On-the-Fly Control Software Synthesis *

V. Alimguzhin[1,2], F. Mari[1], I. Melatti[1], I. Salvo[1], and E. Tronci[1]

[1] Dip. di Informatica Sapienza Università di Roma,
Via Salaria 113, 00198 Roma, Italy

[2] Department of Computer Science and Robotics Ufa State Aviation Technical University 12 Karl Marx Street, Ufa, 450000, Russian Federation

Abstract. The Model Based Design approach for Hybrid Systems control software synthesis is particularly appealing since Formal System Level Specifications are usually much easier to define than the control software itself. In this setting, *Design Space Exploration* has the goal to find a suitable (with respect to *costs* and *performance*) choice for system *design parameters*. Unfortunately, a substantial part of the time devoted to design space exploration is spent trying to solve control software synthesis problems that do not have a solution. We present an *on-the-fly* algorithm to control software synthesis that enables effective design space exploration by speeding-up termination when no controller is found. Our experimental results show the effectiveness of our approach and how it can support a concrete realizability and schedulability analysis.

1 Introduction

A *Software Based Control System* (SBCS) consists of two main subsystems, the *controller* and the *plant* that together form the *closed loop system*. In an endless loop, every T seconds (*sampling time*), output y from plant sensors go through an *analog-to-digital* (AD) conversion, yielding a *quantized* value $\hat{y}$ to the control software implementing the *control law*. The control software then computes the command $\hat{u}$ to be sent (after a *digital-to-analog* (DA) conversion) to plant actuators in order to guarantee that the closed loop system satisfies given *safety* and *liveness* specifications (*System Level Formal Specifications*).

Traditionally, the control software is designed using a *separation-of-concerns* approach. That is, *Control Engineering* techniques (e.g., see [10]) are used to design functional specifications (control law) from the closed loop system level specifications, whereas *Software Engineering* techniques are used to design control software implementing functional specifications.

Motivations In SBCS design the interface between Control Engineering and Software Engineering activities is basically summarized by the choice of: 1) control law, 2) number of quantization bits b, 3) sampling time T. Taking into account that a SBCS is a real-time system, the control software *Worst Case Execution Time* (WCET) must be less than or equal to T. As a result we have contrasting requirements on the choice of *design parameters* b and T. Namely, typically performance (e.g., set-up time and ripple) of the closed loop system improves as b increases or T decreases. On the other hand, hardware/software

* This work has been partially supported by the the EC FP7 projects SmartHG (Energy Demand Aware Open Services for Smart Grid Intelligent Automation, 317761) and PAEON (Model Driven Computation of Treatments for Infertility Related Endocrinological Diseases, 600773).

costs decrease when b decreases or T increases (e.g., a faster processor is needed in order to guarantee that the control software WCET is less than T).

In our context, one of the main goals of *Design Space Exploration* is to find a suitable (with respect to costs and performance) choice for *design parameters* b and T. The current approach is to define (using Control Engineering techniques) a control law along with values for b and T and then to devise (using Software Engineering techniques) a software implementation for it. Once the software is implemented, its *realizability* its *schedulability* must be evaluated. Namely, the software is realizable if it fits in the microcontroller flash memory. Moreover, it is schedulable if its WCET is smaller of the sampling time and small enough to make feasible the schedulability of other periodic processes (as reading quantized values from plant sensors) that run on the same microcontroller (see e.g. [15] for a more-in-depth discussion). Performance of the closed loop system is then evaluated using *Hardware In the Loop Simulation* (e.g., nicely supported by *Model Based* tools like Simulink [20] or Reactis [34]).

One may wish to partially automate design space exploration by using tools like QKS [25] that from the plant model, system level formal specifications for the closed loop system and implementation parameters (namely, number of quantization bits), automatically synthesize correct-by-construction control software meeting the given requirements and with a guaranteed WCET. We note that, for many choices of the design parameters b and T, QKS fails to find control software solving the synthesis problem. As a result, a substantial part of the time devoted to design space exploration will be spent trying to solve control software synthesis problems that do not have a solution. Unfortunately the control software synthesis algorithm presented in [25] takes about the same time both when it finds a solution and when it cannot find one.

This paper investigates control software synthesis algorithms that can support design space exploration by detecting *as soon as possible* when a solution to the synthesis problem cannot be found.

Our Contributions We model the plant as a *Discrete Time Linear Hybrid System* (DTLHS), that is a (discrete time) hybrid system whose dynamics is modeled with linear constraints over a set of continuous as well as discrete variables. Safety and liveness specifications for the closed loop system are defined as linear constraints on state variables. A DTLHS $\mathcal{H}$ approximates a continuous time system dynamics by *sampling* it only at discrete time points multiple of a *time step* τ chosen on the base of physical considerations. Building on this we can approximate the dynamics of a system *sampled* each $T = n\tau$ seconds by iterating n times the dynamics of $\mathcal{H}$. Using such an approach we can investigate in our DTLHS framework existence of a controller for $\mathcal{H}$ for different configurations of b and $T = n\tau$. Our main contributions can be summarized as follows.

On-the-fly control software synthesis algorithm. We present an *on-the-fly* algorithm for DTLHS control software synthesis, in the same spirit of on-the-fly Model Checking [19]. Such an approach enables effective design space exploration by speeding-up termination of the control software synthesis algorithm in the typical case occurring in the design space exploration phase, namely when no controller is found for the given configuration parameters (b, T).

Experimental results. We implemented our algorithm within the *QKS* tool [25]. To assess the effectiveness of our approach we present results on its usage for design space exploration of control software for the inverted pendulum, a challenging and widely studied example (e.g., see [22]). We carry out such a design space exploration using both the on-the-fly algorithm presented here and the synthesis algorithm presented in [25]. We have considered 18 choices for the design parameters b and T, 10 of which return a control software. Our experimental results (Sect. 6) show that, using our on-the-fly algorithm we have a time saving of nearly 80%. Finally, we show how our Model Based Design approach can effectively support a concrete realizability and schedulability analysis on a specific family of microcontrollers.

Related Work *Model based* design space exploration for embedded systems (typically modeled as *Hybrid Systems* [5]) has been widely studied in the last decades. Many tools and paradigms have been proposed to support designers in this phase. For example, see [6] and citations thereof for a *formal* (using UPPAAL [17]) model based tool and a survey on available tools. In this respect we note that all proposed methods focus on designing the software/hardware system once the control law is given and, in particular, once b (number of quantization bits) and T (sampling time) are given. To the best of our knowledge none of them supports *trading* between Control Engineering *wishes* (large b and small T) and System/Software Engineering *wishes* (small b and large T) *before* the control law is designed. In such a framework our contribution complements the available approaches by enabling trade-offs between the control law, b and T before the control law is designed.

The sampling time T is one of the main requirements to take into account for schedulability analysis. In [24] is proposed a scheduling algorithm that cleverly trades, at run time, T (by delaying execution of control software) and closed loop performances. The main difference with our contribution is that in [24] the control law and b are both given whereas our approach enables exploring (*off-line*) the possibility of changing any of them in order to increase T. It is worth noticing that indeed the approach in [24] could be used to further increase (at run time) the T resulting from our control software synthesis method.

We check performance of the closed loop system after control software synthesis. Methods to synthesize control laws satisfying given performance indexes on the closed loop system have been investigated, for example, in [21]. We differ from such work since our plant model is a DTLHS rather than a multi-modal system for [21].

Automatic synthesis of software from models has also been widely studied. For example, see [23] and citations thereof. We differ from such approaches since our starting point is the plant model and closed loop specifications for the closed loop system whereas model based software generation (e.g., as the one also available in tools like Simulink) starts from a model based definition (e.g., using Stateflow/Simulink diagrams) of the control law and then generates a software implementation for such a control law model.

Control software synthesis from formal system level specifications for Discrete Time (possibly non Linear) Hybrid Systems has been investigated in [25, 26, 3, 2]. The *on-the-fly* algorithm presented here improves on the one in [25, 26] by

reducing of about 99% the time to terminate when it cannot find a controller, possibly at a price of a 25% time penalty when it can find one. This, in turn, enables, formal model based design space (i.e.: *control law*, b, T) exploration.

On-the-fly algorithms for the analysis of Timed Games has been proposed in [12]. Our backward algorithm has to handle linear constraints where both continuous and discrete state variables may appear. In fact, we need to solve many MILP problems to back-propagate a state region. This is quite different from the class of Timed Automata considered in [12], where constraints have the form $x \sim k$, where x is a clock and $\sim$ is one of $<, \leq, \geq, >, =$.

In [31] it is presented a semi-automatic method that, taking as input a continuous time linear system and a goal specification, produces a control law (represented as an OBDD) through PESSOA [30, 35]. Such an approach differs from ours as follows. First, our method is fully automatic whereas the one in [31] is not, since it relies on a user provided Lyapunov function, much in the spirit of [22]. Second, [31] does not provide any guarantee on the WCET of the generated software, thus it cannot be used for design space exploration in our context.

Verification and control law synthesis for *Linear Hybrid Automata* (LHA) [4] has been investigated, e.g., in [18, 38, 16, 9]. Control law synthesis for *Piecewise Affine Discrete Time Hybrid Systems* (PWA-DTHS) has been investigated in [7, 8]. All such approaches, when dealing with control synthesis, do not account for state feedback quantization since they all assume *exact* (i.e. real valued) state measures and do not generate control software with a guaranteed WCET. As a result they cannot be used for design space exploration in our context, where the number of AD bits b and the software WCET play a crucial role.

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 of variables. We may regard X as a set when convenient. Each variable x ranges over a bounded or unbounded interval Γ_x, being either $\Gamma_x \subseteq \mathbb{R}$ or $\Gamma_x \subseteq \mathbb{Z}$. We say that Γ_x is a *typing* for x and $\Gamma_X = \prod_{x \in X} \Gamma_x$ is a typing for X. If, for all $x \in X$, Γ_x is a bounded interval, we say that Γ_X is a *bounded* typing for X.

Predicates A *linear expression* $L(X)$ over a list of variables X is a linear combination of variables in X with rational coefficients, $\sum_{x_i \in X} a_i x_i$. A *linear constraint* over X (or simply a *constraint*) is an expression of the form $L(X) \leq b$, where b is a rational constant. *Predicates* are inductively defined as follows. A constraint $C(X)$ is a predicate. If $A(X)$ and $B(X)$ are predicates then $(A(X) \wedge B(X))$ and $(A(X) \vee B(X))$ are predicates. 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: $L(X) \geq b$ for $-L(X) \leq -b$, $L(X) = b$ for $((L(X) \leq b) \wedge (L(X) \geq b))$, and $a \leq x \leq b$ for $x \geq a \ \wedge \ x \leq b$, where $x \in X$.

A *valuation* over a list of variables X is a function v that maps each variable $x \in X$ to a value $v(x) \in \Gamma_x$. Given a valuation v, we denote with $X^* \in \Gamma_X$ 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 $P(X)$. A variable $x \in X$ is said to be *bounded* in P if there exist a, $b \in \Gamma_x$ such that $P(X)$ implies $a \leq x \leq b$. A predicate P is bounded if all its variables are bounded.

Given a constraint $C(X)$ and a fresh boolean variable (*guard*) $y \notin X$, the *guarded constraint* $y \rightarrow C(X)$ (if y then $C(X)$) denotes the predicate $((y = 0) \vee C(X))$. Similarly, we use $\bar{y} \rightarrow C(X)$ (if not y then $C(X)$) to denote the predicate $((y = 1) \vee C(X))$. A *guarded predicate* is a conjunction of either constraints or guarded constraints. If a guarded predicate P is bounded, then P can be transformed into a (bounded) conjunctive predicate [27].

A *linear predicate* $P(X)$ is a (guarded) predicate or an expression of form $\exists Z \in \Gamma_Z \ \tilde{P}(X, Z)$, where $\tilde{P}(X, Z)$ is a (guarded) predicate and Z is set of *auxiliary variables*. Note that, if $\tilde{P}(X, Z)$ is bounded, then $P(X)$ is also bounded.

Mixed Integer Linear Programming A MILP problem with *decision variables* X is a tuple (max, $J(X)$, $A(X)$) where: X is a list of variables, $J(X)$ (*objective function*) is a linear expression on X, and $A(X)$ (*constraints*) is a conjunctive predicate on X. A *solution* to $(\max, J(X), A(X))$ is a valuation X^* such that $A(X^*)$ and $\forall Z \ (A(Z) \ \rightarrow \ (J(Z) \leq J(X^*)))$. $J(X^*)$ is the *optimal value* of the MILP problem. A *feasibility* problem is a MILP problem of the form $(\max, 0, A(X))$. We write also $A(X)$ for $(\max, 0, A(X))$. We write $(\min, J(X), A(X))$ for $(\max, -J(X), A(X))$.

Moore Automata A *Nondeterministic Moore Automaton* (NMA) [13] is a tuple $\mathcal{M} = (S, A, O, T, \Omega)$ where: S is a set of states, A is a set of *actions*, O is a set of *outputs*, $T : S \times A \times S \rightarrow \mathbb{B}$ is the *transition relation* of $\mathcal{M}$, and $\Omega : S \times O \rightarrow \mathbb{B}$ is the *output predicate*, such that $\forall s \in S \ \exists o \in O \ \Omega(s, o)$ (there is an output for each state). In the following, let $s \in S$, $a \in A$ and $o \in O$.

The set of actions *enabled* in s is denoted by $\mathrm{En}(\mathcal{M}, s) = \{a \in A \mid \exists s' T(s, a, s')]\}$. An action a is *enabled* in $o \in O$, notation $\mathrm{En}(\mathcal{M}, o)$ if there exists a state s such that $\Omega(s, o)$ holds and $a \in \mathrm{En}(\mathcal{M}, s)$. An action a is *admissible* in o, notation $\mathrm{Adm}(\mathcal{M}, o, a)$ if it is enabled in o and for all s such that $\Omega(s, o)$ holds a is enabled in s. The *image* of s through a is denoted by $\mathrm{Img}(\mathcal{M}, s, a) = \{s' \in S \mid T(s, a, s')\}$. We call *transition* of $\mathcal{M}$ a tuple (s, a, s') s.t. $T(s, a, s')$ and *self–loop* a transition (s, a, s') s.t. $T(s, a, s') \wedge \exists o[\Omega(s, o) \wedge \Omega(s', o)]$.

A *run* or *path* for an NMA $\mathcal{M}$ is a sequence $\pi = s_0, a_0, s_1, a_1, s_2, a_2, \ldots$ of states s_t and actions a_t such that $\forall t \geq 0 \ T(s_t, a_t, s_{t+1})$. The length $|\pi|$ of a finite run π is the number of actions in π. We denote with $\pi^{(S)}(t)$ the t-th state element of π, and with $\pi^{(A)}(t)$ the t-th action element of π. That is $\pi^{(S)}(t) = s_t$, and $\pi^{(A)}(t) = a_t$.

Given two NMAs $\mathcal{M}_1 = (S, A, O, T_1, \Omega)$ and $\mathcal{M}_2 = (S, A, O, T_2, \Omega)$, we write $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ iff $T_1(s, a, s')$ implies $T_2(s, a, s')$ for each state $s, s' \in S$ and action $a \in A$.

We call a NMA $\mathcal{M} = (S, A, O, T, \Omega)$ a Labelled Transition System (LTS) whenever $S = O$ and for all s_1, s_2 if $\Omega(s_1, s_2)$ holds then $s_1 = s_2$. In such a case we may write simply $\mathcal{M} = (S, A, T)$.

3 Output Feedback Control Problem

A *controller* restricts the dynamics of a system, so that all paths starting in a initial state, eventually reach a state in a goal region (*liveness specifications*), while keeping the system in the safe region (*safety specifications*). In this section, we formally define the notion of output feedback control problem and its solutions, by extending to possibly infinite NMAs the definitions in [37, 14] for finite LTSs. With respect to [25], the output feedback control problem slightly generalize the notion of quantized feedback control problem in order to provide a natural framework for modelling control problems where plant state is not fully observable. In what follows, let $\mathcal{M} = (S, A, O, T, \Omega)$ be an NMA, and $I, \Sigma, G \subseteq S$ be, respectively, the initial, the safe, and the goal region.

An *output feedback controller* for $\mathcal{M}$ is a function $K : O \times A \to \mathbb{B}$ such that $\forall o \in O$, $\forall a \in A$, if $K(o, a)$ then $\mathrm{Adm}(\mathcal{M}, o, a)$. We denote with $\mathrm{dom}(K)$ the set of states for which a control action is defined. Formally, $\mathrm{dom}(K) = \{s \in S \mid \exists a \exists o\ \Omega(s, o) \wedge K(o, a)\}$. $\mathcal{M}^{(K)}$ denotes the *closed loop system*, that is the NMA $(S, A, O, T^{(K)}, \Omega)$, where $T^{(K)}(s, a, s') = T(s, a, s') \wedge \exists o[\Omega(s, o) \wedge K(o, a)]$. $\mathcal{M}_\Sigma$ denotes the *safe system*, that is the NMA $(S, A, O, T_\Sigma, \Omega)$, where $T_\Sigma(s, a, s') = T(s, a, s') \wedge \Sigma(s')$.

We call a path π *fullpath* if either it is infinite or its last state $\pi^{(S)}(|\pi|)$ has no successors (i.e. $\mathrm{Adm}(\mathcal{M}, \pi^{(S)}(|\pi|)) = \varnothing$). We denote with $\mathrm{Path}(s, a)$ the set of fullpaths starting in state s with action a, i.e. the set of fullpaths π such that $\pi^{(S)}(0) = s$ and $\pi^{(A)}(0) = a$. Given a path π in $\mathcal{M}$, we define the measure $j(\mathcal{M}, G, \pi)$ on paths as the distance of $\pi^{(S)}(0)$ to the goal on π. That is, if there exists $n > 0$ s.t. $\pi^{(S)}(n) \in G$, then $j(\mathcal{M}, G, \pi) = \min\{n \mid n > 0 \wedge \pi^{(S)}(n) \in G\}$. Otherwise, $j(\mathcal{M}, G, \pi) = +\infty$. We require $n > 0$ since our systems are nonterminating and each controllable state (including a goal state) must have a path of positive length to a goal state. Taking $\sup \varnothing = +\infty$, the *worst case distance* of a state s from the goal region G is $J(\mathcal{M}, G, s) = \sup\{j(\mathcal{M}, G, \pi) \mid \pi \in \mathrm{Path}(s, a), a \in \mathrm{Adm}(\mathcal{M}, s)\}$.

Definition 1. *An* NMA output feedback control problem *$\mathcal{P}$ is a tuple $(\mathcal{M}, I, \Sigma, G)$. An* LTS control problem *is an NMA output feedback control problem where $\mathcal{M}$ is an LTS and $\Sigma = S$, thus it is a triple $(\mathcal{M}, I, G)$.*

A strong solution *(or simply, a* solution*) to $\mathcal{P}$ is a controller K for $\mathcal{M}_\Sigma$ such that $I \subseteq \mathrm{dom}(K)$, and for all $s \in \mathrm{dom}(K)$, $J(\mathcal{M}_\Sigma^{(K)}, G, s)$ is finite.*

An optimal *solution to $\mathcal{P}$ is a solution K^* to $\mathcal{P}$ such that for all solutions K to $\mathcal{P}$, for all $s \in S$, we have $J(\mathcal{M}_\Sigma^{(K^*)}, G, s) \leq J(\mathcal{M}_\Sigma^{(K)}, G, s)$.*

The most general optimal (mgo) solution *to $\mathcal{P}$ is an optimal solution $\tilde{K}$ to $\mathcal{P}$ such that for all other optimal solutions K to $\mathcal{P}$, for all $o \in O$, for all $a \in A$ we have that $K(o, a) \to \tilde{K}(o, a)$.*

Intuitively, a strong solution takes a *pessimistic* view by requiring that for each initial state, *all* runs in the closed loop system reach the goal, no matter nondeterminism outcomes.

Example 1. Let $S = \{-1, 0, 1\} \times \{0, 1, 2\}$, $A = \{-1, 0, 1\}$, and $T : S \times A \times S \to \mathbb{B}$ be defined by all arrows in Fig. 1. Let us consider the set of outputs $O_1 = \{-1, 0, 1\}$, the output relation $\Omega_1 = \{((s_1, s_2), s_1) \mid (s_1, s_2) \in S\}$, and the NMA

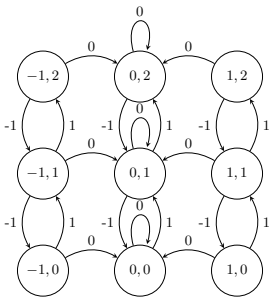

Fig. 1. Transition relation of NMAs $\mathcal{M}_1$ and $\mathcal{M}_2$ in Example 1.

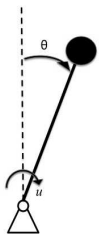

Fig. 2. Inverted Pendulum with Stationary Pivot Point.

$\mathcal{M}_1 = (S, A, O_1, T, \Omega_1)$. Let $I = \Sigma = S$ and $G = \{(0,1)\}$. The NMA output feedback control problem $\mathcal{P}_1 = (\mathcal{M}_1, I, \Sigma, G)$ has no solution, because on output 0 it is not possible to determine if the correct action to enable is 0 (as it is in state $(0,1)$), 1 (as it is in state $(0,0)$), or -1 (as it is in state $(0,2)$).

Let us now consider the set of outputs $O_2 = \{0,1,2\}$ and the output relation $\Omega_2 = \{((s_1,s_2),s_2) \mid (s_1,s_2) \in S\}$, and the NMA $\mathcal{M}_2 = (S, A, O_2, T, \Omega_2)$. The NMA output feedback control problem $\mathcal{P}_2 = (\mathcal{M}_2, I, \Sigma, G)$ has the mgo solution $K(o,a) = ((o=0) \to (a=1)) \wedge ((o=1) \to (a=0)) \wedge ((o=2) \to (a=-1))$.

4 Discrete Time Linear Hybrid Systems

Discrete Time Linear Hybrid Systems (DTLHSs) provide a uniform framework to model both the plant and the closed loop system. In this section, we extend the definition of DTLHSs in [25] by considering *outputs* in order to model measurements of system state (as usual in Control Theory [36]).

Definition 2. *A* Discrete Time Linear Hybrid Systems *(DTLHS) $\mathcal{H}$ is a tuple (X, U, Y, N, W, Γ) such that:*

1. *X is a finite set of real and discrete* present state *variables. The set X' of* next state *variables is obtained by decorating with $'$ all variables in X.*
2. *U is a finite set of discrete* input (controllable) *variables.*
3. *Y is a finite set of discrete* output *variables.*
4. *$\Gamma = \Gamma_X \cup \Gamma_U \cup \Gamma_Y$ is a typing for all variables. Moreover, $\Gamma_{X'} = \Gamma_X$.*
5. *$N(X,U,X')$ is a bounded linear predicate defining the* transition relation *of $\mathcal{H}$.*
6. *$W(X,Y)$ is a linear predicate defining the* output relation *of $\mathcal{H}$. We require that there is always an output associated to any state, formally: $\forall x \in \Gamma_X$ $\exists y \in \Gamma_Y$ $W(x,y)$. We write $W^{-1}(y)$ the set of states that has output y. Formally, $W^{-1}(y) = \{x \in \Gamma_X \mid W(x,y)\}$.*

Observe that Γ_U and Γ_Y are bounded discrete typings for U and Y. This models the fact that software controllers can only read a finite set of discrete values and can only choose one among a finite set of actions. For this reason we only have discrete outputs. Moreover, our DTLHSs also include the model of the AD conversion (always present in our SBCS setting) via predicate W.

Definition 3. *Let $\mathcal{H}$ = (X, U, Y, N, W, Γ) be a DTLHS. The dynamics of $\mathcal{H}$ is defined by the Nondetermistic Moore Automata* $\mathrm{NMA}(\mathcal{H}) = (S, A, O, T, \Omega)$, *where: $S = \Gamma_X$, $A = \Gamma_U$, $O = \Gamma_Y$, $T(s, a, s')$ holds if and only if $N(s, a, s')$ holds, and $\Omega(s, o)$ holds if and only if $W(s, o)$ holds. A* state *x for $\mathcal{H}$ is a state x for* $\mathrm{NMA}(\mathcal{H})$ *and a* run *(or* path*) for $\mathcal{H}$ is a run for* $\mathrm{NMA}(\mathcal{H})$.

Example 2. Let T be a positive constant (time step). We define the DTLHS $\mathcal{H} = ([x_1, x_2], [u], [y_1, y_2], N, \Gamma, W)$, where $\Gamma_{x_1} = [-1, 1]$, $\Gamma_{x_2} = [0, 2]$, $\Gamma_u = \Gamma_{y_1} = \{-1, 0, 1\}$, $\Gamma_{y_1} = \{0, 1, 2\}$, and the transition relation $N(x_1, x_2, u, x_1', x_2')$ is defined by $((u = 0) \rightarrow x_1' = \frac{x_1}{2}) \wedge ((u \neq 0) \rightarrow x_1' = x_1) \wedge (x_2' = x_2 + uT)$. Finally, let the output predicate W be the rounding of the continuous variables x_1 and x_2. Formally, $W(x_1, x_2, y_1, y_2)$ is defined by $(x_1 - \frac{1}{2} \leq y_1 \leq x_1 + \frac{1}{2}) \wedge (x_2 - \frac{1}{2} \leq y_2 \leq x_2 + \frac{1}{2})$.

An output feedback control problem for a DTLHS $\mathcal{H}$ is the NMA output feedback control problem induced by the dynamics of $\mathcal{H}$.

Definition 4. *Given a DTLHS $\mathcal{H}$ = (X, U, Y, N, W, Γ) and linear predicates $I(X)$, $\Sigma(X)$, $G(X)$ the* DTLHS output feedback control problem *$(\mathcal{H}, I, \Sigma, G)$ is the NMA output feedback control problem $(\mathrm{NMA}(\mathcal{H}), I, \Sigma, G)$. Thus, a controller $K : \Gamma_Y \times \Gamma_U \rightarrow \mathbb{B}$ is a* solution *to $(\mathcal{H}, I, \Sigma, G)$ iff it is a solution to $(\mathrm{NMA}(\mathcal{H}), I, \Sigma, G)$.*

Example 3. Let $\mathcal{H}$ be the DTLHS in Ex. 2 and $X = [x_1, x_2]$. Let $I(X) = \Sigma(X) = \Gamma_X$ and $G(X) = (-\frac{1}{2} \leq x_1 \leq -\frac{1}{2}) \wedge (-\frac{1}{2} \leq x_2 \leq \frac{1}{2})$. The DTLHS output control problem $(\mathcal{H}, I, \Sigma, G)$ has the solution $K(y_1, y_2, u) = ((y_2 = 1) \rightarrow (u = 0)) \wedge ((y_2 = 2) \rightarrow (u = -1)) \wedge ((y_2 = 0) \rightarrow (u = 1))$. Observe, that this solution depends on the output variable y_2 only. As a consequence, if we consider the DTLHS $\mathcal{H}' = ([x_1, x_2], [u], [y_2], \Gamma, W')$ with the output predicate W' defined by $W(x_1, x_2, y_1, y_2) = (x_2 - \frac{1}{2} \leq y_2) \wedge (y_2 \leq x_2 + \frac{1}{2})$ (rounding of the variable x_2), we have that K is a solution also to the control problem $(\mathcal{H}', I, \Sigma, G)$.

4.1 A DTLHS Model for the Inverted Pendulum Case Study

In this section, we present the DTLHS model of the inverted pendulum, on which our experiments focus. The inverted pendulum (see Fig. 2) is a classical, hard control problem [22] whose DTLHS formulation is far from trivial [2]. The inverted pendulum is modeled by taking the angle θ and the angular velocity $\dot{\theta}$ as state variables and the torquing force $u \cdot F$ as the system input. The variable u models the direction and the constant F models the intensity of the force. Differently from [22], we consider the problem of finding a discrete controller, whose decisions can be only "apply the force clockwise" ($u = 1$), "apply the force counterclockwise" ($u = -1$)", or "do nothing" ($u = 0$). A linear model can be found by under- and over-approximating the non linear function $\sin x$ with piecewise linear functions f_i^- and f_i^+ (see [2] for details). The resulting model is the DTLHS $\mathcal{I}^b = (X, U, Y, N, W^b, \Gamma)$ discretized with b bits, where $X = \{x_1, x_2\}$ is the set of continuous state variables with $\Gamma_X = \times_{i=1}^{2}[c_{x_i}, d_{x_i}]$ (being c_{x_i}, d_{x_i} the lower and upper bound constants for variable x_i), $U = \{u\}$ is the set of input variables with $\Gamma_u = \{-1, 0, 1\}$, $Y = \{y_1, y_2\}$ is the set of

output variables (where y_1 is a discretization for x_1 and y_2 for x_2) with $\Gamma_{y_1} = \Gamma_{y_2} = \{0, \ldots, 2^b - 1\}$, and the transition relation $N(X, U, X')$ is the following linear predicate (m is the pendulum mass, l is the pendulum length, and g is the gravitational acceleration):

$$\begin{aligned}
&\exists Z \in \Gamma_Z (x'_1 = x_1 + 2\pi z_q + \tau x_2) \wedge (x'_2 = x_2 + \tau \frac{g}{l} z_{\sin} + \tau \frac{1}{ml^2} uF) \\
&\quad \wedge \textstyle\bigwedge_{i \in [4]} z_i \rightarrow f_i^-(z_\alpha) \leq z_{\sin} \leq f_i^+(z_\alpha) \\
&\quad \wedge \textstyle\bigwedge_{i \in [4]} z_i \rightarrow z_\alpha \in I_i \wedge \sum_{i \in [4]} z_i \geq 1 \\
&\quad \wedge x_1 = 2\pi z_k + z_\alpha \ \wedge \ -\pi \leq x'_1 \leq \pi \ \wedge \ X \in \Gamma_X \ \wedge \ U \in \Gamma_U
\end{aligned}$$

Finally, the output predicate is $W^b(x_1, x_2, y_1, y_2) \equiv \bigwedge_{i=1}^{2} c_{x_i} + \frac{d_{x_i} - c_{x_i}}{2^b} y_i \leq x_i \leq c_{x_i} + \left(\frac{d_{x_i} - c_{x_i}}{2^b} + 1 \right) y_i \wedge y_i \in \Gamma_{y_i}$.

5 On-the-Fly Control Software Synthesis

Given a DTLHS output control problem $\mathcal{P} = (\mathcal{H}, I, \Sigma, G)$, a typical approach to the automatic synthesis of controllers consists of building a suitable finite state representation $\hat{\mathcal{H}}_\Sigma$ of the plant $\mathcal{H}$, computing an abstraction $\hat{I}$ (resp. $\hat{G}$) of the initial (resp. goal) region I (resp. G) so that any solution to the control problem $(\hat{\mathcal{H}}_\Sigma, \hat{I}, \hat{G})$ is a finite representation of a solution to $\mathcal{P}$. For example, this can be done by giving conditions ensuring that the abstract system satisfies some equivalence relation with respect to the concrete system (e.g. see [33, 1, 25]).

To avoid useless computation, our on-the-fly control synthesis algorithm (Sect. 5.2) simultaneously computes the finite abstraction $\hat{\mathcal{H}}_\Sigma$ and the solution to the control problem $(\hat{\mathcal{H}}_\Sigma, \hat{I}, \hat{G})$. To make the algorithm description clear, we first present in Sect. 5.1 the notion of *output abstraction* that adapts the notion of control abstraction [25] to the output model considered in this paper.

5.1 Output Abstraction

In our setting [25], the finite state representation induced by the output relation of a DTLHS is a design constraint rather than a methodological tool, since it models the finite precision of sensor measurements.

Definition 5. *Let $\mathcal{H} = (X, U, Y, N, W, \Gamma)$ be a DTLHS and $(\mathcal{H}, I, \Sigma, G)$ be a DTLHS control problem. The* output abstraction *of $\mathcal{H}$ is the LTS $\hat{\mathcal{H}}_\Sigma = (S, A, T_\Sigma)$ such that $S = \Gamma_Y$, $A = \Gamma_U$, and for all $s, s' \in S$, $a \in A$ we have $T_\Sigma(y, a, y')$ iff a is an admissible transition in y and there exists $x, x' \in \Gamma_X$ such that $W(x, y) \wedge W(x', y') \wedge N(x, a, x')$.*

The output abstraction could be a highly non-deterministic LTS, thus making problematic the existence of a strong solution to the output feedback control problem. In particular, for small values of the sampling time, the output abstraction may contain a large number of self-loops: for any output y that is not in the goal region, a self-loop (y, a, y) of $\hat{\mathcal{H}}_\Sigma$ prevents the action a to be enabled in y in any strong solution to the output control problem. On the other hand, if by repeatedly performing an action a in an abstract state y, it is guaranteed that the system will leave the region $W^{-1}(y)$ represented by the output y after a finite number of steps, a self-loop (y, a, y) of $\hat{\mathcal{H}}_\Sigma$ can be eliminated and the action a can be enabled by a strong controller in the state y.

Definition 6. *Let $\mathcal{H} = (X, U, Y, N, W, \Gamma)$ be a DTLHS, $(\mathcal{H}, I, \Sigma, G)$ be a DTLHS control problem and let $\hat{\mathcal{H}}_\Sigma = (S, A, T_\Sigma)$ be its output abstraction.*

A self-loop (y, a, y) of $\hat{\mathcal{H}}_\Sigma$ is non-eliminable *if there exists at least an infinite run $\pi = x_0 a x_1 a x_2 \ldots$ in $\mathcal{H}$ such that $\forall t \in \mathbb{N}\ x_t \in W^{-1}(y)$. Otherwise, a self-loop (y, a, y) of $\hat{\mathcal{H}}_\Sigma$ is said to be an* eliminable self-loop.

We call adequate output abstraction *any LTS $\hat{\mathcal{H}}' \sqsubseteq \hat{\mathcal{H}}_\Sigma$ that omits some eliminable self-loops.*

Example 4. Let $\mathcal{P} = (\mathcal{H}, I, \Sigma, G)$ be the control problem in Ex. 3. An adequate output abstraction of $\mathcal{H}$ is the automaton considered in Ex. 1. Observe that, for all $z \in \Gamma_{y_2}$, the self-loops $((0, z), 0, (0, z))$ are non-eliminable self-loops. In fact, $N((0, z), 0, (0, z))$ holds, and hence there are runs of $\mathcal{H}$ which infinitely cycle on $(0, z)$ with action 0. Thus self-loops $((0, z), 0, (0, z))$ belong to the output abstraction and to all adequate output abstractions. On the contrary, the output abstraction contains, for all $(z_1, z_2) \in \Gamma_Y$, self-loops $((z_1, z_2), 1, (z_1, z_2))$ and $((z_1, z_2), -1, (z_1, z_2))$, as well as self-loops $((z_1, z_2), 0, (z_1, z_2))$ where $z_1 \neq 0$. It is easy to see that all such self-loops are eliminable, thus adequate output abstractions (as the one in Ex. 1) may not contain them. Finally, observe that, for all $z_1 \in \Gamma_{y_1}$, action 1 is not admissible in $(z_1, 2)$, since for example $N((z_1, 2), 1, (z_1, 2+T))$ holds and $\Sigma((z_1, 2+T))$ does not hold. Similarly, for all $z_1 \in \Gamma_{y_1}$, action -1 is not admissible in $(z_1, 0)$.

The following theorem [25] states that it is correct to consider output adequate abstractions when looking for a strong solution to a output feedback DTLHS control problem.

Theorem 1. *Let $\mathcal{H} = (X, U, Y, N, W, \Gamma)$ be a DTLHS, let $(\mathcal{H}, I, \Sigma, G)$ be an output feedback DTLHS control problem, and let $\hat{\mathcal{H}}_\Sigma$ be an adequate abstraction of $\mathcal{H}$. If $\hat{I}, \hat{G} \subseteq \Gamma_Y$ are such that $I \subseteq W^{-1}(\hat{I})$ and $G \supseteq W^{-1}(\hat{G})$, then a strong solution $\hat{K}$ to the LTS control problem $(\hat{\mathcal{H}}_\Sigma, \hat{I}, \hat{G})$ is a strong solution to the output feedback control problem $(\mathcal{H}, I, \Sigma, G)$.*

5.2 On-the-Fly computation of output abstraction

Stemming from Theorem 1, the solution of a output control problem $(\mathcal{H}, I, \Sigma, G)$ can be found as the solution to the finite LTS control problem $(\hat{\mathcal{H}}_\Sigma, \hat{I}, \hat{G})$. In [25], we presented a MILP-based approach to the computation of the output abstraction $\hat{\mathcal{H}}_\Sigma$. The solution to the finite LTS control problem is computed by adapting the symbolic algorithm in [14]. Starting from goal states, the most general optimal controller is found looping backward, adding at each step to the set of states D controlled so far, the *strong preimage* of D, i.e. the set of states for which there exists at least an action a that drives the system to D, regardless of possible nondeterminism.

In order to determine as soon as possible if a solution to a given output control problem cannot be found, and actually compute the solution otherwise, Alg. 1 implements an incremental approach to control software synthesis, in the same spirit of *on-the-fly Model Checking* [19]. Instead of first fully computing $\hat{\mathcal{H}}_\Sigma$, and then solving the finite LTS control problem $(\hat{\mathcal{H}}_\Sigma, \hat{I}, \hat{G})$, function *strongCtrInc*

incrementally and simultaneously computes the abstraction $\hat{\mathcal{H}}_\Sigma$ and the solution $\hat{K}$ to the control problem $(\hat{\mathcal{H}}_\Sigma, \hat{I}, \hat{G})$ in such a way that, at the i-th iteration, the computed abstraction $\hat{\mathcal{H}}_i$ is large enough to correctly determine the set of states that can be driven to the goal in at most i steps.

Function *strongCtrIncr* in Alg. 1. uses *Ordered Binary Decision Diagrams* (OBDD) to represent sets and relations over sets. In Alg. 1, variable $\hat{K}$ is the OBDD representing the computed controller so far, $\hat{D}$ is the domain of $\hat{K}$, $\hat{F} \subseteq \hat{D} \cup \hat{G}$ is the set of outputs which have been added to $\hat{D}$ in the last iteration, and $\hat{N}$ is the transition relation of $\hat{\mathcal{H}}_\Sigma$ computed so far. To save useless computation, the OBDD $\hat{E}$ stores the set of pairs $(y, u) \in \Gamma_Y \times \Gamma_U$ already considered in the construction of $\hat{N}$.

Algorithm 1 Incremental Controller Synthesis

Input: A DTLHS $\mathcal{H} = (X, U, Y, N, W, \Gamma)$, a control problem $(\mathcal{H}, I, \Sigma, G)$.
function *strongCtrInc*$(\mathcal{H}, I, \Sigma, G)$
1. $\hat{G} \leftarrow \{y \in \Gamma_Y | \neg\exists x \in \Gamma_X . W(x, y) \wedge \neg G(x)\}$
2. $\hat{I} \leftarrow \{y \in \Gamma_Y | \exists x \in \Gamma_X . W(x, y) \wedge I(x)\}$
3. $\hat{K} \leftarrow \varnothing$; $\hat{D} \leftarrow \varnothing$; $\hat{N} \leftarrow \varnothing$; $\hat{F} \leftarrow \hat{G}$; $\hat{E} \leftarrow \varnothing$
4. **repeat**
5. **for all** $y \in \hat{F}$, $u \in \Gamma_U$ **do**
6. $\hat{P} \leftarrow$*overCounterImage*(y, u)
7. **for all** $\tilde{y} \in \hat{P}$ **do**
8. **if** $(\tilde{y}, u) \notin \hat{E}$ **then**
9. $\hat{E} \leftarrow \hat{E} \cup \{(\tilde{y}, u)\}$ {mark $(\tilde{y}, u)$ as "examined"}
10. **if** *admissible*$(\Sigma, \tilde{y}, u)$ **then**
11. **if** *selfLoop*$(\tilde{y}, u)$ **then** $\hat{N} \leftarrow \hat{N} \cup \{(\tilde{y}, u, \tilde{y})\}$
12. $\hat{O} \leftarrow$ *overImg*$(\tilde{y}, u)$
13. **for all** $\tilde{y}' \in \hat{O}$ **do**
14. **if** $\tilde{y} \neq \tilde{y}' \wedge$ *existsTrans*$(\tilde{y}, u, \tilde{y}')$ **then**
15. $\hat{N} \leftarrow \hat{N} \cup \{(\tilde{y}, u, \tilde{y}')\}$
16. $\hat{C}_{\text{new}} \leftarrow \{(y, u) \mid y \notin \hat{D}, \exists s' \, \hat{N}(y, u, y') \wedge \forall y' \, \hat{N}(y, u, y') \Rightarrow y' \in \hat{D} \cup \hat{G}\}$
17. $\hat{K} \leftarrow \hat{K} \cup \hat{C}_{\text{new}}$; $\hat{F} \leftarrow \{y \mid (y, u) \in \hat{C}_{\text{new}}\}$; $\hat{D} \leftarrow \hat{D} \cup \hat{F}$
18. **until** $C_{\text{new}} = \varnothing$
19. **if** $\hat{I} \subseteq \hat{D}$ **then return** $\langle$TRUE, $\hat{D}, \hat{K}\rangle$
20. **else return** $\langle$FALSE, $\hat{D}, \hat{K}\rangle$

Function *strongCtrIncr* first computes a finite underapproximation $\hat{G}$ of the goal region G (line 1), and a finite overapproximation $\hat{I}$ of the initial region I (line 2). Then, in line 3, the controller $\hat{K}$, the controllable region $\hat{D}$, the set $\hat{E}$, and the transition relation $\hat{N}$ are initialized to the empty set (i.e. the empty OBDD) and $\hat{F}$ is initialized to the set of abstract goal states $\hat{G}$.

After this initialization phase, function *strongCtrIncr* enters a loop (lines 4–18) in which, at iteration i, all states which may be strongly controlled in at most i steps are added to $\hat{K}$. To this aim, a nested loop (lines 5–15) is performed where, at each iteration, the algorithm computes the part of the transition relation $\hat{N}$ that is necessary to find all states that a controller can drive in one step to the controllable region $\hat{D}$ computed so far. To this end, for any output $y \in \hat{F}$ and for any action u, it is computed an overapproximation $\hat{P}$ of the set of outputs that

can reach y in one step by performing action u (line 6). The overapproximation $\hat{P}$ is computed by function *overCounterImg* which, for each variable $y_i \in Y$, computes the minimum and maximum value that y_i can assume in a satisfying assignment of $N(x, a, x') \wedge W(x, y) \wedge W(x', y')$ (thus $2|Y|$ MILP problems are set up and solved). Since the set $\hat{E}$ contains all the output-action pairs already considered in the construction of $\hat{N}$ so far, to avoid the same part of $\hat{N}$ to be recomputed, only state-action pairs not in $\hat{E}$ will be considered (line 8).

As prescribed by the definition of adequate output abstraction, a transition (y, u, y'), with $y \neq y'$, is added to $\hat{N}$ whenever u is an admissible action in y and there exist $x \in W^{-1(y)}, x' \in W^{-1}(y')$ such that $N(x, u, x')$ (lines 10–15). As for self-loops (y, u, y), we want to add them to $\hat{N}$ only if they are non-eliminable (line 11). Since self-loop elimination is an undecidable problem [29], we employ function *selfLoop* [25] to check a sufficient gradient based condition for self-loop elimination that in practice turns out to be very effective. Namely, for each variable x_i, *selfLoop* tries to establish if x_i is either always increasing or always decreasing inside $W^{-1}(y)$ by performing action u. If this is the case, we have that, being $W^{-1}(y)$ a compact set, no Zeno-phenomena may arise, thus executing action u it is guaranteed that $\hat{\mathcal{H}}_\Sigma$ will eventually leave the region $W^{-1}(y)$.

Lines 16–17 update the controller $\hat{K}$ (and its domain $\hat{D}$) computed so far. The set $\hat{F}$ is updated with the set of new controlled states. Finally, the outermost **repeat-until** loop (lines 4–18) is performed until no more new controlled states have been found.

Theorem 2. *Let $\mathcal{P} = (\mathcal{H}, I, \Sigma, G)$ be a DTLHS output feedback control problem. If function strongCtrInc returns $\langle \textsc{True}, \hat{D}, \hat{K} \rangle$ then $\hat{K}$ is a strong solution to $\mathcal{P}$.*

Finally, the actual control software (i.e., C code) for the DTLHS is synthesized by translating $\hat{K}$ as it is described in [28]. The *guaranteed WCET* (worst case execution time) $T_{\hat{K}}$ of the synthesized control software is also computed.

6 Experimental Results

In this section we present our experiments that aim at evaluating the effectiveness of our control software synthesis technique. We implemented *strongCtrInc* in the C programming language using the CUDD package for OBDD based computations and GLPK for solving MILP problems. The resulting tool, QKS^{otf}, extends the tool *QKS* by adding the possibility of using the on-the-fly approach described in Alg. 1.

The objective of our experiments is threefold. First, in Sect. 6.1 and 6.2 we evaluate, on a meaningful case study, the speedup obtained with the on-the-fly algorithm with respect to the exhaustive method presented in [25] in the context of design space exploration. Second, in Sect. 6.3 we show how our on-the-fly algorithm can be used for realizability and schedulability analysis issues [11] for control software in design space exploration. Finally, in Sect. 6.4 we assess the quality of our controllers, by evaluating their system level performances, such as ripple and set-up time.

6.1 Experimental Setting: Design Space Exploration

In our experiments, we consider the inverted pendulum case study introduced in Sect. 4.1. To this aim, we model the inverted pendulum with the DTLHS $\mathcal{I}^b = (X, U, Y, N, W^b, \Gamma)$ defined in Sect. 4.1, where the state variables bounds are fixed as follows: $c_{x_1} = -1.1\pi$ radiants, $d_{x_1} = 1.1\pi$ radiants, $c_{x_2} = -4$ radiants per second, $d_{x_2} = 4$ radiants per second. As for pendulum parameters, we set $F = 0.5$ N and, as in [22, 2, 3], we set l and m in such a way that $\frac{g}{l} = 1$ (i.e. $l = g$) and $\frac{1}{ml^2} = 1$ (i.e. $m = \frac{1}{l^2}$). Finally, the DTLHS control problem is $(\mathcal{I}^b, \Sigma, I, G)$, where $I(x_1, x_2) \equiv \bigwedge_{i=1}^{2} 0.9c_{x_i} \leq x_i \leq 0.9d_{x_i}$, $G(x_1, x_2) \equiv \bigwedge_{i=1}^{2} 0.1 \leq x_i \leq 0.1$, and $\Sigma(x_1, x_2) \equiv \bigwedge_{i=1}^{2} x_i \in \Gamma_{x_i}$. That is, the goal is to turn the pendulum nearly steady to the upright position, starting from nearly any possible initial position and without going out of the state variables bounds.

Our aim here is to carry out experiments for different values of the number of quantization bits b and of the *sampling time* T, i.e., the time between two samples of the system state in the closed loop system. On the other hand, the DTLHS $\mathcal{I}^b$ approximates the continuous time pendulum dynamics by discretizing the corresponding differential equations with a time step τ ($\tau = 0.05$ seconds in our experiments). T is typically greater than τ. If we directly set $\tau = T$ in $\mathcal{I}^b$, we would obtain a not accurate model, since τ depends on physical considerations [36] (such considerations are not our focus here). Building on this, we approximate the dynamics of the pendulum with sampling time T by iterating $n = \lceil \frac{T}{\tau} \rceil$ times the transition relation N of $\mathcal{I}^b$. Namely, we consider the transition relation $N_n(X, U, X') \equiv \exists \tilde{X}^{(0)}, \ldots, \tilde{X}^{(n)} \bigwedge_{i=0}^{n-1} N(\tilde{X}^{(i)}, U, \tilde{X}^{(i+1)}) \wedge X = \tilde{X}^{(0)} \wedge X' = \tilde{X}^{(n)}$, being $\tilde{X}^{(0)}, \ldots, \tilde{X}^{(n)}$ sets of variables not occurring in N (note that N_n is a linear predicate). Namely, $N_n(x, u, x')$ holds if, by holding action u for n transitions of step τ, the systems goes from x to x'. This allows us to have a sampling time (at least) T, while retaining model accuracy. In the following, we will use n instead of T, with the understanding that $T = n\tau$. Thus, the DTLHS reference model for our experiments is $\mathcal{I}_n^b = (X, U, Y, N_n, W^b, \Gamma)$, and the DTLHS control problem is $(\mathcal{I}_n^b, I, \Sigma, G)$.

In order to experimentally show that function *strongCtrInc* of Alg. 1 effectively supports design space exploration, we will run both QKS^{otf} and QKS on $\mathcal{I}_n^b$ for $(b, n) \in \{8, 9, 10\} \times \{10, 8, 6, 4, 2, 1\}$, and then compare the corresponding computation times.

6.2 Experimental Results for Design Space Exploration

All experiments have been carried out on an Intel(R) Xeon(R) CPU @ 2.27GHz, with 23GiB of RAM, Kernel: Linux 2.6.32-5-686-bigmem, distribution Debian GNU/Linux 6.0.3 (squeeze).

Results of QKS and QKS^{otf} are in Table 1. Columns meaning in Table 1 are as follows. Columns b and n have the same meaning as in Sect. 6.1. Columns CPU^{exh} (resp., CPU^{otf}) shows the computation time in seconds of QKS (resp., QKS^{otf}). Columns RAM^{exh} (resp., RAM^{otf}) shows the RAM memory usage peak in bytes for QKS (resp., QKS^{otf}). Column $|\hat{K}|$ shows the generated controller size, i.e. the number of nodes in the OBDD representing $\hat{K}$. Column **Speedup** shows the speedup obtained by using QKS^{otf} instead of QKS, that is $\frac{\text{CPU}^{\text{exh}}}{\text{CPU}^{\text{otf}}}$.

Table 1. Experimental results for pendulum

b	n	CPU^{exh}	RAM^{exh}	CPU^{otf}	RAM^{otf}	$\lvert\hat{K}\rvert$	%	**Speedup**	**Result**
8	10	9.90e+04	1.70e+08	4.58e+02	3.03e+07	1.25e+02	99.54	216.16	FAIL
8	8	4.41e+04	1.68e+08	3.06e+02	3.05e+07	2.06e+02	99.31	144.12	FAIL
8	6	2.28e+04	1.65e+08	2.77e+04	9.12e+07	6.40e+03	-21.49	0.82	PASS
8	4	1.17e+04	1.63e+08	1.47e+04	8.68e+07	7.53e+03	-25.64	0.80	PASS
8	2	4.91e+03	1.63e+08	1.35e+01	2.98e+07	1.63e+02	99.73	363.70	FAIL
8	1	2.69e+03	1.53e+08	4.72e+00	2.98e+07	1.61e+02	99.82	569.92	FAIL
9	10	4.95e+05	2.39e+08	2.70e+03	3.16e+07	1.88e+02	99.45	183.33	FAIL
9	8	2.31e+05	2.31e+08	2.40e+05	2.70e+08	1.08e+04	-3.90	0.96	PASS
9	6	1.20e+05	2.18e+08	1.19e+05	2.71e+08	1.25e+04	0.83	1.01	PASS
9	4	5.66e+04	1.98e+08	5.34e+04	2.50e+08	1.55e+04	5.65	1.06	PASS
9	2	2.18e+04	1.91e+08	2.29e+04	2.43e+08	2.16e+04	-5.05	0.95	PASS
9	1	1.16e+04	1.78e+08	1.97e+01	3.02e+07	2.11e+02	99.83	588.83	FAIL
10	10	3.82e+06	6.08e+08	1.45e+04	3.65e+07	2.87e+02	99.62	263.45	FAIL
10	8	1.71e+06	5.40e+08	6.74e+03	3.83e+07	6.01e+02	99.61	253.71	FAIL
10	6	7.45e+05	4.72e+08	6.67e+05	8.81e+08	2.45e+04	10.47	1.12	PASS
10	4	3.05e+05	4.13e+08	2.77e+05	8.31e+08	2.99e+04	9.18	1.10	PASS
10	2	1.05e+05	3.29e+08	9.96e+04	8.12e+08	4.52e+04	5.14	1.05	PASS
10	1	5.29e+04	2.64e+08	5.09e+04	8.07e+08	6.31e+04	3.78	1.04	PASS
Overall		7.85e+06	6.08e+08	1.60e+06	8.81e+08		79.62	4.91	

Column % shows the gain (in terms of computation time) obtained by using QKS^{otf} instead of QKS, that is $\% = 100(1 - \frac{\text{CPU}^{\text{exh}} - \text{CPU}^{\text{otf}}}{\text{CPU}^{\text{exh}}})$. Column **Result** is PASS if a controller for $\mathcal{I}_n^b$ exist (i.e., if function *strongCtrInc* returns TRUE), FAIL otherwise. Finally, the last row in Table 1 shows the sum of all computation times for QKS and QKS^{otf}, the maximum RAM memory usage peak for QKS and QKS^{otf}, and the overall computation time gain of QKS^{otf} w.r.t QKS.

From Table 1 we note that, as expected, QKS^{otf} obtain a huge speedup (near to 100%) for the cases in which a control software is not found, while it requires approximately the same time of QKS otherwise. This is due to the fact that the on-the-fly algorithm introduces both an overhead (mainly due to counterimages computations at line 6 of Alg. 1 and OBDD $\hat{E}$ management) and a speedup (even when the control software is found, the adequate output abstraction $\hat{N}$ may be not fully computed). Summing up, our approach obtain an overall gain of nearly 80% when performing design space exploration, with an acceptable memory usage overhead. This shows effectiveness of QKS^{otf} for design space exploration.

6.3 Control software realizability and schedulability

In order to verify if the control software works properly on a given microcontroller, two issues must be taken into account: *realizability* and *schedulability*.

A control software is *realizable* on a given microcontroller if the whole control software fits in the microcontroller flash memory. Since our approach directly outputs the C code for the control software, it is sufficient to compile the C code on the given microcontroller architecture, obtain the hex file to be copied on the microcontroller flash, and check if its size fits in the microcontroller flash.

As for schedulability, we note that the real-time requirement $T_W \leq T = n\tau$ must hold, being T_W an upper bound for the control software WCET. Since

our approach also outputs the synthesized control software guaranteed WCET, we are able to directly check if this requirement is fulfilled. Namely, since $2b$ (resp. 2) bits are needed to encode pendulum states (resp. actions), in all our experiments the WCET is $T_W \leq 4bT_B$, being T_B an upper bound for the time needed to compute an `if-then-else` C block of a given known structure [28]. More in detail, by directly looking at the assembly code generated for such an `if-then-else` C block on a candidate microcontroller (an example is shown in Fig. 3), and by considering the number of clock cycles needed for each assembly instruction, we obtain the upper bound for the number of microcontroller clock cycles A needed to compute such a block. Thus, given the microcontroller frequency $F = \frac{1}{T_C}$, we have that $T_B \leq AT_C$.

In order to complete the schedulability analysis of the control software, we need to consider that other *processes* need to run with given periods together with the controller itself. Namely, the controller computation (which in this setting is a process with period $n\tau$) must be preceded by processes reading quantized values from plant sensors (one process per plant state variable) and must be followed by a process sending the computed action to plant actuators. Moreover, other processes may be needed, e.g. to accept keyboard input for debugging. In the following, we will assume each of such processes to require at most 100 clock cycles, and to have a period of 10^{-3} seconds (which is less than $n\tau$ for all n). In order to determine beforehand (i.e., without having to actually copy the control software in the microcontroller and test it) if all such periods may be met in the given microcontroller architecture, we employ the schedulability test for the *Rate-Monotonic Scheduling* (RMS, see e.g. [11]), that is $\sum_{i=1}^{k+1} \frac{C_i}{T_i} \leq (k+1)(2^{1/(k+1)} - 1)$, being C_i the WCET and T_i the period for process i and k the number of processes running together with the controller. Supposing the controller to be the process with index $(k+1)$, we have that the schedulability test is implied by $\frac{4bAT_C}{n\tau} + k\frac{100T_C}{10^{-3}} < 0.69$.Again, being all the required measures either known or computed by our model-based approach, we are able to determine beforehand if the control software is schedulable in the given microcontroller.

Our experimental results on control software schedulability and realizability are shown in Table 2. Columns meaning in Table 2 are as follows. Columns b and n have the same meaning as in Sect. 6.1. Column $|\hat{K}_{\text{hex}}|$ shows the generated controller size, as the number of bytes to be written in the target microcontroller flash memory. Column **Arch** shows the microcontroller having the smallest fit flash memory for $|\hat{K}_{\text{hex}}|$. Namely, we consider the following microcontrollers of the Atmel family [32]: atmega8 (8K of flash), atmega16 (16K of flash) and at91sam (1MB of flash). For both atmega8 and atmega16, the clock frequency F is 4MHz (i.e., each clock tick needs $T_C = 250$ nanoseconds), and the upper bound of the number of clock cycles needed to compute the greatest `if-then-else` C block in the software implementing $\hat{K}$ is $A = 16$. For at91sam, which, being ARM-based, is shown as ARM in Table 2, $F = 50$ MHz, $T_C = 250$ nanoseconds and $A = 12$. Column **WCET** shows an upper bound for the control software WCET, i.e., $4bAT_C$. Column α shows the ratio between the WCET and the period of the controller process (note that this is part of the schedulability test for RMS), i.e., $\alpha = \frac{\text{WCET}}{n\tau}$. Let β be an upper bound for the ratio

Table 2. Experimental results for realizability and schedulability

b	n	$\lvert\hat{K}_{\text{hex}}\rvert$	Arch	WCET	α	k
8	10	5.00e+03	atmega8	3.20e-04	6.40e-04	27
8	8	7.39e+03	atmega8	2.56e-04	6.40e-04	27
8	6	1.45e+05	atmega16	1.92e-04	6.40e-04	27
8	4	1.74e+05	atmega16	1.28e-04	6.40e-04	27
8	2	4.85e+03	atmega8	6.40e-05	6.40e-04	27
8	1	4.31e+03	atmega8	3.20e-05	6.40e-04	27
9	10	7.66e+03	atmega8	3.60e-04	7.20e-04	27
9	8	2.37e+05	atmega16	2.88e-04	7.20e-04	27
9	6	2.80e+05	atmega16	2.16e-04	7.20e-04	27
9	4	3.37e+05	atmega16	1.44e-04	7.20e-04	27
9	2	9.50e+05	ARM	4.32e-06	4.32e-05	344
9	1	5.98e+03	atmega8	3.60e-05	7.20e-04	27
10	10	1.20e+04	atmega8	4.00e-04	8.00e-04	27
10	8	2.18e+04	atmega8	3.20e-04	8.00e-04	27
10	6	1.06e+06	ARM	1.44e-05	4.80e-05	344
10	4	1.31e+06	ARM	9.60e-06	4.80e-05	344
10	2	1.96e+06	ARM	4.80e-06	4.80e-05	344
10	1	2.63e+06	ARM	2.40e-06	4.80e-05	344

```
.L398:
    ldd r24,Z+10
    cpi r24,lo8(1)
    brne .L17
.L37:
    ld r24,Z
    cpi r24,lo8(1)
    breq .L17
    ldi r24,lo8(0)
    ldi r25,hi8(0)
    or r18,r19
    brne .L38
    ldi r24,lo8(1)
    ldi r25,hi8(1)
.L38:
    movw r18,r24
.L39:
    ldd r24,Z+9
    rjmp .L440
.L35:
    ldi r18,lo8(0)
    ldi r19,hi8(0)
```

Fig. 3. Snapshot of Atmel atmega16 assembly control software.

between WCET and period for all other possible processes as computed in our strengthened RMS schedulability test, i.e., $\beta = 0.69 - \alpha$ ($\beta \approx 0.69$ in all cases of Table 2). Column k shows a lower bound for the maximum number of processes which may be run together with the controller on the given microcontroller, under the hypothesis that each process requires 100 clock cycles and has a period of 10^{-3} seconds. Namely, following again the RMS schedulability test, $k = \lfloor \frac{10^{-3}\beta}{100 T_C} \rfloor$. Note that k must be at least 3 for the inverted pendulum case study, since 2 processes are required to read the quantized value plant state from sensors and a third process is needed to send the computed action to the actuators. Indeed, in all cases we have $k \geq 27$.

Summing up, our on-the-fly approach allows us to directly obtain the final microcontroller implementation, by using a *model-based* methodology.

6.4 Control Software Performances

For the sake of completeness, though it is not the scope of our paper, we evaluate performances of the generated control software for different values of b and n. Namely, we simulate $\mathcal{I}_n^{b\,(\hat{K})}$, that is the pendulum closed loop system. In order to show impact of parameter n, in Figs. 4 and 5, we show simulations (on setup time and ripple) for a fixed value of b (namely, $b = 10$) and for $n \in \{1, 6\}$. Finally, in order to show impact of parameter b, in Figs. 6 and 7, we show simulations (on setup time and ripple) for a fixed value of n (namely, $n = 6$) and for $b \in \{8, 10\}$.

7 Conclusion

In this paper, we address correct-by-construction control software synthesis from Formal System Level Specifications for Discrete Time Linear Hybrid Systems. Since in our approach the control software has a WCET known in advance, a

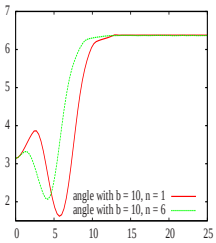

Fig. 4. Pendulum setup for $b = 10$, $n \in \{1, 6\}$ (angle x_1 is shown, time is in seconds)

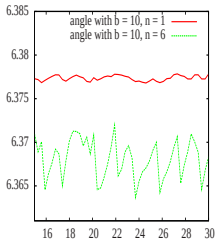

Fig. 5. Pendulum ripple for $b = 10$, $n \in \{1, 6\}$ (angle x_1 is shown, time is in seconds)

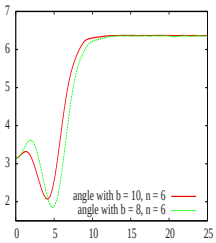

Fig. 6. Pendulum setup for $n = 6$, $b \in \{8, 10\}$ (angle x_1 is shown, time is in seconds)

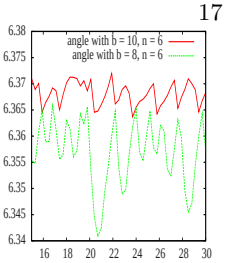

Fig. 7. Pendulum ripple for $n = 6$, $b \in \{8, 10\}$ (angle x_1 is shown, time is in seconds)

concrete schedulability analysis can be easily carried out. We present an on-the-fly algorithm for control software synthesis that detects as soon as possible if it can not find a solution to a given control problem. This property turns out to be very useful in design space exploration. Looking for an optimal choice of design parameter, it is typical to try to solve control software synthesis problems that do not have a solution. As confirmed by our experimental results, our algorithm effectively supports design space exploration. On the inverted pendulum benchmark, using our on-the-fly algorithm we get a time saving of about 80% with respect to an exhaustive approach.

References

1. M. Agrawal and P. S. Thiagarajan. The discrete time behavior of lazy linear hybrid automata. In *HSCC*, LNCS 3414, pp. 55–69, 2005.
2. V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci. Automatic control software synthesis for quantized discrete time hybrid systems. In *CDC*, 2012.
3. V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci. On model based synthesis of embedded control software. In *EMSOFT*, pp. 227–236, 2012.
4. 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. *TCS*, 138(1):3 – 34, 1995.
5. R. Alur. Formal verification of hybrid systems. In *EMSOFT*, pp. 273–278, 2011.
6. T. Basten, E. Benthum, M. Geilen, M. Hendriks, F. Houben, G. Igna, F. Reckers, S. Smet, L. Somers, E. Teeselink, N. Trcka, F. Vaandrager, J. Verriet, M. Voorhoeve, and Y. Yang. Model-driven design-space exploration for embedded systems: The octopus toolset. In LNCS 6415, 2010.
7. A. Bemporad. Hybrid Toolbox, 2004. http://cse.lab.imtlucca.it/ bemporad/hybrid/toolbox/.
8. A. Bemporad and N. Giorgetti. A sat-based hybrid solver for optimal control of hybrid systems. In *HSCC*, LNCS 2993, pp. 126–141, 2004.
9. M. Benerecetti, M. Faella, and S. Minopoli. Revisiting synthesis of switching controllers for linear hybrid systems. In *CDC-ECC*, pp. 4753 –4758, 2011.
10. William L. Brogan. *Modern control theory (3rd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.
11. Giorgio C. Buttazzo. *Hard Real-Time Computing Systems (Third Edition)*. Springer, 2011.
12. 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*, pp. 66–80, 2005.
13. G. Castiglione, A. Restivo, and M. Sciortino. Nondeterministic moore automata and brzozowski's algorithm. In *CIAA 2011*, LNCS 6807, pp. 88–99, 2011.

14. A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. In *AIPS*, pp. 36–43, 1998.
15. Arvind Easwaran, Insup Lee, Insik Shin, and Oleg Sokolsky. Compositional schedulability analysis of hierarchical real-time systems. In *ISORC*, pp. 274–281, 2007.
16. G. Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *Int. J. Softw. Tools Technol. Transf.*, 10(3):263–279, 2008.
17. K. G. Larsen, P. Pettersson, and Wang Yi. UPPAAL: Status & Developments. In *CAV*, pp. 456–459, LNCS 1254, 1997.
18. T. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. In *ICALP*, pp. 582–593, 1997.
19. G. J. Holzmann. The spin model checker. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.
20. http://www.mathworks.com. Simulink by mathworks.
21. S. Jha, S. A. Seshia, and A. Tiwari. Synthesis of optimal switching logic for hybrid systems. In *EMSOFT*, pages 107–116. ACM, 2011.
22. G. Kreisselmeier and T. Birkhölzer. Numerical nonlinear regulator design. *IEEE Trans. on on Automatic Control*, 39(1):33–46, 1994.
23. R. Lublinerman, C. Szegedy, and S. Tripakis. Modular code generation from synchronous block diagrams: modularity vs. code size. In *POPL*, pp. 78–89, 2009.
24. R. Majumdar, I. Saha, and M. Zamani. Performance-aware scheduler synthesis for control systems. In EMSOFT 2011, pp. 299–308.
25. F. Mari, I. Melatti, I. Salvo, and E. Tronci. Model based synthesis of control software from system level formal specifications. *ACM Trans. Softw. Eng. Methodol.*, 2013. To appear. A preliminary version is available at `http://arxiv.org/pdf/1107.5638v2`.
26. F. Mari, I. Melatti, I. Salvo, and E. Tronci. Synthesis of quantized feedback control software for discrete time linear hybrid systems. In *CAV*, pp. 180–195, 2010.
27. F. Mari, I. Melatti, I. Salvo, and E. Tronci. Linear constraints as a modeling language for discrete time hybrid systems. In *ICSEA 2012*, pp. 664–671, 2012.
28. Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci. Synthesizing control software from boolean relations. *Int. J. on Advances in SW*, 5(3&4):212–223, 2012.
29. F. Mari, I. Melatti, I. Salvo, and E. Tronci. Undecidability of quantized state feedback control for discrete time linear hybrid systems. In *ICTAC*, LNCS 7521, pp. 243–258, 2012.
30. M. Mazo, A. Davitian, and P. Tabuada. Pessoa: A tool for embedded controller synthesis. In *CAV*, LNCS 6174, pp. 566–569, 2010.
31. M. Mazo and P. Tabuada. Symbolic approximate time-optimal control. *Systems & Control Letters*, 60(4):256–263, 2011.
32. Atmel megaAVR Microcontroller: http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx, 2013.
33. G. Pola, A. Girard, and P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.
34. Reactis White Paper: http://www.reactive-systems.com/simulink-testing-validation.html, 2013.
35. P. Roy, P. Tabuada, and R. Majumdar. Pessoa 2.0: a controller synthesis tool for cyber-physical systems. In HSCC 2011, pp. 315–316.
36. E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems. (2nd Edition)*. Springer, New York, 1998.
37. E. Tronci. Automatic synthesis of controllers from formal specifications. In *ICFEM*, pp. 134–143. IEEE, 1998.
38. H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *CDC*, pp. 4607–4612 vol. 5. IEEE, 1997.