Synthesizing Discrete Controllers from Hybrid Automata — Preliminary Report*

Marcus Bjäreland and **Dimiter Driankov**
Dept of Comp and Info Science
Linköping University
S-581 83 Linköping, Sweden
{marbj, dimdr}@ida.liu.se

Abstract

The task of synthesizing and verifying a specification of a controller for a hybrid system is different from the task of synthesizing an executable controller. Specifications of controllers are very useful for the verification of the closed-loop system, but the verification procedure often requires that the system behaves in an ideal way. In, e.g., automata approaches, it is typically required that the initial state is known, and that the actual system under control behaves according to its model. In industrial applications such requirements are unrealistic; an industrial controller can never make assumptions about the initial state, and it must be able to handle de-synchronizations, i.e. deviations from the expected effects of control actions.

In this paper we present preliminary results on how to synthesize robust discrete controllers from a class of Rectangular Hybrid Automata. The language used for the synthesized controllers is Nils Nilsson's Teleo-Reactive Trees. We prove that the synthesized controllers handle at least the situations for which the original automata is verified.

Introduction

Formalisms for modeling Hybrid Systems are mainly designed for verification purposes. The problem of controller synthesis from such formalisms has been widely studied, but the aim of that research has emphasized synthesizing *specifications* of controllers, rather than on synthesizing the actual controller programs (see e.g. (Zhang and Mackworth 1995; Lennartsson *et al.* 1996; Henzinger and Kopke 1997)). A specification of a controller is useful for verifying correctness of the closed loop system, but it is quite different from an executable controller which has to exhibit properties that may not be of importance for verification. For example, an executable controller has to be fast, i.e. we cannot allow arbitrary computations or logging of system history. In engineering practice this implies that controllers typically are reactive and that they lack memory. Moreover, the controllers must be robust in the sense that there must be some control action invoked for whatever the input may be to the controller. Another requirement is that the controller must be able to handle de-synchronizations, i.e., deviations from expected system states. Furthermore, the controller should work whatever state it is initiated in. This poses a problem for some automata approaches (Alur *et al.* 1992; Ramadge and Wonham 1989), where the initial states are assumed to be known, and the system is assumed to follow the expected course of events. The requirements for controllers described above have been implemented and discussed in e.g. (Pell *et al.* 1996; Williams and Nayak 1996).

In this preliminary work we have chosen to study the problem of synthesizing (homeostatic) *teleo-reactive trees* (Nilsson 1994) from a specification in terms of a *rectangular hybrid automaton* (Alur *et al.* 1992; Henzinger and Kopke 1997). Teleo-reactive (T-R) trees are structures consisting of condition-action pairs combined to form trees. T-R trees have successfully been used for a number of applications (see e.g. (Benson 1996)). The interest in the T-R representation is motivated by the strong similarities between it and languages for *Programmable Logic Controller* (PLC) (Lewis 1997) (especially the Petri-net descendant language SFC) which are used for industrial applications.

Rectangular hybrid automata (RHAs) is a modeling formalisms where the continuous behavior is modeled by differential inequalities (e.g. $0 \leq \dot{x} \leq 1$), and the discrete behavior in terms of (instantaneous) mode switches. We will assume that the RHAs we study are specifications of controllers, and that every mode switch is due to an invoked control action. Moreover, we will restrict ourselves to handle controllers for which

*This research has been supported in parts by the Swedish Research Council for Engineering Sciences (TFR), the Knut and Alice Wallenberg Foundation, and ASEA Brown Bovery (dept ISY/AMC).

the control goal can be formulated as safety requirements (that some condition should, or should not, be maintained during the control).

We will begin with an example of a hybrid automaton representing a specification for the control of a simple water tank, then we will present the RHA and T-R-tree formalisms. After that, we will present the translation algorithm from RHA to T-R trees and prove that whenever a RHA specified closed-loop system is behaving as expected, the controller will be able to maintain the particular control goal. We will finish with a discussion on robustness and execution monitoring of the synthesized controllers.

Water Tank Example

This is example is similar to an example in (Ho 1995).

Imagine a water tank of 12 inches height, where the inflow is regulated via a valve, and the outflow is constant but uncontrollable. If the valve is open the water level increases with 1 inch per second, and if it is closed it decreases with 2 inches per second. We assume that there is a time delay of 2 seconds from when a controller sends a `CLOSE_VALVE` or `OPEN_VALVE` signal to the valve, to when the valve actually closes or opens. The control goal is to maintain the water level between 1 and 12 inches.

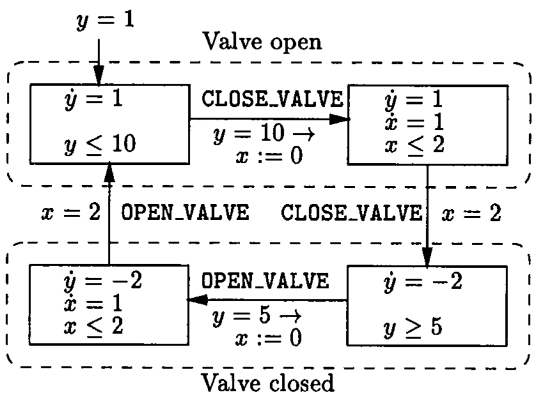

Figure 1: A rectangular hybrid automata modelling the water tank example. The variable y denotes the water level, and x denotes a clock.

Figure 1 depicts a rectangular hybrid automaton (Alur *et al.* 1992; Henzinger and Kopke 1997) that specifies how the plant (water tank) should be controlled. By running the HyTech model checker (Henzinger *et al.* 1997) on the automaton, we verified that the system does maintain the control goal. This is of course desirable and nice, but it does not solve a number of problems involved in the design of a robust control system for the particular application.

Preliminaries

In this section we will present the formal definitions of rectangular hybrid automata and T-R trees. The definitions are taken from (Henzinger and Kopke 1997).

Rectangular Hybrid Automata

Definition 1 Let $X = \{x_1, \ldots, x_n\}$ be a set of real-valued variables. A *rectangular inequality* over X is an expression of the form $x_i \sim c$, where c is an integer constant, and $\sim \in \{<, \leq, \geq, >\}$. A *rectangular predicate* over X is a conjunction of rectangular inequalities. We denote the set of all rectangular predicates over X with $Rect(X)$. The set of vectors $\vec{z} \in \mathbb{R}^n$ that satisfies a rectangular predicate is called a *rectangle.* For a particular rectangular predicate ϕ, we denote the corresponding rectangle with $[\![\phi]\!]$. By writing ϕ^i, for a rectangular predicate ϕ, and a variable index i, we denote the conjunction of all rectangular inequalities in ϕ only involving the variable x_i. For a set of indices, I, we define $\phi^I = \bigwedge_{i \in I} \phi^i$.$\square$

Definition 2 (Rectangular Automaton)
A *rectangular automaton* A consist of the following components.

Variables. The finite set $X = \{x_1, \ldots, x_n\}$ of real-valued variables representing the continuous part of the system. We write $\dot{X} = \{\dot{x}_i \mid x_i \in X\}$ for the set of dotted variables, representing the first derivatives. For convenience, we write X' to denote the set $\{x'_i \mid x_i \in X\}$ (which we will use to connect variable values before and after mode switches).

Control Graph. The finite directed multigraph $\langle V, E \rangle$ represents the discrete part of the system. The vertices in V are called *control modes* which we also will refer to as *locations.* The edges in E are called *control switches.* The switches will sometimes be viewed as functions, i.e. we can say that $e(v) = v'$ iff $e = \langle v, v' \rangle$. In a graphical representation of an automaton the locations correspond to the boxes and the switches to the arrows between boxes.

Initial Conditions. The function $init : V \rightarrow Rect(X)$ maps each control mode to its *initial condition,* a rectangular predicate. When the automaton control starts in mode v, the variables have initial values inside the rectangle $[\![init(v)]\!]$.

Invariant Conditions. The function $inv : V \rightarrow Rect(X)$ maps each control mode to its *invariant condition,* a rectangular predicate. The automaton control may reside in mode v only as long as the values of the variables stay inside the rectangle $[\![inv(v)]\!]$. We define $inv(A)$ as $inv(A) = \bigwedge_{v \in V} inv(v)$.

Jump Conditions. The function *jump* maps each control switch $e \in E$ to a (non-rectangular) predicate $jump(e)$ of the form $\phi \wedge \phi' \wedge \bigwedge_{i \notin update(e)} x_i = x'_i$, where $\phi \in Rect(X)$, $\phi' \in Rect(X')$, and $update(e) \subset \{1, \ldots, n\}$. The jump condition $jump(e)$ specifies the effect of the change in control mode on the values of the variables: each unprimed variable x_i refer to the corresponding value before the control switch e, and each primed variable x'_i to a corresponding value after the switch. So the automaton may switch across e if

1. the values of the variables are inside $[\![\phi]\!]$, and
2. the value of every variable x_i with $i \notin update(e)$ is in the rectangle $[\![\phi'^{i}]\!]$.

Then, the value of every variable x_i with $i \notin update(e)$ remains unchanged by the switch. The value of every x_i with $i \in update(e)$ is assumed to be updated nondeterministically to an arbitrary value in the rectangle $[\![\phi'^{i}]\!]$. For a jump condition $jump(e) \equiv \phi_e \wedge \phi'_e \wedge \bigwedge_{i \notin update(e)} x_i = x'_i$, we define $jump'(e) \equiv \phi_e$, to denote the actual condition that forces the switch e.

Flow Conditions. the function $flow : V \rightarrow Rect(\dot{X})$ maps each control mode v to a *flow condition*, a rectangular predicate that constrains the behavior of the first derivatives of the variables. While time passes with the automaton control in mode v, the values of the variables are assumed to follow nondeterministically any differentiable trajectory whose first derivative stays inside the rectangle $[\![flow(v)]\!]$.

Events. Given a finite set Λ of *events*, the function $event : E \rightarrow \Lambda$ maps each control switch to an event.

Thus, a rectangular automaton A is a nine-tuple $\langle X, V, E, init, inv, jump, flow, \Lambda, event \rangle$.□

In this paper we are only interested in control to maintain safety requirements, which means that, for a particular automaton, we assume that we have a *goal*, in terms of a rectangular predicate ϕ_g, such that the values assigned (or sensed) to variables always belong to it. we will also assume that the automata we study always have goals for which they are verified.

Definition 3 (State)
Let A be a rectangular automaton. A *state* of A is a pair $\langle v, \vec{z} \rangle$, where $v \in V$ is a control mode and $\vec{z} \in [\![inv(v)]\!]$ is a vector satisfying the invariant condition of v. The set of states for A is denoted Q.□

The state notion makes control quite simple, if we define a controller as a function from states to control actions, in terms of the corresponding jump conditions. For example, a control function (or law) C can be defined as, for every state $\langle v, \vec{z} \rangle$ and event λ,

$$C(v, \vec{z}) = \lambda,$$

iff, there exists a mode switch e such that $e(v)$ is defined, $\lambda = event(e)$, and $\vec{z} \in [\![jump'(e)]\!]$. However, we cannot assume that we can distinguish locations from each other in other ways than by using sensor data. This means that we equip the system with a *state estimation* mechanism.

In this paper we assume that the invariants are mutually exclusive. One way of doing this, which is common in engineering practice, is to equip the actuators in the controlled system with sensors, to be able to track the status of the actuators. In our example, that would mean that we introduce a boolean sensor *open* that is true whenever the valve is open, and false otherwise.

Teleo-Reactive Trees

A teleo-reactive (T-R) tree is, in its simplest form, a set of production rules (see (Nilsson 1994) for the original definitions):

$$\begin{array}{c} K_1 \rightarrow a_1 \\ \vdots \\ K_m \rightarrow a_m \end{array}$$

The K_is are conditions (on sensory inputs) and the a_is are actions (on the world). There exists a function F mapping the pairs on the *next expected condition*, i.e. whenever $F(K, a) = K'$, we expect the condition K' to materialize when the action a has been executed in a situation satisfying condition K. F is assumed to be defined for every production rule, except for the root node, which implies that F defines a tree structure on the production rules.

The *execution cycle* of a T-R tree consists of three steps

1. Input is read,
2. the condition closest to the root, that is satisfied is chosen (nondeterministically if no unique such condition exists), and,
3. the corresponding action is executed.

Translation Algorithm

In general, it is not possible to translate a hybrid automata into a T-R tree. For example, if a mode switch is not due to a control action, the controller would require some kind of memory. So, we need to identify a class of automata for which the translation is possible. Tentatively, we propose the following restrictions:

- we assume that mode switches are due to control actions and that the labels on the switches are control actions or conjunctions of control actions (that are to be executed in parallel).

- We view assignments in jump conditions as control actions. In a controller, there is no technical difference between invoking an action, and assigning a value to an internal variable. This means that the starting and the resetting of clocks are control actions. We introduce this in the automata by adding `START_CLOCK` and `STOP_CLOCK` actions for every clock. The `START_CLOCK` action resets the clock and makes in running (we assume that invoking `START_CLOCK` on an already running clock has no effects), and `STOP_CLOCK` stops the clock. This information can be derived from the automaton by adding `START_CLOCK` as a label on every mode switch where the clock variable is set to some value, and `STOP_CLOCK` on every mode switch $e = \langle v, v' \rangle$ where the clock variable has a continuous behavior in location v (i.e. the clock variable derivative is non-zero) and no continuous behavior in v'.

We will call an automaton that satisfies the restrictions above and that have mutually exclusive invariants a *admissible* automaton. In Figure 2 we see an admissible automaton specifying the closed-loop water tank world.

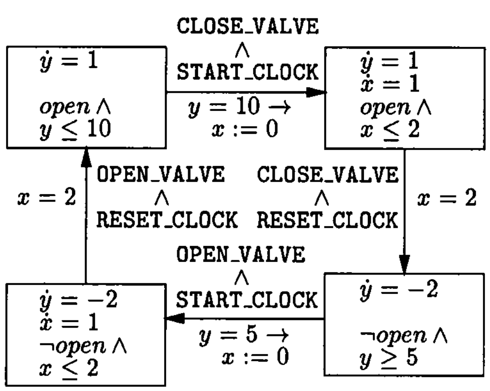

Figure 2: The automaton from Figure 1 from a controller point of view.

Let ϕ_g be the rectangular predicate of the control goal, and let I be the set of indices of the variables in ϕ_g. Construct the set $D = \{v \in V \,|\, inv(v)^I \wedge \phi_g$ is satisfiable$\}$. The set D contains all locations whose invariants intersect with the control goal. The purpose of the control will be to move the system into such locations. We will call those locations *idle* locations. The condition of the root node of the T-R tree should be a rectangle which describes the behavior of the plant inside any idle location, and where no jump conditions are satisfied. Since the jump conditions are included in the invariants of the idle locations, the root node condition is

$$\phi_g \wedge \bigvee_{v \in D} inv(v) \wedge \neg jump'(e), \tag{1}$$

for every e such that $e(v)$ is defined.

Next, we construct the layers of the tree. For every switch $e = \langle v, v' \rangle$ where $v' \in D$ there is an edge into the top node, labeled with the control action $event(e)$. Let I' denote the set of indices for the variables involved in $jump'(e)$. We will construct the condition related to the control action $event(e)$ by negating the invariant $inv(v)^{I'}$, and adding the jump condition and the rest of the invariant. In this way the controller will choose this node if the jump condition and the invariant not concerning the variables involved in the jump condition are satisfied, or if the variables in the jump condition does not satisfy the invariant. In this way we may extend the the number of cases of inputs that the controller can handle, compared to the cases defined by the invariants of the locations in the automaton. Thus, the condition is

$$inv(v)^{\overline{I'}} \wedge (jump'(e) \vee \neg inv(v)^{I'}) \tag{2}$$

where $\overline{I'}$ denotes the set of all indices of variables not in I'.

This procedure is continued for consecutive layers for every condition that does not correspond to an idle location, i.e., if a condition is constructed from an idle location, that condition will be a leaf in the tree.

In Figure 3 we see the T-R tree resulting from the synthesis from the water tank model.

Now, we set out to prove soundness of our synthesize algorithm. We do this by tracking the behaviour of the original automaton, and then showing that, at every time point, the controller behaves as expected.

Proposition 4 Let A be an admissible RHA, ϕ_g the verified control goal of A, and T a T-R tree synthesized from A. We assume that the execution cycles of T can be executed sufficiently fast. Then, at every time point, we have the following:

i If A is in an idle location and no jump condition is satisfied, then the root node of T is satisfied.

ii If A is in a non-idle location and no jump condition is satisfied, then no node in T is satisfied.

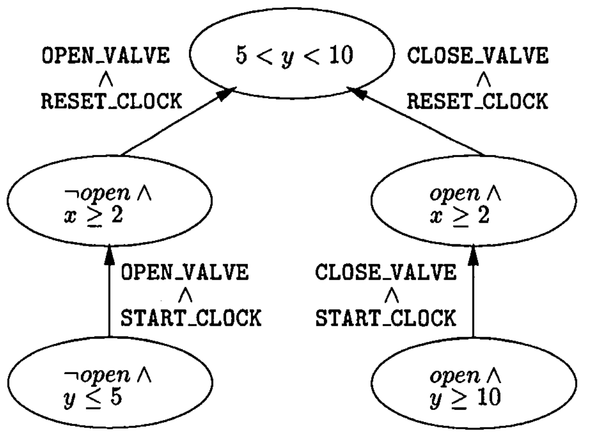

Figure 3: The T-R tree generated from the automaton in Figure 2.

iii If A is in any location and some jump condition is satisfied, then the the node in T corresponding to the location is satisfied and exactly the control action labeling the switch of the jump is invoked.

Proof (sketch): We deal with the three cases separately.

i If A is in an idle location v and no jump condition is satisfied, the variable values must satisfy $inv(v)$ but not any $jump'(e)$ for which $e(v)$ is defined. By the definition of an idle location and Equation (1), the root condition is satisfied. It also clear that Equation (2) is not satisfied.

ii Since we have assumed that the invariants of the locations are mutually exclusive it is clear that the conjunction of Equation (1) is empty, and thus, the root node is not satisfied. Next, since $jump'(e)$ for any switch e from the location, and the negation of the invariant in the location is not satisfied, the intersection in Equation (2) is empty. Thus, no node is satisfied.

iii Obviously, the root node is not satisfied. We have assumed that for some switch e from the current location $jump(e')$ is satisfied. Moreover, the invariant conditions on variables not mentioned in $jump(e')$ must be satisfied. Therefore, the corresponding node with condition described by Equation (2) is satisfied and since we have transferred the labeling from A to T directly, the same control actions are invoked.

□

Proposition 4 basically states that if the real, controlled plant behaves as the verified automaton predicts, the synthesized controller will achieve the control goal.

Discussion and Future Work

The construction of the internal nodes of the tree described above may be able to handle cases which the original RHA was not designed for. For example, in the synthesized T-R tree in Figure 3 we can see that the bottom left condition handles the case when the water level is *less than* or equal to 5, while the case where y is less than 5 is not explicitly mentioned in the RHA (Figure 2). This simple extension that provides more robustness is based on the intuition (from Equation 2) that if we have identified the current location according to the variables not included in the jump condition, and the invariant is not satisfied for that location *or* a particular jump condition is satisfied, the control action corresponding to that jump should be taken. In our tank world example, the original RHA was verified for $y = 1$ starting in the upper right location. If we imagine that the flow rate is increased, e.g. so that the "real" flow condition is $\dot{y} = 2$ when the valve is open and $\dot{y} = -1$ when the valve is closed, the hybrid automata would fail to detect the jump conditions, since the measurements of y will be $1, 3, 5, 7, 9, 11, \ldots$ where the jump condition $y = 10$ never is satisfied. The automata will eventually be de-synchronized. By letting the controller invoke control actions when the jump condition is satisfied or when the current invariant is not satisfied, the situation can be handled (but with the possibility that the control goal is not ratified, temporarily).

A controller synthesized from a RHA gives us a unique opportunity for Execution Monitoring. Since the translation is sound (Proposition 4) we can track the behavior of the controlled system in the RHA and *explain* some de-synchronizations. How this should be done formally and algorithmically is out of the scope of this paper, and belongs in part to future work. In earlier work we have introduced "Ontological Control" (OC) (Driankov and Fodor 1993; Fodor 1998; Bjäreland and Fodor 1998) where the aim is to construct domain-independent execution monitors for industrial control applications. The fundamental problem of OC is: When is a detected discrepancy between expectations and the real measured state due to external disturbances, and when is it due to modeling faults (i.e. faulty expectations)? The approach in OC has been to take a PLC program, generate precondition-action-postcondition triples from the program, and then to monitor the execution according to these structures. There are two problems with this approach: First, it is difficult to generate the precondition-action-postcondition triples from the programs; we have not yet found a completely automatic way of doing this. Secondly, the generated structures contains far less information about the dynamics of the controlled system than the engineer uses when writing the program. The extra information is not vital for solving the fundamental problem of OC, but it is important to an operator when she tries to find the physical reason for the discrepancy. In the future we will study how OC can be used when the controller is constructed from a formal model. However, in this paper we only synthesized the controller.

Conclusions

We have presented an algorithm that synthesizes a controller, in terms of a Teleo-Reactive tree, from a hybrid systems formalism, a class of Rectangular Hybrid Automata. We have shown that the algorithm is sound and argued that the resulting controller is more robust than the automata controller, in the sense that it can handle at least all the situations that the automata is verified for, and sometimes more situations.

References

R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and and verification of hybrid systems. In *Workshop on Theory of Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer Verlag, Lyngby, Denmark, October 1992.

S. Benson. *Learning Action Models for Reactive Autonomous Agents.* Phd thesis, Stanford University, 1996.

M. Bjäreland and G. Fodor. Ontological control. In *Working Papers of the Ninth International Workshop on Principles of Diagnosis*, Sea Crest Resort, N. Falmouth, MA, USA, May 1998.

D. Driankov and G. Fodor. Ontological real-time control. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies*, Aachen, Germany, September 1993.

G. Fodor. *Ontologically Controlled Autonomous Systems: Principles, Operations and Architecture.* Kluwer Academic, 1998.

T. Henzinger and P. Kopke. Discrete-time control for rectangular hybrid automata. In *Proceedings of the Twentyfourth International Colloquium on Automata, Languages, and Programming (ICALP'97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 582–593. Springer-Verlag, 1997.

T. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In *Proceedings of the Ninth International Conference on Computer-Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer-Verlag, 1997.

P. Ho. *Automatic Analysis of Hybrid Systems.* Phd thesis, Cornell University, 1995.

B. Lennartsson, M. Tittus, B. Egardt, and S. Pettersson. Hybrid systems in process control. *IEEE Control Magazine*, October 1996.

R.W. Lewis. *Programming industrial control systems using IEC 1131-3.* Number 50 in IEE Control Engineering Series. The Institution of Electrical Engineers, London, United Kingdom, 1997.

N.J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, (1):139–158, 1994.

B. Pell, E. Gat, R. Keesing, N. Muscettola, and B. Smith. Plan execution for autonomous spacecraft. In *Working Notes of the AAAI Fall Symposium on Plan Execution*, Cambridge, MA, 1996.

P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE: Special Issue on Discrete Event Systems.*, 77:81–98, 1989.

B.C. Williams and P.P. Nayak. A model-based approach to reactive self-configuring systems. In *AAAI'96*, 1996.

Y. Zhang and A. Mackworth. Synthesis of hybrid constraint-based controllers. In *Hybrid Systems II*, number 999 in Lecture Notes in Computer Science. Springer-Verlag, 1995.