. 6. Our MILP based technique requires that \(H\) is represented by conjunctive predicates. The DTLHS modelization \(H\) of the buck DC-DC converter given in Sect. 8.1 makes use of guarded predicates. The tranformation given in Prop. 1 requires that \(H\) is bounded, as it is indeed the case.

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

Columns labeled Controller Synthesis show the computation time (column CPU, in secs) for the generation of \(K_b\), and the size of its OBDD rep-
| b  | CPU    | MEM    | Arcs   | MaxLoops | LoopFrac | CPU | |K| | CPU | MEM | μ  |
|----|--------|--------|--------|----------|----------|-----|---|---|-----|-----|---|
| 8  | 1.95e+03 | 4.41e+07 | 6.87e+05 | 2.55e+04 | 0.00333 | 2.10e-01 | 1.39e+02 | 1.96e+03 | 4.46e+07 | UNK |
| 9  | 9.55e+03 | 5.67e+07 | 3.91e+06 | 1.87e+04 | 0.00440 | 2.64e+01 | 3.24e+03 | 9.58e+03 | 7.19e+07 | SOL |
| 10 | 1.42e+05 | 8.47e+07 | 2.61e+07 | 2.09e+04 | 0.00781 | 7.36e+01 | 1.05e+04 | 1.42e+05 | 1.06e+08 | SOL |
| 11 | 8.76e+05 | 1.11e+08 | 2.15e+08 | 2.26e+04 | 0.01435 | 2.94e+02 | 2.88e+04 | 8.76e+05 | 2.47e+08 | SOL |
From Tab. 1 we see that computing control abstractions (i.e., Alg. 2) is the most expensive operation in QKS and that thanks to function \textit{SelfLoop} \( \hat{\mathcal{M}}_b \) contains no more than 2% of the loops in \( \hat{\mathcal{H}}_b \).

8.2.1 MILP problems Analysis

For each MILP problem solved in QKS, Tab. 2 shows (as a function of \( b \)) the total and the average CPU time (in seconds) spent solving MILP problems, together with the number of MILP problems solved, divided by different kinds of MILP problems as follows. MILP1 refers to the MILP problems described in Sect. 6.3 i.e. those computing the quantization for \( I \) and \( G \), MILP2 refers to MILP problems in function \textit{SelfLoop} (see Alg. 3), MILP3 refers to the MILP problems used in function \textit{overImg} (line 4 of Alg. 2), MILP4 refers to MILP problems used to check actions admissibility (line 5 of Alg. 2), and MILP5 refers to MILP problems used to check transitions witnesses (line 4 of Alg. 2). Columns in Tab. 2 have the following meaning: \textit{Num} is the number of times that the MILP problem of the given type is called, \textit{Time} is the total CPU time (in secs) needed to solve all the \textit{Num} instances of the MILP problem of the given type, and \textit{Avg} is the average CPU time (in secs), i.e. the ratio between columns \textit{Time} and \textit{Num} (details are in App. B). Fig. 6 graphically shows (as a function of \( b \)) the number of
| MILP | b = 8 | b = 9 | b = 10 | b = 11 |
|------|------|------|--------|--------|
|      | Num  | Avg  | Time   | Num  | Avg  | Time   | 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 | 1.0e+06 | 2.7e-04 | 2.8e+02 | 4.2e+06 | 2.3e-04 | 9.7e+02 |
| 2    | 4.0e+05 | 1.5e-03 | 3.3e+02 | 1.6e+06 | 1.4e-03 | 1.1e+03 | 6.4e+06 | 3.8e-03 | 1.3e+04 | 2.5e+07 | 3.3e-03 | 4.6e+04 |
| 3    | 2.3e+05 | 9.1e-04 | 2.1e+02 | 9.2e+05 | 9.2e-04 | 8.4e+02 | 3.7e+06 | 3.0e-03 | 1.1e+04 | 1.5e+07 | 2.6e-03 | 3.8e+04 |
| 4    | 7.8e+05 | 9.9e-04 | 7.7e+02 | 4.4e+06 | 1.0e-03 | 4.5e+03 | 3.0e+07 | 2.6e-03 | 7.8e+04 | 2.6e+08 | 2.2e-03 | 5.7e+05 |
| 5    | 4.3e+05 | 2.8e-04 | 1.2e+02 | 1.7e+06 | 2.8e-04 | 4.9e+02 | 6.8e+06 | 1.8e-03 | 1.3e+04 | 2.7e+07 | 1.6e-03 | 4.2e+04 |
MILP4 instances solved (column *Num* of columns group MILP4 in Tab. 2).

From Tab. 2 column *Avg*, we see that the average time spent solving each MILP instance is small. Fig. 7 graphically shows that MILP average computation time does not heavily depend on *b*. As observed in Remark 4, Fig. 6 shows that the number of MILP4 invocations is much closer to $|\Gamma(A_X)||\Gamma(A_U)| = 2^{2b+1}$, rather than the theoretical worst case running time $|\Gamma(A_X)|^2|\Gamma(A_U)| = 2^{4b+1}$ of Alg. 2. This shows effectiveness of function over-Img heuristic.

**8.3 Controller Performance**

In this section we discuss the performance of the generated controller. Fig. 9 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.

```
int Controllable_R (int *x) { int ret_b = 0;
L_2af64a1: if (x[2] == 1) goto L_2b001e0;
else { ret_b = !ret_b; goto L_2afff40; }
L_21f95e0: return ret_b;
L_2b07f00: if (x[14] == 1) goto L_21f95e0;
else goto L_2b07ee0;
/* ... */
}
int Control_Law(int *x, int *a) {
    a[0] = Control_Law_Bits(x, 0); return 0;
}
int Control_Law_Bits(int *x, int b) { int ret_b;
    switch(b) {
        case 0: ret_b = 0; goto L_2af6081; }
L_2af6081: if (x[2] == 1) goto L_2a6d2e0;
else { ret_b = !ret_b; goto L_2af6060; }
L_21f95e0: return ret_b;
/* ... */
}
```

Figure 9: A snapshot of the synthesized control software for the Buck DC-DC converter with 10 bit AD conversion.

**8.3.1 Controllable Region**

One of the most important features of our approach is that it returns the guaranteed operational range (precondition) of the synthesized software (Theor. 9). This is the *controllable region* $\hat{D}$ returned by Alg. 1. In our case study, 9 bit turns out to be enough to have a controllable region that covers the initial
Increasing the number of bits, we obtain even larger controllable regions. Fig. 8 shows the controllable region $D_{10} = \Gamma_{10}^{-1}(\hat{D}_{10})$ for $K_{10}$ along with some trajectories (with time increasing counterclockwise) for the closed loop system. We see that the initial region $I \subseteq D_{10}$. Thus we know (on a formal ground) that 10 bit AD conversion suffices for our purposes.

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-Loop (HIL) simulation approach or, even better, following a formal approach, as discussed in [20, 25]. In our context HIL simulation is quite easy since we already have a DTLHS model for the plant and the control software is generated automatically.

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

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

The ripple measures the wideness of the oscillations around the goal (steady state) once this has been reached. Fig. 10(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 [42, 37]) and with the results available from the literature (e.g. [40, 43]).

9 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
Figure 10: Controller performances: setup time and ripple.

(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.

In order to speed-up the computation and to avoid possible numerical errors due to MILP solvers [38], a natural possible future research direction is to investigate fully symbolic control software synthesis algorithms based on efficient quantifier elimination procedures (e.g., see [36] and citations thereof).

References

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

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

[3] R. Alur and D. L. Dill. A theory of timed automata. Theor. Comput. Sci., 126:183–235, April 1994.
[4] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

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

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

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

[8] P. C. Attie, A. Arora, and E. A. Emerson. Synthesis of fault-tolerant concurrent programs. *ACM Trans. on Program. Lang. Syst.*, 26(1):125–185, 2004.

[9] D. Baneres, J. Cortadella, and M. Kishinevsky. A recursive paradigm to solve boolean relations. *IEEE Trans. Comput.*, 58:512–527, 2009.

[10] A. Bemporad. Hybrid Toolbox - User’s Guide, 2004. http://www.ing.unitn.it/~bemporad/hybrid/toolbox.

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

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

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

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

[15] G. Della Penna, D. Magazzeni, A. Tofani, B. Intrigila, I. Melatti, and E. Tronci. *Automated Generation of Optimal Controllers through Model Checking Techniques*, volume 15 of *Lecture Notes in Electrical Engineering*. Springer, 2008.
[16] A. Dominguez-Garcia and P. Krein. Integrating reliability into the design of fault-tolerant power electronics systems. In *PESC*, pages 2665–2671. IEEE, 2008.

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

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

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

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

[21] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond hytech: Hybrid systems analysis using interval numerical methods. In *HSCC*, LNCS 1790, pages 130–144, 2000.

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

[23] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What’s decidable about hybrid automata? *J. of Computer and System Sciences*, 57(1):94–124, 1998.

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

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

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

[27] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal: Status & developments. In *CAV*, LNCS 1254, pages 456–459, 1997.
[28] P.-Z. Lin, C.-F. Hsu, and T.-T. Lee. Type-2 fuzzy logic controller design for buck dc-dc converters. In *FUZZ*, pages 365–370, 2005.

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

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

[31] F. Mari, I. Melatti, I. Salvo, and E. Tronci. From boolean functional equations to control software. *CoRR*, abs/1106.0468, 2011.

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

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

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

[35] M. L. Minsky. Recursive unsolvability of post’s problem of "tag" and other topics in theory of turing machines. *The Annals of Mathematics*, 74(3):pp. 437–455, 1961.

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

[37] National semiconductor: http://www.national.com/analog/power/1m20xxx.

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

[39] G. Schrom, P. Hazucha, J. Hahn, D. Gardner, B. Bloechel, G. Dermer, S. Narendra, T. Karnik, and V. De. A 480-mhz, multi-phase interleaved buck dc-dc converter with hysteretic control. In *PESC*, pages 4702–4707 vol. 6. IEEE, 2004.
[40] W.-C. So, C. Tse, and Y.-S. Lee. Development of a fuzzy logic controller for dc/dc converters: design, computer simulation, and experimental evaluation. *IEEE Trans. on Power Electronics*, 11(1):24–32, 1996.

[41] P. Tabuada and G. J. Pappas. Linear time logic control of linear systems. *IEEE Trans. on Automatic Control*, 2004.

[42] Slvp182: High accuracy synchronous buck dc-dc converter: http://focus.ti.com.cn/cn/lit/ug/slvu046/slvu046.pdf, 2001.

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

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

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

A.1 From Bounded Predicates to Conjunctive Predicates

In this section, we prove Prop. 1.

Proposition 1. For each bounded guarded predicate $P(X)$, it is possible to compute an equivalent bounded conjunctive predicate $Q(X)$.

Proof. Predicate $Q(X)$ is obtained from the guarded predicate $P(X)$ by replacing each guarded constraint $\varphi$ in $P(X)$ with an equivalent linear constraint $\varphi^*$. We construct such a linear constraint $\varphi^*$ as follows. Let $x \in X$. Since $P(X)$ is bounded there exist $m_x, M_x \in \mathbb{D}_x$ such that $P(X)$ implies $m_x \leq x \leq M_x$. Let $a$ be a real number and $x \in X$. We write $\sup(ax) [\inf(ax)]$ for $aM_x [aM_x]$ when $a \geq 0$ and for $am_x [am_x]$ when $a < 0$. Let $L(X) = \sum_{i=1}^n a_i x_i$ be a linear expression. We write $\sup(L(X))$ for $\sum_{i=1}^n \sup(a_i x_i)$ and $\inf(L(X))$ for $\sum_{i=1}^n \inf(a_i x_i)$. Let $\varphi$ be $z \rightarrow (L(X) \leq b)$. We pick $\varphi^*$ to be the linear constraint $(\sup(L(X)) - b)z + L(X) \leq \sup(L(X))$. If $z = 0$ we have $\varphi \equiv \varphi^*$ since $\varphi$ holds trivially and $\varphi^*$ reduces to $L(X) \leq \sup(L(X))$ that holds by construction. If $z = 1$ both $\varphi$ and $\varphi^*$ reduce to $L(X) \leq b$. Along the same line of reasoning, if $\varphi$ has form $\bar{z} \rightarrow (L(X) \leq b)$ we pick $\varphi^*$ to be $(b - \sup(L(X)))z + L(X) \leq b$.

A.2 Uniqueness of the Most General Optimal Controller

In this section, we prove Prop. 2.

Proposition 2. An LTS control problem $(S, \emptyset, G)$ has always an unique strong mgo $K^*$. Moreover, for all $I \subseteq S$, we have:

- if $I \subseteq \text{Dom}(K^*)$, then $K^*$ is the unique strong mgo for the control problem $(S, I, G)$;
- if $I \not\subseteq \text{Dom}(K^*)$, then the control problem $(S, I, G)$ has no strong solution.
Proof. Let $\mathcal{S} = (S, A, T)$ be an LTS, and let $(\mathcal{S}, I, G)$ be an LTS control problem. We define the sequences of sets $D_n$ and $F_n$ as follows:

\[
\begin{align*}
D_0 &= \emptyset \\
F_1 &= \{s \in S \mid \exists a \in A. a \in \text{Adm}(\mathcal{S}, s) \land \text{Img}(\mathcal{S}, s, a) \subseteq G\} \\
F_{n+1} &= \{s \in S \setminus D_n \mid \exists a \in A. a \in \text{Adm}(\mathcal{S}, s) \land \text{Img}(\mathcal{S}, s, a) \subseteq D_n\} \\
D_{n+1} &= D_n \cup F_{n+1}
\end{align*}
\]

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 \footnote{\textsuperscript{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. $F_i \cap F_j = \emptyset$ for all $i \neq j$. We have that if $s \in F_{n+1}$ then $s \notin D_n$. By previous point \footnote{\textsuperscript{2}} we have that $s \notin D_n$ implies $s \notin F_j$ for $1 \leq j \leq n$. Hence, $s \in F_{n+1}$ implies that $s \notin F_j$ for all $1 \leq j \leq n$. If by absurd a state $s$ exists s.t. $s \in F_i \cap F_j$ for some $i > j$, then $s \in F_i$ would imply $s \notin F_j$.

For all $s \in S$ and $a \in A$, we define the controller $\tilde{K} : S \times A \to \mathbb{B}$ as follows:

\[
\tilde{K}(s, a) \iff (\exists n > 1. s \in F_n \land a \in \text{Adm}(\mathcal{S}, s) \land \text{Img}(\mathcal{S}, s, a) \subseteq D_{n-1}) \land (s \in F_1 \land a \in \text{Adm}(\mathcal{S}, s) \land \text{Img}(\mathcal{S}, s, a) \subseteq G)
\]

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

$\tilde{K}$ is a strong solution to $(\mathcal{S}, \emptyset, G)$. To prove this, we show that, if $t \in F_n$, then $J_{\text{strong}}(\mathcal{S}^{\tilde{K}}, G, t) = n$ (note that $t \in \text{Dom}(\tilde{K})$ implies
Moreover, there exists a path \( \pi \). By property 3 above, this implies that there exists \( 1_{\pi} \) and \( \pi \in F \) all \( n \) s.t. \( \emptyset \neq \text{Img}(S, t, a) \subseteq G \) = \( \sup\{ \text{sup}\{s.t. \prod(\emptyset \neq \text{Img}(S, t, a) \subseteq G) \} = \sup\{ \min\{n \mid n > 0 \land \pi(S(n) \in G) \} \mid \pi \in \{ \pi \in \text{Path}(S(K), t, a) \mid a \text{ s.t. } \emptyset \neq \text{Img}(S, t, a) \subseteq G\} \}. \) Since for all \( \pi \in \{ \pi \in \text{Path}(S(K), t, a) \mid a \text{ s.t. } \emptyset \neq \text{Img}(S, t, a) \subseteq G\} \) we have that \( \pi(S)(1) \in G \), we finally have that \( J_{\text{strong}}(S(K), G, t) = \sup\{1\} = 1 \). On the other hand, if \( t \in F_n \) then \( J_{\text{strong}}(S(K), G, t) = \sup\{ \min\{n \mid n > 0 \land \pi(S(n) \in G) \} \mid \pi \in \{ \pi \in \text{Path}(S(K), t, a) \mid a \text{ s.t. } \emptyset \neq \text{Img}(S, t, a) \subseteq D_{n-1}\} \} = \sup\{n_1, \ldots, n_j, \ldots\} \). We have that, for all \( j, n_j \leq n \). In fact, being \( t \in F_n \) and \( a \text{ s.t. } \emptyset \neq \text{Img}(S, t, a) \subseteq D_{n-1} \), we have that \( \pi(S)(1) \in D_{n-1} \) for all paths \( \pi \in \text{Path}(S(K), t, a) \). This implies that \( \pi(S)(1) \in D_{n-2} \lor \pi(S)(1) \in F_{n-1} \). By property 8 above, this implies that there exists \( 1 \leq i \leq n - 1 \) s.t. \( \pi(S)(i) \notin G \). Suppose by absurd that for all paths \( \pi \in \text{Path}(S(K), t, a) \) we have that, if for all \( 0 < i < n \pi(S)(i) \notin G \), then \( \pi(S)(n) \notin G \). By using an iterative reasoning as above, it is possible to show that this contradicts \( t \) being in \( F_n \) and \( a \) being s.t. \( \emptyset \neq \text{Img}(S, t, a) \subseteq D_{n-1} \). Thus, being \( n_j \leq n \) for all \( j \) and existing a \( j \) s.t. \( n_j = n \), we have that \( J_{\text{strong}}(S(K), G, t) = \sup\{n_1, \ldots, n_j, \ldots\} = n \).

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

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

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

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

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

If $n = 1$ we have that $\text{Img}(S, z_0, a) \subseteq G$ 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 $s \in \text{Img}(S, z_0, a)$, $K(s, u)$ implies $\bar{K}(s, u)$. This implies that $\text{Img}(S, z_0, a) \in D_{n-1}$ and thus $\bar{K}(z_0, a)$ holds.

\[\square\]

### A.3 LTS controller synthesis

Symbolic (OBDD based) control software synthesis algorithms for finite state deterministic LTSs have been studied in [43] and citations thereof. In such a context of course strong and weak solutions are the same. Symbolic (OBDD based) control synthesis algorithms for finite state nondeterministic LTSs have been studied in [14] in a universal planning setting. In such a context strong and weak solutions in general differ.

To compute strong solutions, we implemented a variant of the algorithm in [14] in function $\text{strongCtr}$. In our variant, a strong controller for the given LTS control problem is always returned, even if it is not possible to entirely control the given initial region (see Sect. 6.1). More precisely, it returns the strong mgo (see Def. 4), i.e. the unique strong solution $K$ to a control problem $(S, I, G)$ that, disallowing as few actions as possible, drives as many states as possible to a state in $G$ along a shortest path. For the sake of completeness, we show the resulting algorithm in Alg. 4.

Analogously, function $\text{existsWeakCtr}$ exploits the algorithm in [43] to verify the existence of weak solutions. Function $\text{existsWeakCtr}$ is shown in Alg. 5.

Correctness of function $\text{strongCtr}$ in Alg. 4 is proved in Prop. 12.

**Proposition 12.** Let $S = (S, A, T)$ be an LTS and $\mathcal{P} = (S, I, G)$ be an LTS control problem. Then, $\text{strongCtr}(S, I, G)$ returns $\langle b, D, K \rangle$ s.t. $K$ is the strong mgo for $(S, \emptyset, G)$, $D = \text{Dom}(K)$ and $b$ is True iff $K$ is the strong mgo for $\mathcal{P}$. 49
Algorithm 4 Building a strong mgo for an LTS control problem

Input: An LTS control problem \((S, I, G), S = (S, A, T)\).

1. \texttt{strongCtr}(S, I, G)
2. \(K(s, a) \leftarrow 0, D(s) \leftarrow G(s), \bar{D}(s) \leftarrow 0\)
3. \textbf{while} \(D(s) \neq \bar{D}(s)\) \textbf{do}
4. \(F(s, a) \leftarrow \exists s' T(s, a, s') \wedge \forall s' [T(s, a, s') \Rightarrow D(s')]\)
5. \(K(s, a) \leftarrow K(s, a) \lor \left( F(s, a) \land \neg \exists a K(s, a) \right)\)
6. \(\bar{D}(s) \leftarrow D(s), D(s) \leftarrow D(s) \lor \exists a K(s, a)\)
7. return \(\langle \forall s [I(s) \Rightarrow \exists a K(s, a)], \exists a K(s, a), K(s, a) \rangle\)

Proof. We observe that during a generic iteration \(i\) the set of states \\{ \(s \mid \exists a F(s, a)\) \} is exactly the set of states \(F_i\) and \\{ \(s \mid \bar{D}(s)\) \} is exactly the set of states \(D_i\) considered in the proof of Prop. 2 in App. A.2. As a consequence, the thesis holds by the proof of Prop. 2.

Algorithm 5 Existence of LTS control problem weak solutions

Input: An LTS control problem \((S, I, G), S = (S, A, T)\).

1. \texttt{existsWeakCtr}(S, I, G)
2. \(K(s, a) \leftarrow 0, D(s) \leftarrow G(s), \bar{D}(s) \leftarrow 0\)
3. \textbf{while} \(D(s) \neq \bar{D}(s)\) \textbf{do}
4. \(F(s, a) \leftarrow \exists s' T(s, a, s') \wedge \forall s' [T(s, a, s') \Rightarrow D(s')]\)
5. \(K(s, a) \leftarrow K(s, a) \lor \left( F(s, a) \land \neg \exists a K(s, a) \right)\)
6. \(\bar{D}(s) \leftarrow D(s), D(s) \leftarrow D(s) \lor \exists a K(s, a)\)
7. \textbf{return} \texttt{TRUE}
8. \(\bar{D}(s) \leftarrow D(s), D(s) \leftarrow D(s) \lor \exists a K(s, a)\)
9. return \texttt{FALSE}

Correctness of function \texttt{existsWeakCtr} in Alg. 5 may be proved analogously to Prop. 12.

Proposition 13. Let \(S = (S, A, T)\) be an LTS and \(P = (S, I, G)\) be an LTS control problem. Then, \texttt{existsWeakCtr}(S, I, G) returns \texttt{TRUE} iff there exists a weak mgo for \(P\).
A.4 Undecidability of DTLHS Quantized Feedback Control Problem

In this section we prove the undecidability of the DTLHS quantized feedback control problem, i.e. that existence of QFC strong and weak solutions to a DTLHS quantized control problem is undecidable (Theor. 3). Along the same lines of similar undecidability proofs [23], we first show that a two-counter machine \( M \) can be encoded as a DTLHS \( H_M \) without inputs in such a way that \( M \) halts if and only if \( H_M \) reaches a goal region. This immediately implies that DTLHS reachability is undecidable. Since \( H_M \) has no controllable actions, existence of a weak controller is equivalent to a reachability problem, thus it is undecidable. For the same reason, actions enabled by any controller for \( H_M \) do not depend on real valued variables. As a consequence, a quantized weak control problem on \( H_M \) is equivalent to a DTLHS control problem on \( H_M \). Since weak solutions to deterministic LTS control problems are also strong solutions, and being \( H_M \) deterministic, existence of a strong solution to a DTLHS (quantized) control problem is undecidable.

A.4.1 Two-Counter Machines

A two-counter machine \([35]\) \( M \) consists of two counters that store unbounded natural numbers and a finite control that is a finite sequence of statements \( \langle 1 : \text{stmt}_1, \ldots, n : \text{stmt}_n \rangle \), where \( \text{stmt} ::= \text{inc}(i)\ k \mid \text{dec}(i)\ k \mid \text{beq}\ i\ k \mid \text{halt} \), with \( i \in \{0, 1\} \). Computations start from the statement labeled 1. The execution of \( j : \text{inc}(i)\ k \) increments the counter \( i \), and the execution of \( j : \text{dec}(i)\ k \) decrements the counter \( i \), leaving it unchanged if it is 0. In both cases, execution continues to the statement labeled \( k \). If the counter \( i \) is 0, the execution of \( j : \text{beq}\ i\ k \) causes a jump to the statement labeled \( k \). Otherwise, the statement labeled \( j + 1 \) will be executed. Finally, the execution stops if a \text{halt} statement is executed. The halting problem for two-counter machine is undecidable.

Lemma 14. For any two-counter machine \( M \), there exist a bounded and deterministic DTLHS \( H_M \), and two predicates \( I \) and \( G \) such that \( M \) halts if and only \( G \) is reachable from \( I \) in \( H_M \).

Proof. Let \( M \) be a two-counter machine and let \( H_M \) be the DTLHS \((X, U, Y, N)\), where \( X^r = \{x_0, x_1\}, X^d = \{l, g\}, \) and \( U = Y = \emptyset \). Since we are
dealing with bounded DTLHSs, we use two real variables $x_0$ and $x_1$ to encode values stored in counters. Each natural number $n$ is encoded by the rational number $\frac{1}{2^n}$. Variables $x_i$ are both bounded by the predicate $0 \leq x_i \leq 1$. A discrete variable $l$ stores the label of the statement currently under execution and it is bounded by $0 \leq l \leq n$, where $n$ is the number of statements in $M$ finite control. Finally, the boolean variable $g$ encodes termination of the computation of $M$. The transition relation $N$ encodes the execution of the control program. Let $U(X, X')$ be the predicate $\bigwedge_{x \in X} x' = x$. A program $(1 : stmt_1, \ldots, n : stmt_n)$ is encoded by the predicate $N = \bigwedge_{j=1}^n [j : stmt_j]$, where:

\begin{align*}
[j : \text{dec}(i) \ k] &\equiv (l \neq j) \lor ((x_i = 1) \lor (x'_i = 2x_i)) \land \\
&\quad \land ((x_i \neq 1) \lor (x'_i = 1)) \land \\
&\quad \land (l' = k) \land U(x_{1-i}, g))
\end{align*}

\begin{align*}
[j : \text{inc}(i) \ k] &\equiv (l \neq j) \lor ((x'_i = \frac{1}{2})) \land \\
&\quad \land (l' = k) \land U(x_{1-i}, g))
\end{align*}

\begin{align*}
[j : \text{beq} i \ k] &\equiv (l \neq j) \lor (((x_i \neq 1) \lor (l' = k)) \land \\
&\quad \land ((x_i = 1) \lor (l' = l + 1)) \land \\
&\quad \land U(x_{1-i}, g))
\end{align*}

\begin{align*}
[j : \text{halt}] &\equiv (l \neq j) \lor ((l' = j) \land \\
&\quad \land (g' = 1) \land U(x_0, x_1))
\end{align*}

We observe that we use negation as syntactic sugar to improve readability. Indeed, since $x_i$ can assume only values of the form $\frac{1}{2^n}$ for some $n \in \mathbb{N}$, the condition $x_i \neq 1$ can be replaced by the constraint $x_i \leq \frac{1}{2}$. Moreover, since $l$ is a discrete variable, the condition $l \neq j$ can be can be replaced by the predicate $(l \leq j - 1) \lor (l \geq j + 1)$.

It is possible to check that $N(\{l, \frac{1}{2^n}, \frac{1}{2^n}, g\}, \epsilon, \{l', \frac{1}{2^m}, \frac{1}{2^m}, g'\})$ if and only if after executing the statement labeled $l$ with $n$ and $m$ as counter values, $M$ will execute the statement labeled $l'$ with $n'$ and $m'$ as counter values. Moreover if $g = 0$, $g'$ will be 1 if and only if the statement labeled $l$ is an halt statement.

Let $I$ be the predicate $I \equiv l = 1 \land g = 0$ and $G$ the predicate $g = 1$. $G$ is reachable from $I$ in $\mathcal{H}_M$ if and only if the computation of $M$ terminates.

Finally, we have to show that $N$ can be written as a conjunctive predicate. Any predicate $P(X)$ can be written as an equivalent DNF $\bigvee_{i=1}^n \bigwedge_{j=1}^{m_i} C_{ij}(X)$, where $C_{ij}(X)$ are constraints. By introducing $n$ fresh boolean auxiliary variables $z_1, \ldots, z_n$ this is equivalent to $\bigwedge_{i=1}^n (z_i \rightarrow \bigwedge_{j=1}^{m_i} C_{ij}(X)) \land \sum_{i=1}^n z_i \geq 1$, which in turn is equivalent $\bigwedge_{i=1}^n \bigwedge_{j=1}^{m_i} (z_i \rightarrow C_{ij}(X)) \land \sum_{i=1}^n z_i \geq 1$. Being $N$
bounded, by Prop. 1 this can be transformed into a conjunctive predicate. For example:

\[
\llbracket j : \text{halt} \rrbracket \equiv (z_{j,1} \rightarrow (l \geq j + 1)) \land (z_{j,2} \rightarrow (l \leq j - 1)) \land \\
(z_{j,3} \rightarrow (l' = j)) \land (z_{j,3} \rightarrow (g' = 1)) \land \\
(z_{j,3} \rightarrow (x'_0 = x_0)) \land (z_{j,3} \rightarrow (x'_1 = x_1)) \land \\
\sum_{i=1}^{3} z_{j,i} \geq 1
\]

\[\square\]

Note that Lemma 14 immediately implies undecidability for the DTLHS reachability problem, i.e. it is undecidable if there exists a path from a DTLHS state in a region \( I \) to a DTLHS state in a region \( G \).

**Proposition 15.** Existence of strong and weak solutions to a bounded DTLHS control problem is undecidable.

**Proof.** For any two-counter machine \( M \), the DTLHS \( H_M \) has no controllable actions. Let \( K \) be the controller that enables all actions, i.e. such that \( \forall x \in D_X K(x, \epsilon) \) holds. \( K \) is a weak solution to the control problem \((H_M, I, G)\) if and only if \( G \) is reachable from \( I \) (observe that states in \( G \) are controlled by \( K \)). Moreover, since the transition relation of \( H_M \) is deterministic, \( K \) is a weak solution to \((H_M, I, G)\) if and only if it is a strong solution. \( \square \)

**Theorem 3.** The DTLHS quantized control problem is undecidable.

**Proof.** The controller \( K \) considered in the proof of Theorem 15 is a quantized controller. Indeed, for any quantization \( Q = (A, \Gamma) \), let \( \hat{K} \) be the defined by \( \forall \hat{x} \in \Gamma(A_X) \hat{K}(\hat{x}, \epsilon) \). We have that \( K(x, \epsilon) = \hat{K}(\Gamma(x), \epsilon) \). \( \square \)

### A.5 Undecidability of Self Loops Eliminability

In this section we prove Prop. 4. In order to do this, we introduce non-deterministic two-counter machines \( \text{NDTCM} \) (NDTCM in the following). W.r.t. the deterministic definition given in Sect. 4 we modify \text{inc} and \text{dec} statement as follows: \text{inc}(i) \( k_1 k_2 \) \text{dec}(i) \( k_1 k_2 \) increments [decrements] counter \( i \) and then nondeterministically continues its execution at label \( k_1 \) or \( k_2 \). A run \( \rho = (l_0, v_{00}, v_{10}), \ldots, (l_i, v_{0i}, v_{1i}), \ldots \) on a NDTCM \( M \) with finite control \( \{1 : \text{stmt}_1, \ldots, n : \text{stmt}_n\} \) is a sequence of configurations \((l_i, v_{0i}, v_{1i})\) s.t. \( l_0 = 1 \) and:
• for all $i \geq 0$, $1 \leq l_i \leq n$ is a label and $v_{ji}$ is the value of counter $j$;
• for all $i \geq 0$, $(l_{i+1}, v_{0,i+1}, v_{1,i+1})$ is a possible result of executing statement at label $l_i$ with register values $v_{0i}, v_{1i}$. Namely:
  - if the statement at label $l_i$ is inc$(j)$ $k_1 k_2$ [dec$(j)$ $k_1 k_2$] then $l_{i+1} \in \{k_1, k_2\}$, $v_{j,i+1} = v_{ji}$ + 1 [$v_{j,i+1} = \max\{v_{ji} - 1, 0\}$] and $v_{0,j,i+1} = v_{0,j,i}$;
  - if the statement at label $l_i$ is beq $j$ $k$ then $l_{i+1} = k$ if $v_{ji} = 0$ and is $l_i + 1$ otherwise, and $v_{b,i+1} = v_{bi}$ for $b = 0, 1$;
  - if the statement at label $l_i$ is halt then $\rho = (l_0, v_{00}, v_{10}), \ldots, (l_i, v_{0i}, v_{1i})$ has finite length $i$.

Note that a (deterministic) two-counter machine as defined in Sect. 4 is a NDTCM where, for all statements inc$(i)$ $k_1 k_2$, we have $k_1 = k_2$ (and analogously for dec statements). Thus, from undecidability of two-counter machines halting problem, it is easy to show that it is undecidable whether there exists an infinite run $\rho = (l_0, v_{00}, v_{10}), \ldots, (l_i, v_{0i}, v_{1i}), \ldots$ on a NDTCM.

**Proposition 4.** Given a DTLHS $H$ and a quantization $Q$, it is undecidable to state if a self loop is non-eliminable.

**Proof.** Let $M$ be a NDTCM. We encode $M$ in a DTLHS $H_M = (X, U, Y, N)$, where $X_r = \{x_0, x_1, l\}$, $X^d = \{g\}$, and $U = Y = \emptyset$. $N = (\bigvee_{j=1}^{n} l = j) \land (\bigvee_{j=1}^{n} l' = j) \land \bigwedge_{j=1}^{n} \{j : \text{stmt}_j\}$ is defined as in proof of Lemma 14 with the following differences:

$$\llbracket j : \text{dec}(i) \ k_1 k_2 \rrbracket \equiv (l \neq j) \lor ((x_i = 1) \lor (x'_i = 2x_i)) \land (l' = k_1 \lor l' = k_2) \land U(x_{1-i}, g))$$

$$\llbracket j : \text{inc}(i) \ k_1 k_2 \rrbracket \equiv (l \neq j) \lor ((x'_i = \frac{a_j}{2}) \land (l' = k_1 \lor l' = k_2) \land U(x_{1-i}, g))$$

Let $Q = (A, \Gamma)$ be the quantization defined as follows: $A_{x_0} = A_{x_1} = [0, 1], A_l = [1, n], A_g = \mathbb{B} = \{0, 1\}, A_U = \{0\}, \gamma_{x_0}(x) = \gamma_{x_1}(x) = \gamma_l(x) = 1$. Note that we have only two abstract states: $\langle \hat{x}_0, \hat{x}_1, \hat{l}, g \rangle = \langle 1, 1, 1, 0 \rangle$ and $\langle \hat{x}_0, \hat{x}_1, \hat{l}, g \rangle = \langle 1, 1, 1, 1 \rangle$. Then, the self loop $(\langle 1, 1, 1, 0 \rangle, 0, \langle 1, 1, 1, 0 \rangle)$ is non-eliminable iff there exists an infinite run on $M$. Being the latter an undecidable problem, we cannot decide if a self loop is eliminable or non-eliminable. \[\square\]
A.6 Control Abstraction Properties

In this section we give proofs about control abstraction properties.

**Fact 5.** Let $M_1 = (S, B, T_1)$ and $M_2 = (S, B, T_2)$ be two admissible $Q$ control abstractions of a DTLHS $H$, with $Q = (A, \Gamma)$ quantization for $H$. Then $\forall \hat{x}, \hat{x}' \in S \ s.t. \ \hat{x} \neq \hat{x}', \forall \hat{a} \in B \ [T_1(\hat{x}, \hat{a}, \hat{x}') \Leftrightarrow T_2(\hat{x}, \hat{a}, \hat{x}')]$. The same holds if $M_1, M_2$ 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 point 1 of Def. 14, that $\hat{a}$ is $A$-admissible in $\hat{x}$. From point 1 of Def. 13 (for the admissible control abstraction case) or Def. 14 of full control abstraction (for the full control abstraction case), and from $T_1(\hat{x}, \hat{a}, \hat{x}')$ follows that $\exists x \in \Gamma^{-1}(\hat{x}) \exists a \in \Gamma^{-1}(\hat{a}) \exists y N(x, a, y, x')$. By point 2 of Def. 13 this implies that $T_2(\hat{x}, \hat{a}, \hat{x}')$ holds.

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

**Fact 6.** 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 admissible [full] $Q$ control abstractions of $H$ $(C_a(H, Q), \sqsubseteq)$ $(C_f(H, Q), \sqsubseteq)$ is a lattice.

**Proof.** By conditions 2 and 3 of Def. 13 all control abstractions do contain all admissible actions that have a concrete witness and all non-eliminable self-loops.

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

Analogously, both $(C_a(H, Q), \sqsubseteq)$ and $(C_f(H, Q), \sqsubseteq)$ are isomorphic to the complete lattice $(2^S, \subseteq)$. □

**Theorem 8.** Let $H = (X, U, Y, N)$ be a DTLHS, $Q = (A, \Gamma)$ be a quantization for $H$, and $(H, I, G)$ be a control problem.

1. If $\hat{H}$ is an admissible $Q$ control abstraction and $\hat{K}$ is a strong solution to the LTS control problem $(\hat{H}, \Gamma(I), \Gamma(G))$ then $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ is a $Q$ QFC strong solution to the DTLHS control problem $(H, I, G)$. 

55
2. If $\hat{H}_1, \hat{H}_2$ are two admissible $Q$ control abstractions of $H$ s.t. $\hat{H}_1 \subseteq \hat{H}_2$, and $K$ is a strong solution to the LTS control problem ($\hat{H}_2, \Gamma(I), \Gamma(G)$), then $\hat{K}$ is a strong solution to the LTS control problem ($\hat{H}_1, \Gamma(I), \Gamma(G)$).

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

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

Proof. The idea underlying the proof is that two different admissible control abstractions, with the same quantization, have the same loop free structure, i.e. the same arcs except from self loops, as proved by Prop. $\text{[5]}$. For ease of notation, given a state $x$ (resp. an action $u$) we will often denote the corresponding abstract state $\Gamma(x)$ (resp. action $\Gamma(u)$) with $\hat{x}$ (resp. $\hat{u}$).

Analogously, we will often write $I$ (resp. $G$) for $\Gamma(I)$ (resp. $\Gamma(G)$). In the following, $P = (H, I, G)$, $\hat{P} = (\hat{H}, \Gamma(I), \Gamma(G))$, and $\hat{H} = (\Gamma(A_x), \Gamma(A_U), N)$.

Proof of point $\text{[1]}$. Applying the definition of solution to a DTLHS control problem (Def. $\text{[10]}$), 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 $K(x, u) = \hat{K}(\hat{x}, \hat{u})$ is a strong solution to the LTS control problem ($\text{LTS}(\hat{H}), I, \mathcal{B}_{|\Gamma|}(G)$).

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 $\text{[2]}$ of Def. $\text{[13]}$, this implies that, for any transition $(x, u, x')$ of $\text{LTS}(\hat{H})^{(K)}$ such that $\hat{x} \neq \hat{x}'$, $(\hat{x}, \hat{u}, \hat{x}')$ is a (abstract) transition of $\hat{H}^{(K)}$.

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

Now, we prove that for all $x \in \text{Dom}(K)$, $J_{\text{strong}}(\text{LTS}(\hat{H})^{(K)}, \mathcal{B}_{|\Gamma|}(G), x)$ is finite. Let us suppose by absurd that $J_{\text{strong}}(\text{LTS}(\hat{H})^{(K)}, \mathcal{B}_{|\Gamma|}(G), x) = \infty$. This implies that one of the two following holds:
1. there exists a finite fullpath \( \pi = x_0 u_0 x_1 u_1 \ldots x_n u_n \) in \( \text{LTS}(\mathcal{H})^{(K)} \) such that \( x_0 = x \), \( \text{Adm}(\text{LTS}(\mathcal{H})^{(K)} x_n) = \emptyset \) and, for all \( i \in [n] \), \( x_i \notin B_{\|\Gamma\|}(G) \);

2. there exists an infinite fullpath \( \pi = x_0 u_0 x_1 u_1 \ldots x_n u_n \ldots \) in \( \text{LTS}(\mathcal{H})^{(K)} \) such that \( x_0 = x \) and, for all \( i \in \mathbb{N} \), \( x_i \notin B_{\|\Gamma\|}(G) \).

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

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

By the fact (proved above) that if \( (x, u, x') \) is a transition of \( \text{LTS}(\mathcal{H})^{(K)} \) with \( \hat{x} \neq \hat{x}' \), then \( (\hat{x}, \hat{u}, \hat{x}') \) is a transition of \( \mathcal{H}^{(K)} \), we have that \( \rho \) is a run of \( \mathcal{H}^{(K)} \). Let \( m = |\rho| = \max_{i \in [n]} k(i) \). Since \( \hat{K} \) is a strong solution to \( \hat{P} \) and \( \hat{x} \in \text{Dom}(\hat{K}) \), we have that \( \hat{x}_m \in \text{Dom}(\hat{K}) \). This implies that there exists \( \hat{u} \in \Gamma(A_U) \) s.t. \( \hat{K}(\hat{x}_m, \hat{u}) \), thus that there exists \( \hat{u} \in \text{Adm}(\mathcal{H}^{(K)}, \hat{x}_m) \). Thus by property 2 of Def. 14 (and since \( x_n \in \Gamma^{-1}(\hat{x}_m) \)) we have that \( \text{Adm}(\text{LTS}(\mathcal{H})^{(K)}, x) \supseteq \Gamma^{-1}(\hat{u}) \neq \emptyset \), which implies that \( \pi \) cannot be a finite fullpath.

As for the infinite fullpath case (point 2 above), we observe that in \( \pi \) we cannot have an infinite sequence \( x_k u_k x_{k+1} u_{k+1} \ldots \) such that for all \( j \geq k \), \( \Gamma(x_j) = \Gamma(x_k) \) and \( \Gamma(u_j) = \Gamma(u_k) \). In fact, suppose by absurd that this is true, and let \( k \) be the least \( k \) for which this happens. Then \((\hat{x}_k, \hat{u}_k, \hat{x}_k)\) is a non-eliminable self loop. Since \( x_j \notin B_{\|\Gamma\|}(G) \) for all \( j \geq k \), and thus \( \hat{x}_j \notin \hat{G} \) for all \( j \geq k \), we also have that \( J_{\text{strong}}((\hat{H}^{(K)}), \hat{G}, \hat{x}_k) = \infty \). By applying the same reasoning used for the finite fullpath case, we have that there is a path in \( \hat{H}^{(K)} \) leading from \( \hat{x} \) to \( \hat{x}_k \), which implies that \( J_{\text{strong}}((\hat{H}^{(K)}), \hat{G}, \hat{x}) = \infty \). Finally, this contradicts the fact that \( \hat{K} \) is a strong solution to \( \hat{P} \) and \( \hat{x} \in \text{Dom}(\hat{K}) \).

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 \( \text{LTS}(\mathcal{H})^{(K)} \) with \( x_0 = x \), we can extract an infinite abstract fullpath \( \rho \) s.t.
\( \rho(i) = \Gamma(\pi^{(S)}(k(i))) \), where the function \( k : \mathbb{N} \rightarrow \mathbb{N} \) is recursively defined as follows:

- \( k(0) = 0 \)
- \( k(i+1) = \min\{j \mid k(i) < j \land \Gamma(x_j) \neq \Gamma(x_{k(i)})\} \).

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

**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 admissible \( Q \) 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, 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. That is, there exists a transitions set \( B = \{(\hat{x}_1, \hat{u}_1, \hat{x}_1), \ldots, (\hat{x}_m, \hat{u}_m, \hat{x}_m)\} \) s.t. for all \((\hat{x}_i, \hat{u}_i, \hat{x}_i) \in B\) we have that \( T_1(\hat{x}_i, \hat{u}_i, \hat{x}_i) = 0 \land T_2(\hat{x}_i, \hat{u}_i, \hat{x}_i) = 1 \), and for all \((\hat{x}, \hat{u}, \hat{x}') \in \Gamma(A_X) \times \Gamma(A_U) \times \Gamma(A_X) \) we have that if \((\hat{x}, \hat{u}, \hat{x}') \notin B \) then \( T_1(\hat{x}, \hat{u}, \hat{x}') = T_2(\hat{x}, \hat{u}, \hat{x}') \). Let \( \hat{K} \) be the strong mgo to the \( \text{LTS} \) control problem \((\hat{H}_2, \hat{I}, \hat{G})\) and let \((\hat{x}_i, \hat{u}_i, \hat{x}_i) \in B \).

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

We are now ready to prove the thesis. Since we already know that \( \hat{I} \subseteq \text{Dom}(\hat{K}) \), 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 \( \hat{u} \in \text{Adm}(\hat{H}_1, \hat{x}) \) (Def. 3). 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 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 $J_{\text{strong}}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$. This can be proved by induction on the value of $J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$.

Suppose $J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x}) = 1$. Then, $\emptyset \neq \text{Img}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{x}, \hat{u}) \subseteq \hat{G}$ for all $\hat{u}$ s.t. $\hat{K}(\hat{x}, \hat{u})$. If for all $\hat{u}$ s.t. $\hat{K}(\hat{x}, \hat{u})$ there exists a state $\hat{x}' \neq \hat{x}$ s.t. $\hat{x}' \in \text{Img}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{x}, \hat{u})$, then we have that $\hat{x}' \in \text{Img}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{x}, \hat{u})$ by Fact A and since $\emptyset \neq \text{Img}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{x}, \hat{u}) \subseteq \text{Img}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{x}, \hat{u}) \subseteq \hat{G}$ we have that $J_{\text{strong}}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{G}, \hat{x}) = 1 = J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$. Otherwise, let $\hat{u}$ be s.t. $\hat{K}(\hat{x}, \hat{u})$ and $T_{3}(\hat{x}, \hat{u}, \hat{x}') \rightarrow \hat{x}' = \hat{x}$. 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 $J_{\text{strong}}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{G}, \hat{x}) = 1 = J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$. The other case, i.e. $(\hat{x}, \hat{u}, \hat{x}) \in B$ is impossible since, by the reasoning above and being $\hat{x} \in \hat{G}$, it would imply that there exists a state $\hat{x}' \neq \hat{x}$ such that $T_{2}(\hat{x}, \hat{u}, \hat{x}')$.

Suppose now that for all $\hat{x}$ s.t. $J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x}) = n$,  

$J_{\text{strong}}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$. Let $\hat{x} \in \text{Dom}(\hat{K})$ be s.t. $J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x}) = n + 1$. If $(\hat{x}, \hat{u}, \hat{x}) \notin B$ for any $\hat{u}$, then $\text{Img}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{x}, \hat{u}) = \text{Img}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{x}, \hat{u})$ for all $\hat{u}$, thus $J_{\text{strong}}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$ by induction hypothesis. Otherwise, let $(\hat{x}, \hat{u}, \hat{x}) \in B$ for some $\hat{u}$. By the reasoning above, if $\hat{x} \notin \hat{G}$ then $\hat{K}(\hat{x}, \hat{u}) = 0$, and again $J_{\text{strong}}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$ by induction hypothesis. If $\hat{x} \in \hat{G}$, then there exists a state $\hat{x}' \neq \hat{x}$ such that $T_{2}(\hat{x}, \hat{u}, \hat{x}')$ (and $T_{1}(\hat{x}, \hat{u}, \hat{x}')$). Being $J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x}) = n + 1$, we must have $J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x}') \leq n$, thus again $J_{\text{strong}}(\hat{\mathcal{H}}_{1}(\hat{K}), \hat{G}, \hat{x}) = J_{\text{strong}}(\hat{\mathcal{H}}_{2}(\hat{K}), \hat{G}, \hat{x})$ by induction hypothesis.

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

**Proof of point 3** Applying the definition of DTLHS control problem (Def. 101), we will show that if $K$ is a weak solution to the LTS control problem $(\text{LTS}(\mathcal{H}), I, B_{\|\|}(G))$, and $\hat{\mathcal{H}}$ is any full $Q$ control abstraction of $\mathcal{H}$.
then there exists a weak solution $\hat{K}$ to the control problem $(\hat{\mathcal{H}}, \hat{\mathcal{I}}, \hat{\mathcal{G}})$.

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 $\mathcal{H}$.

Let $\mathcal{H}$ be a full $Q$ control abstraction of $\mathcal{H}$. First of all, we show that $\hat{K}$ is a controller for $\mathcal{H}$ (Def. 3), i.e. that $\hat{K}(\hat{x}, \hat{u})$ implies $\hat{u} \in \text{Adm}(\mathcal{H}, \hat{x})$. Suppose $\hat{K}(\hat{x}, \hat{u})$ holds: this implies that there exist $x \in \Gamma^{-1}(\hat{x}), u \in \Gamma^{-1}(\hat{u})$ s.t. $K(x, u)$ and $u \in \text{Adm}(\mathcal{H}, x)$. If there exists $x' \in A_X$ s.t. $x' \in \text{Img}(\mathcal{H}, x, u)$ and $x' \neq \hat{x}$, then, being $\mathcal{H}$ a full $Q$ control abstraction of $\mathcal{H}$, we have that $(\hat{x}, \hat{u}, x')$ is a transition of $\mathcal{H}$, thus $\hat{u} \in \text{Adm}(\mathcal{H}, \hat{x})$. Otherwise, one of the following must hold:

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

We now have to prove that $\hat{K}$ is a weak solution to $\mathcal{H}$, being $\mathcal{H}$ a full $Q$ control abstraction of $\mathcal{H}$. First of all, we show that $\hat{I} \subseteq \text{Dom}(\hat{K})$. Given $\hat{x} \in \hat{I}$, we have that there exists $x \in \Gamma^{-1}(\hat{x})$ such that $x \in I$. Since $K$ is a weak solution to $\mathcal{P}$, there exists $u \in A_U$ s.t. $K(x, u)$, thus by definition of $\hat{K}$, $\hat{K}(\hat{x}, \hat{u})$ holds, and hence $\hat{x} \in \text{Dom}(\hat{K})$.

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

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

- let \( Z_z = \{ j \mid z < j \leq n \land \Gamma(x_j) \neq \Gamma(x_z) \} \)
- \( k(0) = 0 \)
- \( k(i + 1) = \begin{cases} k(i) & \text{if } Z_k(i) = \emptyset \\ \min Z_k(i) & \text{otherwise} \end{cases} \)

In a full \( Q \) control abstraction \( \hat{H} \), if \( (x, u, x') \) is 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 \( \hat{H}^{(K)} \) that leads from \( \hat{x}_0 = \hat{x} \) to the goal. As a consequence, \( K \) is a weak solution to \( P \).

**Proof of point 4** Analogously to the proof of point 2 let \( \hat{H}_1 = (\Gamma(A_1), \Gamma(A_U), T_1) \) and \( \hat{H}_2 = (\Gamma(A_2), \Gamma(A_U), T_2) \) 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, the only difference between \( \hat{H}_1 \) and \( \hat{H}_2 \) may be 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 \( K \) 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 \( I \subseteq \text{Dom}(K) \), we only have to prove that i) \( K \) is a controller for \( \hat{H}_2 \) and that ii) \( J_{\text{weak}}(\hat{H}_2^{(K)}, \hat{G}, \hat{x}) < \infty \) for all \( \hat{x} \in \text{Dom}(K) \).

As for the first point, we have to show that \( K(\hat{x}, \hat{u}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}_2, \hat{x}) \) (Def. 3). Since \( K(\hat{x}, \hat{u}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}_2, \hat{x}) \), and since \( \hat{u} \in \text{Adm}(\hat{H}_1, \hat{x}) \) implies \( \hat{u} \in \text{Adm}(\hat{H}_2, \hat{x}) \), this point is proved.

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

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

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

\[ \square \]

### A.7 Function selfLoop correctness

**Proposition 11.** Let \(\mathcal{H} = (X, U, Y, N)\) be a DTLHS, \(Q = (A, \Gamma)\) be a quantization for \(\mathcal{H}\), \(\hat{x} \in \Gamma(A_X)\), and \(\hat{u} \in \Gamma(A_U)\). If the abstract self loop \((\hat{x}, \hat{u}, \hat{x})\) has a concrete witness and selfLoop\((\mathcal{H}, Q, \hat{x}, \hat{u})\) returns False, then \((\hat{x}, \hat{u}, \hat{x})\) is an eliminable self loop.

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

For \(i \in [|[X]|]\), let \(w_i \leq W_i\) be the values computed in lines 3 and 6 of Alg. 3, i.e. \(w_i = \text{optimalValue}(\min, x_i - x_t, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})\) and \(W_i = \text{optimalValue}(\max, x_i - x_t, N(X, U, Y, X') \land \Gamma(X) = \hat{x} \land \Gamma(U) = \hat{u} \land \Gamma(X') = \hat{x})\).

Since selfLoop\((\mathcal{H}, Q, \hat{x}, \hat{u})\) returns False, there exists at least an index \(j \in [|[X]|]\) such that \(w_j > 0\) or \(W_j < 0\) (see lines 1 and 6 of Alg. 3 resp.). Let us consider the former case (note that \(w_j > 0\) implies \(W_j > 0\)).

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

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

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

\[ \square \]
A.8 Functions \textit{minCtrAbs} and \textit{minFullCtrAbs} correctness

Proposition 10. Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS and $Q = (A, \Gamma)$ be a quantization for $\mathcal{H}$.

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

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

Proof. Here we prove only the part regarding function \textit{minCtrAbs}, since the other part may be proved analogously. We first show that the control abstraction $\hat{\mathcal{H}} = (\Gamma(A_X), \Gamma(A_U), \hat{N})$ satisfies conditions 1–3 of Def. 13.

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

2. Let $x, s' \in A_X$ and $u \in A_U$ be such that $\exists y N(x, u, y, x')$ and $\Gamma(x) \neq \Gamma(x')$. Since \textit{minCtrAbs} examines all tuples in $\Gamma(A_X) \times \Gamma(A_U) \times \Gamma(A_X)$, it will eventually examine the tuple $(\hat{x}, \hat{u}, \hat{x}')$ s.t. $\hat{x} = \Gamma(x)$, $\hat{u} = \Gamma(u)$, and $\hat{x}' = \Gamma(x')$. If $\hat{u}$ is not $Q$-admissible in $\hat{x}$ no transition is added to $\hat{N}$ because of the check in line 4. Otherwise, since $\exists y N(x, u, y, x')$ holds, \textit{existsTrans}(\hat{x}, \hat{u}, \hat{x}') returns TRUE and the transition $(\hat{x}, \hat{u}, \hat{x}')$ is added to $\hat{N}$ in line 9 of Alg. 2.

3. Note that condition 3 of Def. 13 may be rephrased as follows: if $(\hat{x}, \hat{u}, \hat{x})$ is a non-eliminable self loop, then $\hat{N}(\hat{x}, \hat{u}, \hat{x})$ must hold. That is, if $\hat{N}(\hat{x}, \hat{u}, \hat{x}) = 0$ then either there is not a concrete witness for the self loop $(\hat{x}, \hat{u}, \hat{x})$, or $(\hat{x}, \hat{u}, \hat{x})$ is an eliminable self loop. This is exactly the case for which function \textit{selfLoop}($\mathcal{H}$, $Q$, $\hat{x}$, $\hat{u}$) returns FALSE (resp. by line 1 of Alg. 3 and by Prop. 11). Since a self loop $(\hat{x}, \hat{u}, \hat{x})$ is not added to $\hat{N}$ only if \textit{selfLoop}($\mathcal{H}$, $Q$, $\hat{x}$, $\hat{u}$) returns FALSE in line 5 of Alg. 2 and since function \textit{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.

\hfill $\square$
B  Details about the Experiments

In this section we give (Tab. 3) all details about MILP problems arising in our experiments of Sect. 8. Namely, in Tab. 3 MILP\textsubscript{i} has the same meaning as in Sect. 8.2.1 i.e. MILP1 refers to the MILP problems described in Sect. 6.3 i.e. those computing the quantization for \( I \) and \( G \), MILP2 refers to MILP problems in function \textit{SelfLoop} (see Alg. 3), MILP3 refers to the MILP problems used in function \textit{overImg} (line 6 of Alg. 2), MILP4 refers to MILP problems used to check actions admissibility (line 8 of Alg. 2), and MILP5 refers to MILP problems used to check transitions witnesses (line 4 of Alg. 2). In Tab. 3 columns \( b \), \( Num \), \( Avg \) and \( Tot \) are the same as columns \( b \), \( Num \), \( Avg \) and \( Time \) of Tab. 2 thus \( b \) shows the number of AD bits, \( Num \) is the number of times that the MILP problem of the given type is called, \( Tot \) is the total CPU time needed to solve all the \( Num \) instances of MILP problem of the given type, and \( Avg \) is the ratio between \( Tot \) and \( Num \). In Tab. 3 we also show in columns \( Min \) and \( Max \) the average, minimum and maximum time to solve one MILP problem of the given type. The standard deviation for such statistics is given in column \textit{DevStd}.

C  From Boolean Relations to Control Software

In this section we give more details about how we obtain our control software, starting from a strong mgo \( \hat{K}(\hat{x}, \hat{u}) \) (see Sect. 7). To this aim, we follow the exposition given in [32]. Further details are in [31].

C.1  Basic Definitions

In the following, we will denote boolean functions \( f : \mathbb{B}^n \to \mathbb{B} \) with boolean expressions on boolean variables involving + (logical OR), \( \cdot \) (logical AND, usually omitted thus \( xy = x \cdot y \)), - (logical complementation) and \( \oplus \) (logical XOR). We will also denote vectors of boolean variables in boldface, e.g. \( \vec{x} = \langle x_1, \ldots, x_n \rangle \). Moreover, we also denote with \( f|_{x_i=g}(\vec{x}) \) the boolean function \( f(x_1, \ldots, x_{i-1}, g(\vec{x}), x_{i+1}, \ldots, x_n) \) and with \( \exists x_i \ f(\vec{x}) \) the boolean function \( f|_{x_i=0}(\vec{x}) + f|_{x_i=1}(\vec{x}) \).
## Table 3: Complete statistics for Tab. 2 of Sect. 8

|       | Num  | Tot  | Avg   | Min   | Max    | Dev    | Std    |
|-------|------|------|-------|-------|--------|--------|--------|
| **MILP1** |      |      |       |       |        |        |        |
| 8     | 6.55e+04 | 4.61e+00 | 7.03e-05 | 0.00e+00 | 1.00e-02 | 8.35e-04 |
| 9     | 2.62e+05 | 1.84e+01 | 7.02e-05 | 0.00e+00 | 1.00e-02 | 8.35e-04 |
| 10    | 1.05e+06 | 2.79e+02 | 2.66e-04 | 0.00e+00 | 1.00e-02 | 1.61e-03 |
| 11    | 4.19e+06 | 9.65e+02 | 2.30e-04 | 0.00e+00 | 1.00e-02 | 1.50e-03 |
| **MILP2** |      |      |       |       |        |        |        |
| 8     | 3.99e+05 | 3.25e+02 | 1.52e-03 | 0.00e+00 | 1.00e-02 | 4.39e-03 |
| 9     | 1.59e+06 | 1.12e+03 | 1.41e-03 | 0.00e+00 | 1.00e-02 | 4.14e-03 |
| 10    | 6.36e+06 | 1.35e+04 | 3.78e-03 | 0.00e+00 | 1.00e-02 | 6.43e-03 |
| 11    | 2.54e+07 | 4.56e+04 | 3.26e-03 | 0.00e+00 | 1.00e-02 | 6.10e-03 |
| **MILP3** |      |      |       |       |        |        |        |
| 8     | 2.31e+05 | 2.10e+02 | 9.10e-04 | 0.00e+00 | 1.00e-02 | 3.20e-03 |
| 9     | 9.21e+05 | 8.44e+02 | 9.16e-04 | 0.00e+00 | 1.00e-02 | 3.18e-03 |
| 10    | 3.68e+06 | 1.11e+04 | 3.00e-03 | 0.00e+00 | 2.00e-02 | 4.26e-03 |
| 11    | 1.47e+07 | 3.76e+04 | 2.55e-03 | 0.00e+00 | 2.00e-02 | 4.02e-03 |
| **MILP4** |      |      |       |       |        |        |        |
| 8     | 7.80e+05 | 7.71e+02 | 9.89e-04 | 0.00e+00 | 1.00e-02 | 2.98e-03 |
| 9     | 4.42e+06 | 4.49e+03 | 1.02e-03 | 0.00e+00 | 1.00e-02 | 3.02e-03 |
| 10    | 3.01e+07 | 7.75e+04 | 2.58e-03 | 0.00e+00 | 2.00e-02 | 4.37e-03 |
| 11    | 2.61e+08 | 5.66e+05 | 2.17e-03 | 0.00e+00 | 2.00e-02 | 4.13e-03 |
| **MILP5** |      |      |       |       |        |        |        |
| 8     | 4.27e+05 | 1.20e+02 | 2.80e-04 | 0.00e+00 | 1.00e-02 | 1.65e-03 |
| 9     | 1.71e+06 | 4.87e+02 | 2.85e-04 | 0.00e+00 | 1.00e-02 | 1.66e-03 |
| 10    | 6.84e+06 | 1.25e+04 | 1.83e-03 | 0.00e+00 | 2.00e-02 | 3.87e-03 |
| 11    | 2.74e+07 | 4.25e+04 | 1.55e-03 | 0.00e+00 | 2.00e-02 | 3.62e-03 |
C.1.1 OBDD Representation for Boolean Functions

A Binary Decision Diagram (BDD) $R$ is a rooted directed acyclic graph (DAG) with the following properties. Each $R$ node $v$ is labeled either with a boolean variable $\text{var}(v)$ (internal node) or with a boolean constant $\text{val}(v) \in \mathbb{B}$ (terminal node). Each $R$ internal node $v$ has exactly two children, labeled with $\text{high}(v)$ and $\text{low}(v)$. Let $x_1, \ldots, x_n$ be the boolean variables labeling $R$ internal nodes. Each terminal node $v$ represents $f_v(x) = \text{val}(v)$. Each internal node $v$ represents $f_v(x) = x_i f_{\text{high}(v)}(x) + \bar{x}_i f_{\text{low}(v)}(x)$, being $x_i = \text{var}(v)$.

An Ordered BDD (OBDD) is a BDD where, on each path from the root to a terminal node, the variables labeling each internal node must follow the same ordering.

C.2 OBDDs with Complemented Edges

In this section we introduce OBDDs with complemented edges (COBDDs, Def. [15]), which were first presented in the 90’s. Intuitively, they are OBDDs where else edges (i.e. edges of type $(v, \text{low}(v))$) may be complemented. Then edges (i.e. edges of type $(v, \text{high}(v))$) complementation is not allowed to retain canonicity. Edge complementation usually reduce resources usage, both in terms of CPU and memory.

**Definition 15.** An OBDD with complemented edges (COBDD in the following) is a tuple $\rho = (V, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ with the following properties:

i) $V = \{x_1, \ldots, x_n\}$ is a finite set of ordered boolean variables; ii) $V$ is a finite set of nodes; iii) $1 \in V$ is the terminal node of $\rho$, corresponding to the boolean constant $1$ (non-terminal nodes are called internal); iv) for each internal node $v$, $\text{var}(v) < \text{var}(\text{high}(v))$ and $\text{var}(v) < \text{var}(\text{low}(v))$; v) $\text{var}, \text{low}, \text{high}, \text{flip}$ are functions defined on internal nodes, namely: $\text{var} : V \setminus \{1\} \to V$ assigns to each internal node a boolean variable in $V$, $\text{high} : V \setminus \{1\} \to V$ assigns to each internal node $v$ a high child (or true child), $\text{low} : V \setminus \{1\} \to V$ assigns to each internal node $v$ a low child (or false child), representing the case in which $\text{var}(v) = 1$ (or $\text{var}(v) = 0$), $\text{flip} : V \setminus \{1\} \to \mathbb{B}$ assigns to each internal node $v$ a boolean value; namely, if $\text{flip}(v) = 1$ then the else child has to be complemented, otherwise it is regular (i.e. non-complemented).

**COBDDs associated multigraphs** We associate to a COBDD $\rho = (V, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ a labeled directed multigraph $G(\rho) = (V, E)$ s.t. $V$ is the same set of nodes of $\rho$ and there is an edge $(v, w) \in E$ iff $w$ is a child
of $v$. Moreover, each edge $e \in E$ has a type $\text{type}(e)$, indicating if $e$ is a then, a regular else, or a complemented else edge. Fig. 11 shows an example of a COBDD depicted via its associated multigraph, where edges are directed downwards. Moreover, in Fig. 11 then edges are solid lines, regular else edges are dashed lines and complemented else edges are dotted lines.

The graph associated to a given COBDD $\rho = (\mathcal{V}, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ may be seen as a forest with multiple rooted multigraphs. In order to select one root vertex and thus one rooted multigraph, we define the COBDD restricted to $v \in V$ as the COBDD $\rho_v = (\mathcal{V}, V_v, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ s.t. $V_v = \{w \in V \mid$ there exists a path from $v$ to $w$ in $G(\rho)\}$ (note that $v \in V_v$).

Reduced COBDDs Two COBDDs are isomorphic iff there exists a mapping from nodes to nodes preserving attributes $\text{var}$, $\text{flip}$, $\text{high}$ and $\text{low}$. A COBDD is called reduced iff it contains no vertex $v$ with $\text{low}(v) = \text{high}(v) \land \text{flip}(v) = 0$, nor does it contains distinct vertices $v$ and $v'$ such that $\rho_v$ and $\rho_{v'}$ are isomorphic. Note that, differently from OBDDs, it is possible that $\text{high}(v) = \text{low}(v)$ for some $v \in V$, provided that $\text{flip}(v) = 1$ (e.g. see nodes 0xf and 0xe in Fig. 11). In the following, we assume all our COBDDs to be reduced.

COBDDs Properties For a given COBDD $\rho = (\mathcal{V}, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ the following properties follow from definitions given above: i) $G(\rho)$ is a rooted directed acyclic (multi)graph (DAG); ii) each path in $G(\rho)$ starting from an internal node ends in 1; iii) let $v_1, \ldots, v_k$ be a path in $G(\rho)$, then $\text{var}(v_1) < \ldots < \text{var}(v_k)$.

C.2.1 Semantics of a COBDD

In Def. 16 we define the semantics $[\cdot]$ of each node $v$ of a given COBDD $\rho$ as the boolean function represented by $v$, given the parity $b$ of complemented edges seen on the path from a root to $v$.

Definition 16. Let $\rho = (\mathcal{V}, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD. The semantics of the terminal node 1 w.r.t. a flipping bit $b$ is a boolean function defined as $[1, b]_\rho := \bar{b}$. The semantics of an internal node $v \in V$ w.r.t. a flipping bit $b$ is a boolean function defined as $[v, b]_\rho := x_i[\text{high}(v), b]_\rho + \bar{x}_i[\text{low}(v), b \oplus \text{flip}(v)]_\rho$, being $x_i = \text{var}(v)$. When $\rho$ is understood, we will write $[\cdot]$ instead of $[\cdot]_\rho$. 

67
Example 15. Let $\rho$ be the COBDD depicted in Fig. [11]. If we pick node 0xe we have $\llbracket 0xe, b \rrbracket = x_2[1, b] + \bar{x}_2[1, b \oplus 1] = x_2b + \bar{x}_2b = x_2 \oplus b$.

Theor. [16] states that COBDDs are a canonical representation for boolean functions.

Theorem 16. Let $f : \mathbb{B}^n \rightarrow \mathbb{B}$ be a boolean function. Then there exist a COBDD $\rho = (V, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$, a node $v \in V$ and a flipping bit $b \in \mathbb{B}$ s.t. $\llbracket v, b \rrbracket = f(x)$. Moreover, let $\rho = (V, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD, let $v_1, v_2 \in V$ be nodes and $b_1, b_2 \in \mathbb{B}$ be flipping bits. Then $\llbracket v_1, b_1 \rrbracket = \llbracket v_2, b_2 \rrbracket$ iff $v_1 = v_2 \land b_1 = b_2$.

D Synthesis of C Code from a COBDD

Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization, $\hat{\mathcal{H}} = (\hat{\Gamma}(A_X), \hat{\Gamma}(A_U), \mathcal{N})$ be the close to minimum $\mathcal{Q}$ control abstraction computed by function $\text{minCtrAbs}$ and $\hat{K} : \hat{\Gamma}(A_X) \times \hat{\Gamma}(A_U) \rightarrow \mathbb{B}$ be the strong mgo for the LTS control problem $(\hat{\mathcal{H}}, \hat{\mathcal{I}}, \hat{\mathcal{G}})$. As it is usual in Model Checking, we assume to have encoding functions $\text{enc}_X : \Gamma(A_X) \rightarrow \mathbb{B}^n$ and $\text{enc}_U : \Gamma(A_U) \rightarrow \mathbb{B}^r$ s.t. $n = \sum_{x \in X} (\lfloor \log_2(\gamma_x(\sup A_x) - \gamma_x(\inf A_x)) + 1 \rfloor + 1)$ and $r = \sum_{u \in U} (\lfloor \log_2(\gamma_u(\sup A_u) - \gamma_u(\inf A_u)) + 1 \rfloor + 1)$. This allows us to regard the mgo $\hat{K}$ as a boolean function $K(x_1, \ldots, x_n, u_1, \ldots, u_r)$, by stipulating that $\hat{K}(\hat{x}, \hat{u}) = K(\text{enc}_X(\hat{x}), \text{enc}_U(\hat{u}))$.

Let $\rho = (V, V, 1, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD s.t. there exist $v \in V, b \in \mathbb{B}$ s.t. $\llbracket v, b \rrbracket = K(x_1, \ldots, x_n, u_1, \ldots, u_r)$. Thus, $V = \mathcal{X} \cup U = \{x_1, \ldots, x_n\} \cup \{u_1, \ldots, u_r\}$ (we denote with $\cup$ the disjoint union operator, thus $\mathcal{X} \cap U = \emptyset$). We will call boolean variables $x_i \in \mathcal{X}$ as (boolean) state variables and variables $u_j \in U$ as (boolean) action variables.

D.1 Synthesis Algorithm: Overview

Our method $\text{Synthesize}$ takes as input $\rho$, $v$ and $b$ s.t. $\llbracket v, b \rrbracket = K(x, u)$. Then, it returns as output a C function $\text{void} K(\text{int} *x, \text{int} *u)$ with the following property: if, before a call to $K$, $\forall i \ x[i - 1] = x_i$ holds (array indexes in C language begin from 0) with $\mathbf{x} \in \text{Dom}(K)$, and after the call to $K$, $\forall i \ u[i - 1] = u_i$ holds, then $K(\mathbf{x}, \mathbf{u}) = 1$. Moreover, the WCET of function $K$ is $O(nr)$.
Note that our method Synthesize provides an effective implementation of the mgo $K$, i.e. a C function which takes as input the current state of the LTS and outputs the action to be taken. Thus, $K$ is indeed a control software.

Function Synthesize is organized in two phases. First, starting from $\rho$, $v$ and $b$ (thus from $K(x, u)$), we generate COBDD nodes $v_1, \ldots, v_r$ and flipping bits $b_1, \ldots, b_r$ for boolean functions $f_1, \ldots, f_r$ s.t. each $f_i = \llbracket v_i, b_i \rrbracket$ takes as input the state bit vector $x$ and computes the $i$-th bit $u_i$ of an output action bit vector $u$, where $K(x, u) = 1$, provided that $x \in \text{Dom}(K)$. This computation is carried out in function SolveFunctionalEq. Second, $f_1, \ldots, f_r$ are translated inside function void $K(\text{int} *x, \text{int} *u)$. This step is performed by maintaining the structure of the COBDD nodes representing $f_1, \ldots, f_r$. This allows us to exploit COBDD node sharing in the generated software. This phase is performed by function GenerateCCode.

Thus function Synthesize is organized as in Alg. 6. Correctness for function Synthesize is stated in Theor. 17.

### Algorithm 6 Translating COBDDs to a C function

**Input:** COBDD $\rho$, node $v$, boolean $b$

**function** Synthesize($\rho$, $v$, $b$):

1. $\langle v_1, b_1, \ldots, v_r, b_r \rangle \leftarrow$ SolveFunctionalEq($\rho$, $v$, $b$)

2. GenerateCCode($\rho$, $v_1, b_1, \ldots, v_r, b_r$)

### D.2 Synthesis Algorithm: Solving Functional Equation

In this phase, starting from $\rho$, $v$ and $b$ (thus from $\llbracket v, b \rrbracket = K(x, u)$), we compute functions $f_1, \ldots, f_r$ s.t. for all $x \in \text{Dom}(K)$, $K(x, f_1(x), \ldots, f_r(x)) = 1$.

To this aim, we follow an approach similar to the one presented in [43]. Namely, we compute $f_i$ using $f_1, \ldots, f_{i-1}$, in the following way: $f_i(x) = \exists u_{i+1}, \ldots, u_n K(x, f_1(x), \ldots, f_{i-1}(x), 1, u_{i+1}, \ldots, u_n)$. Thus, function SolveFunctionalEq($\rho$, $v$, $b$) computes and returns $\langle v_1, b_1, \ldots, v_r, b_r \rangle$ s.t. for all $i \in [r]$, $\llbracket v_i, b_i \rrbracket = f_i(x)$.

### D.3 Synthesis Algorithm: Generating C Code

In this phase, starting from COBDD nodes $v_1, \ldots, v_r$ and flipping bits $b_1, \ldots, b_r$ for functions $f_1, \ldots, f_r$ generated in the first phase, we generate two C functions: i) void $K(\text{int} *x, \text{int} *u)$, which is the required output
function for our method Synthesize; ii) int K_bits(int *x, int action), which is an auxiliary function called by K. A call to K_bits(x, i) returns f_i(x), being x[j - 1] = x_j for all j ∈ [n]. This phase is detailed in Algs. 7 (function GenerateCCode) and 8 (function Translate).

Given inputs ρ, v_1, b_1, . . . , v_r, b_r (output by SolveFunctionalEq), Algs. 7 and 8 work as follows. First, function int K_bits(int *x, int action) is generated. If x[j - 1] = x_j for all j ∈ [n], the call K_bits(x, i) has to return f_i(x). In order to do this, K_bits(x, i) traverses the graph G(ρ_v_i) by taking, in each node v, the then edge if x[j - 1] = 1 (with j s.t. var(v) = x_j) and the else edge otherwise. When node 1 is reached, then 1 is returned iff the integer sum c + b_i is even, being c the number of complemented else edges traversed. Parity of c + b_i is maintained by initializing a C variable ret_b to ¬b_i, then complementing ret_b when a complemented else edge is traversed, and finally returning ret_b.

Thus, Algs. 7 and 8 generate K_bits in order to obtain the above described behavior. Namely, for all v_i output by the first phase (function SolveFunctionalEq), GenerateCCode calls Translate with parameters ρ, v_i, W, where W maintains the set of nodes already translated in C code. This results, for all such v_i, in a recursive graph traversal of G(ρ_v_i) where, for each internal node w /∈ W which was not already translated, a C code block B = B_1B_2 is generated s.t. B_1 is of the form L_w: if (x[j - 1]) goto L_h; (line 7 of Alg. 8) and B_2 has one of the following forms: i) else goto L_l; (if flip(w) = 0, line 9 of Alg. 8) or ii) else {ret_b = !ret_b; goto L_l;} (otherwise, line 8 of Alg. 8). For the terminal node, the block L_1: return ret_b; is generated. Note that maintaining the set of already translated nodes W allows us to fully exploit COBDDs nodes sharing.

Algorithm Correctness Correctness of our approach, i.e. of function Synthesize in Alg. 6 is stated by Th. 17 (for the proof, see [31]).

Theorem 17. Let ρ = (V, V, 1, var, low, high, flip) be a COBDD with V = X∪U, v ∈ V be a node, b ∈ B be a boolean. Let [v, b] = K(x, u). Then function Synthesize(ρ, v, b) generates a C function void K(int *x, int *u) with the following property: for all x ∈ Dom(K), if before a call to K ∀i ∈ [n] x[i - 1] = x_i, and after the call to K ∀i ∈ [r] u[i - 1] = u_i, then K(x, u) = 1. Furthermore, function K has WCET O(nr).
Algorithm 7 Generating C functions

Input: COBDD $\rho$, $v_1, \ldots, v_r$, boolean values $b_1, \ldots, b_r$

function $\text{GenerateCCode}(\rho, v_1, b_1, \ldots, v_r, b_r)$:

1. print 
   
   \[ \text{int K_bits(int *x, int action) \{ int ret_b; switch(action) \{}} \]

2. for all $i \in [r]$ do

3. print 
   
   \[ \text{case } i-1, \text{": ret_b = } \vec{b}_i, \text{; goto L_}; v_i,\text{;}} \]

4. print 
   
   \[ \text{\}/* end of the switch block */} \]

5. $W \leftarrow \emptyset$

6. for all do $i \in [r]$ \(W \leftarrow \text{Translate}(\rho, v_i, W)\) done

7. print 
   
   \[ \text{\} K(int *x, int *u)\{int i; for(i = 0; i < r; i++ \}
   
   \]

   \[ \text{u[i] = K_bits(x, i);} \} \]

Algorithm 8 COBDD nodes translation

Input: COBDD $\rho$, node $v$, nodes set $W$

function $\text{Translate}(\rho, v, W)$:

1. if $v \in W$ then return $W$

2. $W \leftarrow W \cup \{v\}$, print “L_”, $v$, “:”

3. if $v = 1$ then

4. print “return ret_b;”

5. else

6. let $i$ be s.t. $\text{var}(v) = x_i$

7. print “if(x["i-1"] == 1) goto L_”, high($v$)

8. if flip($v$) then print “else \{ret_b=!ret_b; goto L_", low($v$), “;}”

9. else print “else goto L_”, low($v$)

10. $W \leftarrow \text{Translate}(\rho, \text{high}(v), W)$

11. $W \leftarrow \text{Translate}(\rho, \text{low}(v), W)$

12. return $W$
An Example of Translation  Consider the COBDD $\rho$ shown in Fig. 11. Within $\rho$, consider mgo $K(x_0, x_1, x_2, u_0, u_1) = [0x17, 1]$. By applying $\text{SolveFunctionalEq}$, we obtain $f_1(x_0, x_1, x_2) = [0x15, 1]$ and $f_2(x_0, x_1, x_2) = [0x10, 1]$. Note that 0xe is shared between $G(\rho_{0x15})$ and $G(\rho_{0x10})$. Finally, by calling $\text{GenerateCCode}$ (see Alg. 7) on $f_1, f_2$, we have the C code in Fig. 12.

E  A DTLHS Buck Model Robust on $R$ and $V_i$

In this section we address the problem of refining the model given in Sect. 3.1 so as to require a controller for our buck to be robust to foreseen variations in the load $R$ and in the power supply $V_i$. That is, given tolerances $\rho_R$ and $\rho_{V_i}$, we want the controller output by QKS for our buck to work for any $R \in [\max\{0, R(1-\rho_R)\}, R(1+\rho_R)]$ and any $V_i \in [\max\{0, V_i(1-\rho_{V_i})\}, V_i(1+\rho_{V_i})]$.

Variations in the power supply are modeled by replacing Eq. (12) in Sect. 3.1 with the following:

$$v_D \leq v_u - V_i(1-\rho_{V_i}) \quad (15) \quad v_D \geq v_u - V_i(1+\rho_{V_i}) \quad (16)$$

Along the same lines, we may model also variations in the load $R$. However, since $N$ dynamics is not linear in $R$, much more work is needed (along
tonically increasing functions for $R$$\tau$ a
$v$
replace Eq. (4) with two inequalities
the lines of [21]).
T o this aim, we proceed as follows.
as (nonlinear) functions of
$v$
being
auxiliary boolean variables
$z$
$z$
$z$
$z$

The boolean variable
$npp$
n $w$

Figure 12: C code for mgo in Fig. 11 as generated by Synthesize

the lines of [21]). To this aim, we proceed as follows.
The only equation depending on $R$ is Eq. (4) of Sect. 3.1 Consider constants $a_{2,1}(R) = \frac{r}{r_c+R}[-\frac{r_r}{r_c+R} + \frac{1}{r_c}], a_{2,2}(R) = -\frac{1}{r_c+R}[-\frac{r_r}{r_c+R} + \frac{1}{r_c}], a_{2,3}(R) = -\frac{1}{r_c+R}$ as (nonlinear) functions of $R$. It is easy to see that $a_{2,1}(R)$, $a_{2,2}(R)$ are monotonically increasing functions for $R \in \mathbb{R}^+$, while $a_{2,3}(R)$ is monotonically decreasing for $R \in \mathbb{R}^+$. Thus, if signs of $i_L$, $v_O$, $v_D$ are known, it is possible to replace Eq. (4) with two inequalities $v_O \geq Ta_{2,1}(R_{i_L}^-)i_L + (1+Ta_{2,2}(R_{i_O}^-))v_O + Ta_{2,3}(R_{v_D}^-)v_D$ and $v_O \leq Ta_{2,1}(R_{i_L}^+)i_L + (1+Ta_{2,2}(R_{i_O}^+))v_O + Ta_{2,3}(R_{v_D}^+)v_D$, being

- $R_w^- = \text{if } w \geq 0 \text{ then } R(1-\rho_R) \text{ else } R(1+\rho_R)$ and $R_w^+ = \text{if } w \geq 0 \text{ then } R(1+\rho_R) \text{ else } R(1-\rho_R)$ for $w \in \{i_L, v_O\}$;

- $R_{v_D}^- = \text{if } v_D \geq 0 \text{ then } R(1+\rho_R) \text{ else } R(1-\rho_R)$ and $R_{v_D}^+ = \text{if } v_D \geq 0 \text{ then } R(1-\rho_R) \text{ else } R(1+\rho_R)$.

This leads us to replace Eq. (4) of Sect. 3.1 with the equations in Fig. 13
Note that, w.r.t. the model in Sect. 3.1 in Fig. 13 we add to $Y^b$ 11 auxiliary boolean variables $z_{i_L}$, $z_{v_O}$, $z_{v_D}$, $z_{ppp}$, $z_{ppp}$, $z_{ppn}$, $z_{ppn}$, $z_{pnp}$, $z_{pnp}$, $z_{pnn}$, $z_{pnn}$, $z_{nnn}$, $z_{nnn}$ with the following meaning. The boolean variable $z_{i_L}$ is true iff $i_L$ is positive (see Eqs. (17) and (20) [Eqs. (18) and (21), Eqs. (19) and (22)]. The boolean

\begin{verbatim}
int K_bits(int *x, int action) { int ret_b;
switch(action) { case 0: ret_b = 0; goto L_0x15;
case 1: ret_b = 0; goto L_0x10; }
L_0x15: if (x[0] == 1) goto L_0x13;
else { ret_b = !ret_b; goto L_0x14; }
L_0x13: if (x[1] == 1) goto L_0xe;
else { ret_b = !ret_b; goto L_0x1; }
L_0xe: if (x[2] == 1) goto L_0x1;
else { ret_b = !ret_b; goto L_0x1; }
L_0x14: if (x[1] == 1) goto L_0xe;
else goto L_1;
L_0x10: if (x[0] == 1) goto L_0xe;
else { ret_b = !ret_b; goto L_0xf; }
L_0xf: if (x[1] == 1) goto L_0xe;
else { ret_b = !ret_b; goto L_0xe; }
L_1: return ret_b; }

void K(int *x, int *u) { int i;
for(i = 0; i < 2; i++) u[i] = K_bits(x, i); }
\end{verbatim}
\[
\begin{align*}
  z_{iL} &\rightarrow i_L \geq 0 & (17) & \quad \overline{z_{iL}} &\rightarrow i_L \leq 0 & (20) \\
  z_{vO} &\rightarrow v_O \geq 0 & (18) & \quad \overline{z_{vO}} &\rightarrow v_O \leq 0 & (21) \\
  z_{vD} &\rightarrow v_D \geq 0 & (19) & \quad \overline{z_{vD}} &\rightarrow v_D \leq 0 & (22) \\
  \overline{z_{ppp}} &\rightarrow 1 - z_{iL} + 1 - z_{vO} + 1 - z_{vD} \geq 1 & (23) \\
  \overline{z_{pnp}} &\rightarrow 1 - z_{iL} + z_{vO} + 1 - z_{vD} \geq 1 & (24) \\
  \overline{z_{ppn}} &\rightarrow 1 - z_{iL} + 1 - z_{vO} + z_{vD} \geq 1 & (25) \\
  \overline{z_{pnn}} &\rightarrow 1 - z_{iL} + z_{vO} + z_{vD} \geq 1 & (26) \\
  \overline{z_{npp}} &\rightarrow z_{iL} + 1 - z_{vO} + 1 - z_{vD} \geq 1 & (27) \\
  \overline{z_{npn}} &\rightarrow z_{iL} + z_{vO} + 1 - z_{vD} \geq 1 & (28) \\
  \overline{z_{nnp}} &\rightarrow z_{iL} + 1 - z_{vO} + z_{vD} \geq 1 & (29) \\
  \overline{z_{nnn}} &\rightarrow z_{iL} + z_{vO} + z_{vD} \geq 1 & (30) \\
  z_{ppp} &\rightarrow v'_O \leq T_{a_{2,1}}^{(M)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (31) \\
  z_{pnp} &\rightarrow v'_O \leq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(M)} v_D & (32) \\
  z_{ppn} &\rightarrow v'_O \leq T_{a_{2,1}}^{(M)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(M)} v_D & (33) \\
  z_{pnn} &\rightarrow v'_O \leq T_{a_{2,1}}^{(M)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (34) \\
  z_{pnp} &\rightarrow v'_O \leq T_{a_{2,1}}^{(M)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (35) \\
  z_{ppp} &\rightarrow v'_O \geq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(M)} v_D & (36) \\
  z_{npp} &\rightarrow v'_O \leq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(M)} v_D & (37) \\
  z_{nnp} &\rightarrow v'_O \geq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(M)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (38) \\
  z_{nnp} &\rightarrow v'_O \leq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(M)} + 1)v_O + T_{b_{2,1}}^{(M)} v_D & (39) \\
  z_{nnn} &\rightarrow v'_O \leq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (40) \\
  z_{nnp} &\rightarrow v'_O \leq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (41) \\
  z_{nnp} &\rightarrow v'_O \geq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(M)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (42) \\
  z_{nnn} &\rightarrow v'_O \geq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (43) \\
  z_{nnp} &\rightarrow v'_O \geq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(M)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (44) \\
  z_{nnn} &\rightarrow v'_O \leq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (45) \\
  z_{nnn} &\rightarrow v'_O \geq T_{a_{2,1}}^{(m)} i_L + (T_{a_{2,2}}^{(m)} + 1)v_O + T_{b_{2,1}}^{(m)} v_D & (46)
\end{align*}
\]

Figure 13: DTLHS Buck Model Robust on $R$

74
variable $z_{abc}$, with $a, b, c \in \{p, n\}$, is true iff 
\[
\text{if } a = p \text{ then } i_L \geq 0 \text{ else } i_L \leq 0 \right) \land 
\text{if } b = p \text{ then } v_O \geq 0 \text{ else } v_O \leq 0 \right) \land 
\text{if } c = p \text{ then } v_D \geq 0 \text{ else } v_D \leq 0 \right).
\]
This is stated by Eqs. (23)–(30). Boolean variables $z_{abc}$ are then
used as guards for the inequalities replacing Eq. (4) as stated before. This is
done in Eqs. (31)–(46).

Finally, the transition relation $N$ of $\mathcal{H}$ is given by the conjunction of the
constraints given above and the following explicit (safety) bounds:
\[
-4 \leq i_L \leq 4 \land -1 \leq v_O \leq 7 \land -10^3 \leq i_D \leq 10^3 \land -10^3 \leq i_u \leq 10^3 \land -10^7 \leq v_u \leq 10^7.
\]