A Hierarchical Approach for the Synthesis of Stabilizing Controllers for Hybrid Systems*

Janusz Malinowski, Peter Niebert, and Pierre-Alain Reynier

LIF, Université de Provence & CNRS, UMR 6166, France

Abstract. We consider a discretization based approach to controller synthesis of hybrid systems that allows to handle non-linear dynamics. In such an approach, states are grouped together in a finite index partition at the price of a non-deterministic over approximation of the transition relation. The main contribution of this work is a technique to reduce the state explosion generated by the discretization: exploiting structural properties of ODE systems, we propose a hierarchical approach to the synthesis problem by solving it first for sub problems and using the results for state space reduction in the full problem. A secondary contribution concerns combined safety and liveness control objectives that approximate stabilization.

1 Introduction

The model of hybrid systems constitutes a very rich modeling framework as it allows the combination of continuous and discrete-event dynamics. It is used in numerous applications such as the control of physical or chemical processes by computer programs, avionics, etc. For such systems, except under strong restrictions on the continuous dynamics, the set of reachable configurations cannot be computed exactly. As a consequence, numerous approximation techniques have been devised, with the objective of building an abstract system for which analysis is possible. The basic setting of this work is the discretization in both time and (continuous) variables of the system. In particular, we allow arbitrary dynamics, and especially nonlinear ones. To simplify the presentation, we thus focus on the continuous dynamics, and do not consider general hybrid systems, but simply systems of ODEs.

Most works related to hybrid systems are concerned with verification, and especially focus on the approximate computation of the reachability set. We tackle here the (more difficult) problem of controller synthesis. The purpose of this work is to progress in the direction of realistic controller synthesis for nonlinear ODE systems. Given a plant (an open dynamical system), controller synthesis aims at designing a system which interacts with the plant in order to satisfy a given objective. A natural setting, which subsumes standard safety and reachability objectives, consists of the design of a feedback controller which allows to *stabilize* the system around a target configuration. There are two main contributions of this work. One concerns the formalisation and algorithmic handling of a pragmatic choice of control objectives that approximate the ideal of systems stabilized under control. The other, the main objective, concerns a state space reduction

* Partially supported by the ANR project ECSPER(ANR JC09_472677 ECSPER).

approach that helps render the synthesis problem feasible despite combinatory explosion in multi-variate nonlinear dynamics: a dedicated slicing technique is introduced for controller synthesis to rapidly eliminate « hopeless » states from the search space.

The most widespread notions of games found in control applications are safety games (such as Ramadge-Wonham games) where the controller is supposed to avoid something bad from happening (by forbidding some controllable transitions), and reachability games, where the controller is supposed to drive the system into a good state within a finite amount of time. Consider the objective of stabilizing an inverted pendulum in its vertical top position. With a game based approach, we would not know how to express convergence as a goal, but for a given distance ϵ, we can state that we want the pendulum to reach a neighbourhood of radius ϵ around the desired point and to stay forever within that region. We formalize these combined until and safety objectives, and provide an efficient, on-the-fly, linear algorithm for solving such stabilization games. This algorithm is derived from model-checking algorithms for the alternation-free fragment of the propositional μ-calculus [7, 12].

State-space explosion in the number of variables is inherent in discretization techniques. To combat this problem, we propose an original hierarchical approach for the controller synthesis problem. It amounts to identifying subsets of variables of the ODE system whose dynamics is independent from all other variables. We formulize the induced relation as a bisimulation, and prove that it ensures the preservation of controllability in the subproblems w.r.t. stabilization objectives we consider. More precisely, winning states in the global problem are projected on winning states in the subproblems, which allows a strong reduction of the state space explored for the global problem. For simplicity, we only discuss ODE systems in the paper, but our work naturally applies to periodic controller synthesis for hybrid systems. A prototype implementation is presented, and experiments conducted on the inverted pendulum case study prove the vast improvement provided by this hierarchical approach.

Related work on hybrid systems in general and for the discretized approach in particular is vast. There is an obvious tradeoff between state explosion and non-determinism when discretizing hybrid systems for state space analysis. When the synthesis fails, it may not be clear whether this is due to the actual hybrid system or due to an overly coarse discretization. Discrete-state abstractions of nonlinear systems have been considered in [2, 15], and the possibility of building as rough as possible abstractions by successive abstractions has been explored [6, 1, 13]. The problem of controller synthesis for nonlinear hybrid systems is also considered in [16], but only for safety objectives. Finally note that the notion of hierarchy we consider in ODE systems in our approach is not the same as hierarchical decomposition of controllers such as in [14].

Formalization of our notion of stabilization games is presented in Section 2. We present the discretization of a nonlinear ODE system in Section 3. Section 4 contains the presentation of our hierarchical approach, and Section 5 reports experiments.

2 Controller synthesis for stabilization games

A standard way of modeling the control of synchronous systems is a two player game with alternating moves : for the duration of an interval, the controller can (deterministically) set the control parameters and the system replies at the end of the interval with a perturbed target, a non-deterministic response. The set of states thus decomposes into a bipartite graph of controllable and uncontrollable states. We fix finite sets of environment actions Σ_E, controller actions Σ_C, and atomic propositions Γ.

Definition 1 (Control Game Structure (CGS)). *A* CGS *over* $(\Sigma_E, \Sigma_C, \Gamma)$ *is a tuple* $\mathcal{C} = \langle S_E, S_C, T, S_0, \lambda\rangle$ *where* S_C *is the set of* controller states, S_E *the set of* environment states, $S := S_E \uplus S_C$*, a transition relation* $T \subseteq (S_E \times \Sigma_E \times S_C) \cup (S_C \times \Sigma_C \times S_E)$*, an initial state set* $S_0 \subseteq S$*, and* $\lambda : S \rightarrow 2^{\Gamma}$ *labels states by atomic propositions.*

We require the environment to be deadlock free, i.e. for every $s \in S_E$ there exists at least one $(s, a, s') \in T$ and we require controller actions to be deterministic, i.e. for every $s \in S_C$ and $a \in \Sigma_C$ there exists a unique $s' \in S_E$ such that $(s, a, s') \in T$. For a state $s \in S$, we let $\#Succ(s) = |\{s' \in S \mid (s, a, s') \in T \text{ for some } a \in \Sigma_E \cup \Sigma_C\}|$.

The game is turn-based and played as follows: starting from a controller state $s \in S_C$, the controller chooses an action $\sigma \in \Sigma_C$, which leads the system in a (single) environment state s' (as the controller actions are deterministic). Then, a turn of the environment in state $s' \in S_E$ consists of determining a state y such that $(s', \sigma', y) \in T$ for some $\sigma' \in \Sigma_E$. Such an interaction builds a *path* which is a finite or infinite sequence $\rho = (s_0, a_1, s_1, a_2, \ldots)$ such that $(s_k, a_{k+1}, s_{k+1}) \in T$ for every $k \geq 0$ (we do *not* require a path to begin with an initial state). To determine whether the controller or the environment wins, a *control objective* is given as a set W of (winning) paths, which defines which paths are winning for the controller.

Our work falls in the (well-studied) setting of parity games, which are memoryless determined (see for instance [9] for details). As a consequence, and to simplify the presentation, we directly focus on memoryless strategies (a.k.a controllers).

A (memoryless) *controller* is a mapping $c : S_C \rightarrow 2^{\Sigma_C}$ associating to a controller state $s \in S_C$ a non-empty set of controller actions $c(s)$. A path $s_0, a_1, s_1, a_2, \ldots$ is *controlled by* c iff for every $s_i \in S_C$ we have $a_{i+1} \in c(s_i)$. Given a state s, we say that a controller c *guarantees* a control objective W from s iff every path ρ beginning at s and controlled by c belongs to W. A state s is *winning* for control objective W iff there exists a controller c that guarantees W from s. Finally, a control game structure is winning for control objective W iff every initial state is winning for W.

We recall the widely used operator $\mathsf{CPre} : 2^S \longrightarrow 2^S$ (controllable predecessors). Intuitively, it aims at computing, given a set of target states X, the set of states from which the controller can guarantee to end up in X in one step. More formally, we define:

$$\mathsf{CPre}(X) = \{s \in S_C \mid \exists (s, a, s') \in T.s' \in X\} \cup \{s \in S_E \mid \forall (s, a, s') \in T.s' \in X\}$$

Note that if $s \in S_C$, there must exist a controllable action leading to X while if $s \in S_E$, we require that all possible successors for the environment must be in X.

2.1 Stabilization games

Basic control objectives We now consider more specific control objectives for hybrid systems with initial conditions. We define two basic control objectives:

STAY: Given a set of states $A \subseteq S$, stay forever in the set A. Formally, we define:

$$W_{\mathsf{Stay}(A)} = \{\rho = (s_0, a_1, s_1, a_2, \ldots) \mid \forall i \geq 0, s_i \in A\}$$

UNTIL: Given two sets of states $AB \subseteq S$, reach a state of B in a finite number of steps without leaving the set of allowed states A, where we require[1] that $B \subseteq A$. Formally:

$$W_{\mathsf{Until}(A,B)} = \{\rho = (s_0, a_1, s_1, a_2, \ldots) \mid \exists k \geq 0. s_k \in B \wedge \forall 0 \leq i < k, s_i \in A\}$$

$\mathsf{Stay}(\mathcal{C}, A)$ denotes the set of winning states of $\mathcal{C}$ w.r.t. the control objective $W_{\mathsf{Stay}(A)}$ and $\mathsf{Until}(\mathcal{C}, A, B)$ denotes the set of winning states w.r.t. $W_{\mathsf{Until}(A,B)}$. Intuitively, these objectives correspond to the linear time temporal logic properties GA and $A\,\mathcal{U}\,B$.

Fixpoint characterization These winning sets can be defined in terms of fixpoints of operators over sets of states. Therefore, we define the two following operators:

$$O_{\mathsf{Stay}(A)}(X) = A \cap \mathsf{CPre}(X) \tag{1}$$
$$O_{\mathsf{Until}(A,B)}(X) = B \cup (A \cap \mathsf{CPre}(X)) \tag{2}$$

Intuitively, to stay forever in A, the controller should own an action which leads him to a winning state. This explains equation (1). To characterize $\mathsf{Until}(\mathcal{C}, A, B)$ (equation (2)), one starts from sets in B and computes the least fixpoint of states from which the controller can reach such states, using again the CPre operator. Then, we obtain the following fixpoint characterizations:

$$\mathsf{Stay}(\mathcal{C}, A) = \bigcap_{n \geq 0} O^n_{\mathsf{Stay}(A)}(S) \qquad \text{and} \qquad \mathsf{Until}(\mathcal{C}, A, B) = \bigcup_{n \geq 0} O^n_{\mathsf{Until}(A,B)}(\emptyset)$$

Here, we use the notation for the n-fold application of operators : $O^0(X) = X$ and $O^{n+1}(X) = O(O^n(X))$. Note, that for the (finite) set lattice over S the approximation $\bigcup_n O^n(\emptyset)$ is equal to the least fixpoint of O, i.e. the least set S' such that $O(S') = S'$ and similarly $\bigcap_n O^n(S)$ is equal to the greatest fixpoint.

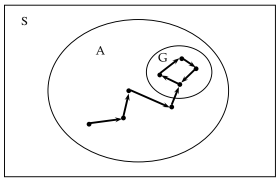
Controllers' computation For a state $s \in \mathsf{Until}(\mathcal{C}, A, B)$, we moreover define the *distance between s and B* as the least n such that $s \in O^{n+1}_{\mathsf{Until}(A,B)}(\emptyset)$. Notably, if $s \in B$ its distance from B is 0. We denote by $d(s, B)$ this value. In order to obtain controllers from these winning sets, one can proceed as follows. Let $A, B \subseteq S$. We distinguish the two objectives:

$\mathsf{Stay}(\mathcal{C}, A)$*:* for a state $s \in \mathsf{Stay}(\mathcal{C}, A) \cap S_C$, the controller has to choose any action $a \in \Sigma_C$ such that $s' \in \mathsf{Stay}(\mathcal{C}, A)$ for the unique triple (s, a, s').

$\mathsf{Until}(\mathcal{C}, A, B)$*:* for a state $s \in \mathsf{Until}(\mathcal{C}, A, B) \cap S_C$, if $s \in B$, then we do not need a strategy for this objective. Otherwise, if $s \notin B$, we must ensure progress towards the set B. This can be guaranteed if the controller chooses an action a such that the target state s' (i.e. such that $(s, a, s') \in T$) verifies $d(s', B) < d(s, B)$. By the fixpoint characterization of the set $\mathsf{Until}(\mathcal{C}, A, B)$, this is possible.

[1] The practically relevant objective is $\mathsf{Until}(A, A \cap B)$, which is equal to $\mathsf{Until}(A, B)$ under this assumption.

Stabilization objective Recall that in our setting, our objective is to synthesize a controller for a dynamical system, which, starting from an intial configuration, leads the system towards a desirable configuration. To express our stabilization objective, we start from the two following properties: Goal is a set of goal states which describes a neigbourhood around the desirable configuration, Allow is a set of allowed states, which describes the legal configurations of the system. In the sequel, we assume that the inclusion $\mathsf{Goal} \subseteq \mathsf{Allow}$ holds.



Intuitively, we are interested in synthesizing a controller which is able to guide the system from an initial state, while staying in the set Allow, towards a state from which it can stay in the set Goal forever. We will express formally this objective using the operators Stay and Until. In order to obtain efficient algorithms, we will use computations starting from initial configurations. We thus introduce some additional definitions. We formalize a subset of « reachable » states from the initial condition while respecting the description of « allowed » states. Formally, this set is defined as follows:

$$\begin{aligned}\mathsf{Acc}(\mathcal{C}, \mathsf{Allow}) = \{s \in S \mid \exists \rho = (s_0, a_1, \ldots, a_{n-1}, s_n) \text{ such that } s_0 \in S_0,\\ s_n = s \text{ and } \forall i \leq n, s_i \in \mathsf{Allow}\}\end{aligned}$$

We are now equipped to formalize the sets of states we are interested in:

$$\mathsf{Stabilize}(\mathcal{C}, \mathsf{Allow}, \mathsf{Goal}) = \mathsf{Until}(\mathcal{C}, \mathsf{Acc}(\mathcal{C}, \mathsf{Allow}), \mathsf{Stay}(\mathcal{C}, \mathsf{Goal} \cap \mathsf{Acc}(\mathcal{C}, \mathsf{Allow})))$$

Intuitively, it reads as $\mathsf{Acc}(\mathcal{C}, \mathsf{Allow})\ \mathcal{U}\ \mathsf{Stay}(\mathcal{C}, \mathsf{Goal} \cap \mathsf{Acc}(\mathcal{C}, \mathsf{Allow}))$. Note that the strategy for $\mathsf{Stabilize}$ is memoryless: it is a simple combination of the strategies for Stay (for states in Stay) and Until (for the other states) indicated above.

For the reader familiar with the μ-calculus, we note that Acc, Stay and Until can be characterized by the following formulae:

$$\begin{aligned}\mathsf{Post}(X) &= \{s' \in S \mid \exists s \in X \text{ s.t. } s \to s' \in T\} && (3)\\ \mathsf{Acc}(\mathcal{C}, A) &: \mu X.S_0 \vee (A \wedge \mathsf{Post}(X)) && (4)\\ \mathsf{Stay}(\mathcal{C}, A) &: \nu X.A \wedge \mathsf{CPre}(X) && (5)\\ \mathsf{Until}(\mathcal{C}, A, B) &: \mu X.B \vee (A \wedge \mathsf{CPre}(X)) && (6)\end{aligned}$$

Note the identity $\mathsf{CPre}(X) = \langle \Sigma_C \rangle X \vee [\Sigma_E] X$. In particular, this allows to deduce a μ-calculus formula characterizing the states in $\mathsf{Stabilize}(\mathcal{C}, \mathsf{Allow}, \mathsf{Goal})$.

2.2 Algorithm for stabilization games

In the sequel, we present the (efficient) algorithm we used to solve the controller synthesis problem. This algorithm is used for each "level" in the hierarchical approach in Section 4.

This algorithm is derived from model checking algorithms for the alternation free μ-calculus, notably integrating concepts first published in [7, 12]. In [5], the authors observe that these local algorithms can be used to decide reachability properties in (un)timed games. We extend this observation to the more complex specifications we consider here. This statement holds because the specification $\mathsf{Stabilize}$ can be expressed as a propsitional μ-calculus formula without alternation. This is a local algorithm, in the sense that the exploration of the transition system is started from initial states, and that the exploration of losing states is stopped.

The algorithm we consider is presented as Algorithm 1. Given a CGS $\mathcal{C}$, and two sets of states Goal, Allow such that $\mathsf{Goal} \subseteq \mathsf{Allow}$, this algorithm returns a pair of sets of states, whose first component is $\mathsf{Stabilize}(\mathcal{C}, \mathsf{Allow}, \mathsf{Goal})$, and second component is $\mathsf{Stay}(\mathcal{C}, \mathsf{Goal} \cap \mathsf{Acc}(\mathcal{C}, \mathsf{Allow}))$.

The algorithm consists of three phases:

1. the first while loop consists in a forward exploration of the reachability graph, restricted to states in Allow. At the end of this phase, the list $Passed$ exactly contains the set $\mathsf{Acc}(\mathcal{C}, \mathsf{Allow})$. In addition, the list NIG (standing for "Not In Goal") contains all the states outside Goal that can be reached from states in $Passed$. Finally, the dependance lists (elements $depend[s]$) contain, for each state $s \in Passed \cup NIG$, the set of predecessor states in $Passed$.
2. the second while loop aims at computing the set $\mathsf{Stay}(\mathcal{C}, \mathsf{Goal} \cap \mathsf{Acc}(\mathcal{C}, \mathsf{Allow}))$, via variable $STAY$. Therefore, it proceeds in a backward propagation of states of NIG, by exploring the reachability graph (restricted to states in Allow) in a backward manner, using the dependency lists. States are added to the $Waiting$ list iff they are declared as losing. For environment states, this happens as soon as a successor is losing, while for controller states, it occurs when no successor is winning (a counter is used to check this).
3. the third while loop computes the set $\mathsf{Stabilize}(\mathcal{C}, \mathsf{Allow}, \mathsf{Goal})$, via variable $UNTIL$. As in the previous case, it proceeds in a backward propagation. However, the computation is dual: while for the computation of Stay (greatest fixpoint), the propagation concerns losing states, for the computation of Until (least fixpoint), it concerns winning states.

Note that the two first while loops can be merged, and thus performed simultaneously. In our implementation (see Section 5), we have given a higher priority to the backward propagation, thus avoiding the exploration of some losing states. The third loop must be performed once the second one has finished, as each edge will be explored only once, the status of the target state must be known when it is explored.

3 Nonlinear systems and discretizations

3.1 Nonlinear systems with inputs

We consider (possibly nonlinear) systems of ordinary differential equations:

Definition 2 (ODE system). *A* system of ordinary differential equations (ODE system for short) *is given by a triple $\mathcal{O} = (f, \mathcal{S}, U)$ where U is a finite set of input parameter*

Algorithm 1: Local Algorithm Local-Stabilize for a Stabilization Objective

```
Data: $\mathcal{C} = \langle S_E, S_C, T, S_0, \lambda \rangle$,Allow,Goal
Result: $UNTIL = \mathsf{Stabilize}(\mathcal{C}, \mathsf{Allow}, \mathsf{Goal})$; $STAY = \mathsf{Stay}(\mathcal{C}, \mathsf{Goal} \cap \mathsf{Acc}(\mathcal{C}, \mathsf{Allow}))$
$Waiting \leftarrow S_0$; $NIG \leftarrow \emptyset$; $Passed \leftarrow \emptyset$;
while $Waiting \neq \emptyset$ do
    $s \leftarrow pop(Waiting)$; $Passed \leftarrow Passed \cup \{s\}$;
    foreach $(s, a, s') \in T$ do
        if $s' \notin \mathsf{Goal}$ then $NIG \leftarrow NIG \cup \{s'\}$ ;
        $depend[s'] \leftarrow depend[s'] \cup \{s\}$;
        if $s' \in \mathsf{Allow} \wedge s' \notin Passed$ then $Waiting \leftarrow Waiting \cup \{s'\}$;
    end
end
$\forall s \in Passed \cap S_C$; $counter_C(s) \leftarrow \#Succ(s)$;
$Waiting \leftarrow NIG$; $STAY \leftarrow Passed$;
while $Waiting \neq \emptyset$ do
    $s \leftarrow pop(Waiting)$;
    $STAY \leftarrow STAY \setminus \{s\}$;
    if $s \in S_C$ then
        foreach $s' \in depend[s]$ do
            if $s' \in STAY$ then $Waiting \leftarrow Waiting \cup \{s'\}$;
        end
    else
        foreach $s' \in depend[s]$ do
            $counter_C(s') \leftarrow counter_C(s') - 1$;
            if $counter_C(s') = 0$ then $Waiting \leftarrow Waiting \cup \{s'\}$;
        end
    end
end
$\forall s \in Passed \cap S_E, counter_E(s) \leftarrow \#Succ(s)$ ;
$Waiting \leftarrow STAY$; $UNTIL \leftarrow \emptyset$ ;
while $Waiting \neq \emptyset$ do
    $s \leftarrow pop(Waiting)$;
    $UNTIL \leftarrow UNTIL \cup \{s\}$;
    if $s \in S_C$ then
        foreach $s' \in depend[s]$ do
            $counter_E(s') \leftarrow counter_E(s') - 1$;
            if $counter_E(s') = 0$ then $Waiting \leftarrow Waiting \cup \{s'\}$;
        end
    else
        foreach $s' \in depend[s]$ do
            if $s' \notin UNTIL$ then $Waiting \leftarrow Waiting \cup \{s'\}$;
        end
    end
end
```

values, $\mathcal{S} \subseteq \mathbb{R}^n$ denotes the state space of the system, and $f : \mathcal{S} \times U \to \mathbb{R}^n$ defines a parameterized system of differential equations [2]*:*

$$\dot{x} = f(x, u), \qquad \textit{with } x : \mathbb{R} \to \mathcal{S}, u \in U \tag{7}$$

A configuration of $\mathcal{O}$ is a pair $c = (x, u) \in \mathcal{S} \times U$. An initial value problem (IVP for short) *is a pair $\mathcal{E} = (\mathcal{O}, c_0)$ composed of an ODE system $\mathcal{O}$ and a (initial) configuration $c_0 = (x_0, u_0)$ of $\mathcal{O}$.*

In the sequel, to ensure the existence of a unique solution to the IVP, we assume that for any $u \in U$, the function $f(\cdot, u)$ is locally Lipschitz.

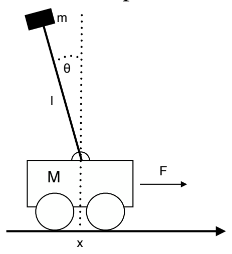
Definition 3 (Trajectory). *Let $\mathcal{O} = (f, \mathcal{S}, U)$ be an ODE system. Given an initial configuration $c_0 = (x_0, u_0)$ of $\mathcal{O}$, a* trajectory *of an ODE system starting from c_0 is a triple $(\mathcal{I}, \sigma, \mathcal{X})$ where:*

- *$\mathcal{I} = \{I_k \mid 0 \leq k \leq N\}$ is a sequence of intervals such that:*
  - *if $N = +\infty$, then for all $k \in \mathbb{N}$, $I_k = [t_k, t'_k]$ with $t'_k = t_{k+1}$,*
  - *if $N < +\infty$, then $I_N = [t_N, t'_N]$ or $I_N = [t_N, +\infty)$ and for all $0 \leq k \leq N-1$, $I_k = [t_k, t'_k]$ with $t'_k = t_{k+1}$,*
  - *in both cases, the initial time is $t_0 = 0$,*
- *$\sigma = \{\sigma_k \mid 0 \leq k \leq N\}$ is a sequence of elements of U such that $\sigma_0 = u_0$, and*
- *$\mathcal{X} = \{x_k \mid 0 \leq k \leq N\}$ is a sequence of continuous, piecewise differentiable functions. For all $0 \leq k \leq N, x_k : I_k \to \mathbb{R}^n$ is the solution of the IVP $(\mathcal{O}, c_k)$, where, for $k \geq 1$, $c_k = (x_{k-1}(t_k), \sigma_k)$.*

We say that a trajectory is finite (resp. infinite) if $N < +\infty$ (resp. $N = +\infty$).

Intuitively, the controller acts on the value of the input parameter u. It has to decide when to change this value (this defines the intervals $\mathcal{I}$) and which value has the input (this defines the sequence σ). Controller synthesis can thus be understood as the synthesis of a mapping which, given the history of the system (a finite trajectory), gives the timestamp of the next input change and the new value of the parameter.

Example 1 (An inverted pendulum). A cart of mass M carries an inverted pendulum of length l with a mass m at the end. The cart can be accelerated somehow by a horizontal force F. This classical control problem can be characterized by a system of four ODEs including as variables θ, the angle of the pendulum relative to the vertical axis, and x, its horizontal position relative to some origin :



$$\begin{cases} \dot{x_1} = x_2 \\ \dot{x_2} = \frac{F + (l.x_4^2 - g.\cos x_3).m.\sin x_3}{M + m.\sin^2 x_3} \\ \dot{x_3} = x_4 \\ \dot{x_4} = \frac{g.\sin x_3.M - \cos x_3.F + (g - l.x_4^2.\cos x_3).m.\sin x_3}{l.M + l.m.\sin^2 x_3} \end{cases}$$

where $x_1 = x$ and $x_3 = \theta$

[2] As usual, $\dot{x}$ denotes the first derivative of x.

3.2 Discretizations

As nonlinear differential equations cannot be solved in general, we will approximate the system by a finite state system, which can be analyzed. Therefore, we first restrict the behaviour of the controller by considering a discrete-time controller, obtained by a sampling rate $\eta \in \mathbb{Q}_{>0}$. This means that discrete changes on the value of the input parameter u can only occur at timestamps in $\eta.\mathbb{N}$.

Then, it remains to approximate the infinite-state dynamics of the ODE system by a finite-state one. In the sequel, we fix an ODE system $\mathcal{O}$, together with an initial configuration $c_0 = (x_0, u_0)$ of $\mathcal{O}$ and let $\mathcal{E} = (\mathcal{O}, c_0)$ be the resulting IVP. We assume that the state-space $\mathcal{S}$ of the system is given by a hyper-rectangle $I_1 \times \cdots \times I_n$ of $\mathbb{R}^n$. This assumption is not restrictive for standard ODE systems. Then, we consider a mesh of the state-space obtained by the product of partitionings of each interval I_j, with $1 \leq j \leq n$. More precisely, we consider, for each $1 \leq j \leq n$, a partitioning $\mathcal{P}_j$ of I_j. This yields a finite state abstraction of the infinite state space of the system. Following definitions introduced in Section 2, we aim at obtaining a control game structure $\mathcal{C}(\mathcal{E}) = \langle S_E, S_C, T, S_0, \lambda \rangle$ over some alphabets $(\Sigma_E, \Sigma_C, \Gamma)$.

In this definition, the controller chooses the value of the input parameter, by choosing a letter in Σ_C. On the other side, the environment resolves the non-determinism associated with the ODE system. In particular, we do not need to label the transitions of the enviroment. This yields the following definitions:

$$\begin{array}{ll} S_C = \Pi_{j=1}^n \mathcal{P}_j & \Sigma_C = U \\ S_E = S_C \times U & \Sigma_E = \{e\}, \text{ for some letter } e \\ S_0 = \{(r_0, u_0)\} & \text{where } r_0 \text{ is such that } x_0 \in r_0 \end{array}$$

Transitions of the controller are the following ones:

$$T \cap S_C \times \Sigma_C \times S_E = \{ s \xrightarrow{u} (s, u) \mid u \in U, s \in S_C \}$$

Regarding transitions of the environment, we want to approximate, given a cell of the mesh (*i.e.* a partition of $\mathcal{S}$), and a value of the input, the reachable cells after a delay of η time units. Note that the assumption that for each value of u, the function $f(\cdot, u)$ is locally Lipschitz ensures the existence and the unicity of a solution to the IVP associated with the ODE system. However, as we consider here as possible initial values any value of a given cell (and a single value of u), there are infinitely many such problems. As a consequence, different cells can be reached from a single one. The problem of the computation of these successors has already been studied by several authors: interval numerical methods [10], standard mathematical techniques based on the evaluation of the Lipschitz constant [3], simulation of the system based on sensitivity analysis [8]... We do not detail here how such approximations can be obtained, as this is orthogonal to the purpose of this paper. However, to obtain a sound method, the transitions of the finite-state system should *over-approximate* the transitions of the ODE system:

Definition 4 (Sound over-approximation). *Let $\mathcal{E} = (\mathcal{O}, c_0)$ be an IVP. The* CGS *$\mathcal{C} = \langle S_E, S_C, T, S_0, \lambda \rangle$ is a* sound over-approximation *of $\mathcal{E}$ if it satisfies the following property:*
$\forall (s, u) \in S_E, \forall x_0 \in s$, let $x(t)$ be the unique solution to the IVP (f, x_0, u). Then for any $s' \in S_C$ such that $x(\eta) \in s'$, we have $(s, u) \xrightarrow{e} s' \in T$.

Finally, we define the labelling function λ. Given a set of atomic propositions Γ, interpreted as subsets of $\mathcal{S}$ by a given mapping χ, we define λ as follows:

$$\forall \gamma \in \Gamma, \forall s \in S_C, \gamma \in \lambda(s) \iff s \cap \chi(\gamma) \neq \emptyset$$

We extend this mapping over S_E by letting $\lambda(s, u) = \lambda(s)$ for any $u \in U$. This is coherent as propositions in Γ are intended to express properties over states but not over parameter values. The above definitions ensure that the discretized CGS built from the partitionning of the state space simulates the behaviour of the ODE system:

Proposition 1 (Simulation). *Let $\mathcal{O} = (f, \mathcal{S}, U)$ be an ODE system, c_0 be a configuration of $\mathcal{O}$, and $\mathcal{C}$ be a CGS that is a sound over-approximation of $\mathcal{E} = (\mathcal{O}, c_0)$. Then, for any trajectory $(\mathcal{I}, \sigma, \mathcal{X})$ of $\mathcal{O}$ such that any interval $I \in \mathcal{I}$ is of the form $[k\eta, (k+1)\eta]$ for some $k \in \mathbb{N}$, there exists a path $\rho = (s_0, a_1, s_1, a_2, \ldots)$ in $\mathcal{C}$ such that $\sigma = (a_i)_i$, and for each i, we have $x_i(t_i) \in s_i$.*

Assume that the partitionings $\mathcal{P}_j$ are compatible with the properties labelings χ, in the sense that for any $s \in S_C$ and any $\gamma \in \Gamma$, we have $s \cap \chi(\gamma) \neq \emptyset$ if, and only if, $s \subseteq \chi(\gamma)$. Then the property of simulation entails that if we can synthesize a controller for the CGS $\mathcal{C}$ w.r.t. some control objective, then this controller can be used as a discrete-time controller for the ODE system $\mathcal{O}$ w.r.t. the same control objective. The only difference is that the atomic properties are ensured only at sampled timestamps.

4 Hierarchical approach to controller synthesis

In this section, we present an original approach for the analysis of the discretizations of ODE systems. In principle, the discretization explodes with the number of variables. Our technique exploits dependencies between variables of the system to first solve smaller subsystems and then use the analysis results to dramatically reduce the size of the state space to explore with the full set of variables.

4.1 Abstractions preserving controllability

We present a particular abstraction used in the sequel, which is a bisimulation w.r.t. possible transitions, but only a simulation w.r.t. properties satisfaction. Within this setting, $\mathcal{C}_2$ can be seen as an abstraction of the system $\mathcal{C}_1$.

Definition 5. *Consider two CGS $\mathcal{C}_i = \langle S_E^i, S_C^i, T^i, S_0^i, \lambda^i \rangle$, and let $S^i = S_E^i \uplus S_C^i$ for $i = 1, 2$. We consider a surjective mapping $\alpha : S^1 \to S^2$, and the associated relation $R \subseteq S^1 \times S^2$ defined by $s_1 \; R \; s_2$ iff $\alpha(s_1) = s_2$.*

We say that α yields a property asymmetric bisimulation relation *R if, and only if, for any pair $s_1 \; R \; s_2$:*

1. *either $s_1 \in S_C^1$ and $s_2 \in S_C^2$, or $s_1 \in S_E^1$ and $s_2 \in S_E^2$,*
2. *if $s_1 \in S_0^1$, then also $s_2 \in S_0^2$,*
3. *for any $\gamma \in \Gamma$, if $\gamma \in \lambda_1(s_1)$, then also $\gamma \in \lambda_2(s_2)$,*
4. *for $(s_1, a, s_1') \in T^1$ there exists s_2' such that $(s_2, a, s_2') \in T^2$ and $s_1' \; R \; s_2'$, and*
5. *for $(s_2, a, s_2') \in T^2$ there exists s_1' such that $(s_1, a, s_1') \in T^1$ and $s_1' \; R \; s_2'$.*

The following proposition states that winning states of the abstract system cover the winning states of the concrete one. This property can be seen as a particular instance of the properties of zig-zags bisimulations, see e.g. [4].

Proposition 2. *Let $\mathcal{C}_i$, $i = 1, 2$ be two* CGS*, and $\alpha : S^1 \to S^2$ be a mapping yielding a property asymmetric bisimulation relation. Let*[3] $\gamma, \gamma' \in \Gamma$*. Then we have:*

$$\mathsf{Stay}(\mathcal{C}_1, \gamma) \subseteq \alpha^{-1}(\mathsf{Stay}(\mathcal{C}_2, \gamma)) \textit{ and } \mathsf{Reach}(\mathcal{C}_1, \gamma, \gamma') \subseteq \alpha^{-1}(\mathsf{Reach}(\mathcal{C}_2, \gamma, \gamma'))$$

Proof (Sketch). We first prove the following property:

$$\forall X_1 \subseteq S_1, X_2 \subseteq S_2, \alpha(X_1) \subseteq X_2 \Rightarrow \begin{cases} \alpha(\mathsf{Post}_1(X_1)) \subseteq \mathsf{Post}_2(X_2) \\ \alpha(\mathsf{CPre}_1(X_1)) \subseteq \mathsf{CPre}_2(X_2) \end{cases}$$

where Post_i (resp. CPre_i) denotes the operator Post (resp. CPre) in the CGS $\mathcal{C}_i$. These properties easily follow from points 1., 4. and 5. of Definition 5. As a consequence, this entails $\alpha(\mathsf{Acc}(\mathcal{C}_1, \gamma)) \subseteq \mathsf{Acc}(\mathcal{C}_2, \gamma)$. Indeed, the property holds for initial states (point 2. of Definition 5). Second, consider the characterization of sets $\mathsf{Stay}(\mathcal{C}_i, \gamma)$ and $\mathsf{Reach}(\mathcal{C}_i, \gamma, \gamma')$ by fixpoints presented in Section 2. We will prove the result by induction on the number of iterations of the fixpoint computation. Initially, the property holds for atomic properties by point 3. of Definition 5, and for the set of reachable states by the above result. The induction follows from the above property of CPre_i. □

4.2 Hierarchical abstractions in ODE systems

Formally, we consider an ODE system $\mathcal{O} = (f, \mathcal{S}, U)$ over real variables $x_1, \ldots, x_n$, to be as follows:

$$\begin{cases} \dot{x_1} = f_1(x_1, \ldots, x_n, u) \\ \vdots \\ \dot{x_n} = f_n(x_1, \ldots, x_n, u) \end{cases}$$

where for each $1 \leq i \leq n$, $f_i : \mathcal{S} \times U \to \mathbb{R}$ is supposed to be locally Lispchitz (notations are taken from Section 3).

Definition 6 (Dependency). *Let $i, j \in \{1, \ldots, n\}$. We say that mapping f_i does not depend on variable x_j iff for any $y, y' \in \mathbb{R}^n$ such that $y_k = y'_k$ for all $k \neq j$, we have $f_i(y) = f_i(y')$. Otherwise, we say that f_i depends on x_j.*

In particular, for standard ODE systems in which mappings f_i's are given by explicit expressions involving polynomials, sine, cosine, ..., the mapping does not depend on a variable as soon as it does not appear in this expression. For instance, regarding the inverted pendulum example, one can note that mappings f_3 and f_4 only depend on variables x_3 and x_4.

Definition 7 (Independent subset of variables). *Let $J \subset \{1, \ldots, n\}$. We say that the subset of variables J is* independent *if the subsystem obtained by the restriction to variables $\{x_j \mid j \in J\}$ constitutes an independent subsystem,* i.e. *iff for any $j \in J$, mapping f_j only depends on variables in the set $\{x_j \mid j \in J\}$.*

[3] For readability, we shortcut $\lambda_i \gamma$ by simply γ in the expression $\mathsf{Stay}(\mathcal{C}_i, \lambda_i(\gamma))$ and similarly for Reach.

For the example of the inverted pendulum, there are four independent subsets of variables : $\emptyset$, $\{x_3, x_4\}$, $\{x_2, x_3, x_4\}$ and $\{x_1, x_2, x_3, x_4\}$.

Proposition 3. *The independent subsets of variables of an ODE system is a complete lattice.*

Definition 8. *Let $\mathcal{O}$ be an ODE system. We denote by $\mathcal{L}(\mathcal{O})$ the complete lattice of its independent subsets of variables. In addition, given $J, J' \in \mathcal{L}(\mathcal{O})$, we write $J' \prec J$ iff $J' \subsetneq J$, and there does not exist a set $J'' \in \mathcal{L}(\mathcal{O})$ such that $J' \subsetneq J''$ and $J'' \subsetneq J$.*

Definition 9. *Consider an IVP $\mathcal{E} = (\mathcal{O}, c_0)$, and a set of partitionings $\mathcal{P}_j$, for $1 \leq j \leq n$. For any set $J \in \mathcal{L}(\mathcal{O})$, we denote by $\mathcal{C}_J(\mathcal{E})$ the discretization of the subsystem of $\mathcal{O}$ restricted to J, w.r.t. partitionings $\mathcal{P}_j$, with $j \in J$.*
Let $J, J' \in \mathcal{L}(\mathcal{O})$ such that $J' \subseteq J$. We denote by $\pi_{J,J'}$ the projection from states of $\mathcal{C}_J(\mathcal{E})$ to states of $\mathcal{C}_{J'}(\mathcal{E})$ obtained by erasing components of J not in J'. We simply write π_J to denote the projection $\pi_{\{1,\ldots,n\},J}$.

The following Lemma states that independent subsets of variables can be used for hierarchical computations:

Lemma 1. *Let $J, J' \in \mathcal{L}(\mathcal{O})$ such that $J' \subseteq J$. The mapping $\pi_{J,J'}$ yields an asymmetric property bisimulation relation between $\mathcal{C}_J(\mathcal{E})$ and $\mathcal{C}_{J'}(\mathcal{E})$.*

Proof (Sketch). Let R denote the relation associated with $\pi_{J,J'}$. We have to prove that R satisfies point 1. to 5. of Definition 5. Points 1. to 4. easily follow by definition of a projection mapping, and would be true for any sets J, J' such that $J' \subseteq J$. Point 4 holds because J and J' are independent subset of variables. This implies that any trajectory $(\mathcal{I}', \sigma', \mathcal{X}')$ in the ODE system $\mathcal{O}$ restricted to J' can be extended into a trajectory $(\mathcal{I}, \sigma, \mathcal{X})$ in $\mathcal{O}$ restricted to J whose projection on J' coincides with $(\mathcal{I}', \sigma', \mathcal{X}')$. $\square$

Algorithm 2: Hierarchical Algorithm for the Synthesis w.r.t. Stabilization Objectives

```
Data: $\mathcal{E} = (\mathcal{O}, c_0), \mathsf{Allow}, \mathsf{Goal}$
Result: $\mathsf{Stabilize}(\mathcal{C}(\mathcal{E}), \mathsf{Allow}, \mathsf{Goal})$
Compute the lattice $\mathcal{L}(\mathcal{O})$ ;
foreach $J \in \mathcal{L}(\mathcal{O})$, ordered by increasing size do
    $A \leftarrow \pi_J(\mathsf{Allow}) \cap \bigcap_{J' \prec J} \pi^{-1}_{J,J'}(U(J'))$;
    $G \leftarrow \pi_J(\mathsf{Goal}) \cap \bigcap_{J' \prec J} \pi^{-1}_{J,J'}(S(J'))$;
    $(U(J), S(J)) \leftarrow \mathsf{Local\text{-}Stabilize}(\mathcal{C}_J(\mathcal{E}), A, G)$ ;
end
Return $U(\{1, \ldots, n\})$;
```

This allows us to derive Algorithm 2, which first solves the control problem for smaller sets of variables, and uses the results to limit the domain explored by further resolutions of the control problem: compute incrementally for all independent subsets bottom up the set of winning states for Stay and Until objectives, and exploit the asymmetric bisimulation and property inheritance (Proposition 2) to eliminate states from these two sets if they are not in Stay or Until in the projection.

This approach allows, based on the analysis on subsets of variables, to reduce the exploration space by observing what happens in the projection. To understand this intuitively, let us consider the independent subsets of the inverted pendulum. If the control objective is to go to a certain position and keep the pendulum close to the vertical position, three different problems have to be solved : first we solve the problem only for angular speed and position which means to solve the problem of balancing the pendulum independently of the vehicle movement. If, afterwards, the problem is extended to include vehicle speed and then vehicle position, states for which it is not possible to balance the pendulum are immediately removed from the sets of candidates with additional objectives for the position.

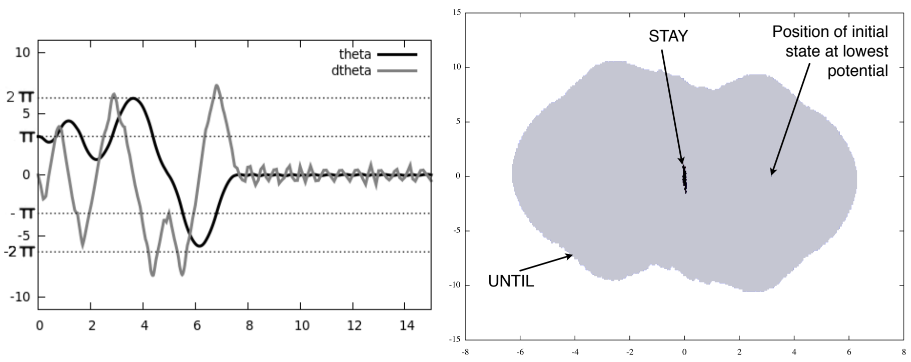
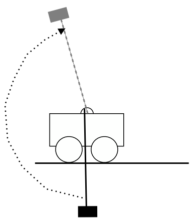


Fig. 1. A strategy simulation for a swing up and a representation of winning states

5 Experiments

We have realized a prototype implementation of the algorithms described in this paper and demonstrate its capacities using the example of the inverted pendulum.

As an illustration of what can be achieved with specifications, we consider the "swing up" problem: we suppose that the pendulum is initially hanging at its lowest potential energy (see image on the right) and we ask for a controller which lifts it to the vertical upright position. Figure 1 at left shows a simulated trajectory obtained from a synthetized controller which illustrates how the angle θ of the pendulum is raised from the lowest position (radiant angle π) with several swings before stabilizing with tiny oscillations at the vertical position (radiant 0). The image on the right shows the winning sets Stay (black) and Until (gray and black) for the stated swingup problem, a small box indicates the initial configuration.



Concerning the reduction potential of the hierarchical algorithm, we give sample figures for the synthesis of a controller limiting all four variables of the pendulum. We count in particular the number of states explored with and without the hierarchical approach. The third line (exploration ratio, $\frac{explored}{|S_C|} * 100$) gives an impression of the advantage of the combined local and hierarchical approach over a global approach: for the biggest example, only 11% of the states are visited. The difference of the local approach with and without hierarchy is explored in the fourth and fifth line: using results from $\{x_3, x_4\}$ in the computation for $\{x_2, x_3, x_4\}$ allows a state space reduction of 48%. Data for $\{x_1, x_2, x_3, x_4\}$ without the hierarchical approach is not available since it was beyond our current implementation and available hardware.

	$\{x_3, x_4\}$	$\{x_2, x_3, x_4\}$	$\{x_1, x_2, x_3, x_4\}$
$\lvert S_E \cap \mathsf{Allow}\rvert$	1.049.600	136.448.000	6.822.400.000
$\lvert S_C \cap \mathsf{Allow}\rvert$	25.600	3.328.000	166.400.000
explored part of $S_C \cap \mathsf{Allow}$	17.616	815.643	18.261.684
exploration ratio	68%	24%	11%
explored without hierarchy	17.616	1.571.127	n/a
savings by hierachical approach	0%	48%	n/a
$\lvert S_C \cap \mathsf{Stay}\rvert$	1.683	50.787	1.305.059
$\lvert S_C \cap \mathsf{Until}\rvert$	10.121	432.547	9.678.467

Optimizations in the prototype. We discuss some of the optimizations we introduced in the prototype to make it work with case studies of the size of the pendulum.

The first two while loops of 1 can actually be merged and combined with a computation of $\mathsf{Stay}(\mathcal{C}, \mathsf{Allow} \cap \mathsf{Acc}(\mathcal{C}, \mathsf{Allow}))$. This combination has the advantage of avoiding the exploration of certain states and making the algorithm a bit more local.

It turned out that an explicit representation of $depend[s]$ by lists creates a major bottleneck in terms of memory usage: looking at the figures for S_E in the table, one can understand why, there are billions of transitions involved over which backward propagation may take place. The experiments shown here therefore add a symbolic overapproximation technique that allows to safely track supersets of the actual predecessors. Using these supersets in backward propagation is like adding non-determinism to the environment, thus, if a controller with this overapproximation exists, so does one for the case without.

In the future, we want to look into the possibility of optimizing memory usage of the algorithms by exploiting more knowledge about the structure of the state space.

6 Conclusions and future work

We have developped an approach for the synthesis of stabilizing controllers of hybrid systems that exploits the structure of differential equations for state reduction.

The combinatory explosion due to the non-deterministic overapproximation introduced by discretization is the big challenge and many techniques must be combined to make such approaches realistic. In our experiments, we found that the slicing approach helps in finding good abstractions for the independent variable subsets before going to more complex levels. This is orthogonal and complementary to compositional approaches such as [14] which are needed if one wants to synthesize controllers for complex systems: in hybrid systems there are limits to decomposition and this is where our slicing type approach can help. Another promising direction is the use of counterexample guided refinements for controller synthesis [11].

The reduction approach uses a notion of bisimulation for its correctness and the algorithms are fundamentally based on a certain fragment of the μ-calculus [4] : formulae without negation on the properties. The hierarchical reduction framework is thus open for extension to a much larger class of properties which can be expressed in the alternation free fragment of the μ-calculus, and it is not difficult to extend the proofs in Section 4 to a more general case.

In the future, we want to extend our work to the case of control objectives changing with time while preserving stabilization in the sense we use it here.

References

1. R. Alur, T. Dang, and F. Ivancic. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006.
2. R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, (88):971–984, 2000.
3. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007.
4. J. Brandfield and C. Stirling. *The Handbook of Modal Logic*, chapter Modal mu-calculi, pages 721–756. Elsevier, 2006.
5. F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *Proc. CONCUR'2005*, volume 3653 of *LNCS*, pages 66–80. Springer, 2005.
6. E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.
7. R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In *FMSD*, pages 48–58. Springer-Verlag, 1993.
8. A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. In *Proc. HSCC'07*, volume 4416 of *LNCS*, pages 174–189. Springer, 2007.
9. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
10. T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond hytech: Hybrid systems analysis using interval numerical methods. In *Proc. HSCC'00*, volume 1790 of *LNCS*, pages 130–144. Springer, 2000.
11. T. A. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *Proc. ICALP'03*, volume 2719 of *LNCS*, pages 886–902. Springer, 2003.
12. X. Liu and S. A. Smolka. Simple linear-time algorithms for minimal fixed points. In *Proc. ICALP'98*, volume 1443 of *LNCS*, pages 53–66. Springer, 1998.
13. T. Moor, J. M. Davoren, and J. Raisch. Learning by doing — systematic abstraction refinement for hybrid control synthesis. In *IEE Proc. Control Theory & Applications, Special issue on hybrid systems*, volume 153, 2006.
14. S. Perk, T. Moor, and K. Schmidt. Controller synthesis for an i/o-based hierarchical system architecture. In *International Workshop on Discrete Event Systems (WODES)*, IEEE, pages 474–479, 2008.
15. A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In *Proc. HSCC'02*, volume 2289 of *LNCS*, pages 465–478. Springer, 2002.
16. C. Tomlin, J. Lygeros, and S. Sastry. Computing controllers for nonlinear hybrid systems. In *Proc. HSCC'99*, volume 1569 of *LNCS*, pages 238–255. Springer, 1999.