USING HYTECH TO SYNTHESIZE CONTROL PARAMETERS FOR A STEAM BOILER

by

Thomas A. Henzinger and Howard Wong-Toi

Memorandum No. UCB/ERL M96/61

15 October 1996

USING HYTECH TO SYNTHESIZE CONTROL PARAMETERS FOR A STEAM BOILER

by

Thomas A. Henzinger and Howard Wong-Toi

Memorandum No. UCB/ERL M96/61

15 October 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Using HyTech to Synthesize Control Parameters for a Steam Boiler*

Thomas A. Henzinger[1] Howard Wong-Toi[2]

[1] Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA
[2] Cadence Berkeley Labs, Berkeley, CA

Abstract. We model a steam-boiler control system using hybrid automata. We provide two abstracted linear models of the nonlinear behavior of the boiler. For each model, we define and verify a controller that maintains the safe operation of the boiler. The less abstract model permits the design of a more efficient controller. We also demonstrate how the tool HyTech can be used to automatically synthesize control-parameter constraints that guarantee the safety of the boiler.

1 Introduction

A description of an industrial steam boiler has been proposed as a benchmark problem for the formal specification and verification of embedded reactive systems [1, 2]. Our approach to the problem is unique in that we use *algorithmic* techniques to analyze *directly*, without discretization, the mixed discrete-continuous components of the system. In this way we are able to fully automatically synthesize safe values for the parameters that control the continuous behavior of the boiler.

We describe the steam boiler and its controller using *hybrid automata* [4, 3]. These automata model nondeterministic continuous activities within a nondeterministic discrete transition structure. Since the nonlinear behavior of the steam-boiler system is not directly amenable to automatic analysis, we provide an approximating model using *linear hybrid automata*. Linear hybrid automata are a subclass of hybrid automata with linearity restrictions on continuous activities (inequalities between linear combinations of first derivatives) and discrete transitions (inequalities between linear combinations of transition sources and targets). Model-checking based analysis techniques [5] for this subclass have been implemented in HyTech [16, 17] and used to verify numerous distributed real-time systems [14, 19]. Borrowing ideas from the field of abstract interpretation [9, 15], we choose our approximations such that if a desired property holds for an approximating linear automaton, then it holds also for the original nonlinear automaton.

* This research was supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, and by the ARPA grant NAG2-892.

We also take algorithmic analysis a step beyond the checking of system properties. Given a parametric description of a controller, we use HyTech to automatically synthesize constraints on the safe values for the control parameters [6, 10]. These constraints are necessary and sufficient for the correctness of the approximating linear automaton, and because of our choice of approximations, they are also sufficient (though not necessary) for the correctness of the original nonlinear automaton. More accurate approximations thus provide less restrictive constraints on the controller, which can be used to control the system more efficiently.

Steam-boiler description. The steam boiler consists of a water tank, four pumps, and sensors that measure the pumping rates, the steam evacuation rate, the water level, and the operational status of each component (see Figure 1). The entire physical system operates under the guidance of a controller. The controller must keep the water level between the extreme values M_1 and M_2 at all times, and it should try to keep the water level between the normal operating levels of N_1 and N_2 as much as possible. All communication between the controller and the physical plant occurs in discrete rounds, once every Δ seconds. In each round, all units send information to the controller, and the controller responds by sending messages to the units. All communication is assumed to take place instantaneously.

The controller operates in five modes: initialization (waiting for the steam boiler to signal its readiness for operation), normal, rescue (the water-level sensor has failed), degraded (other components have failed, but the water-level sensor is working correctly), and emergency stop. In the initialization mode, the controller receives a signal that the boiler is ready, and then tests the amount of steam escaping from the boiler. If this is nonzero, it enters the emergency stop mode. Otherwise, it either drains the water level to N_2 or activates a pump to raise the water level to N_1. Once the range of normal water levels has been reached, the controller sends a signal to the physical units, waits for acknowledgements, and then proceeds with normal operation. In normal mode, the controller makes its decisions to turn pumps on or off based on the current water level, the states of the physical units, and the rate at which steam is being emitted. No action is taken if the water level lies in the range $[N_1, N_2]$. In degraded mode, some unit, other than the water-level sensor, has failed. Messages are sent to repair the faulty components, and the controller attempts to maintain correct water levels with the operational components. The reader is referred to [2] for a description of the rescue mode.

Our goal is not to provide a detailed model of the message passing between system components and the controller; to do so would result in state explosion. Rather, we focus on the high-level interactions between discrete control decisions and the continuous aspects of the underlying physical plant. To this end, we restrict most of our discussion to two fault-free pumps in normal operating mode. This simplifies the discrete control space of the system and allows us to concentrate on the continuous evolution of variables modeling the water level, steam and pumping volumes. Only in a later section do we briefly consider four

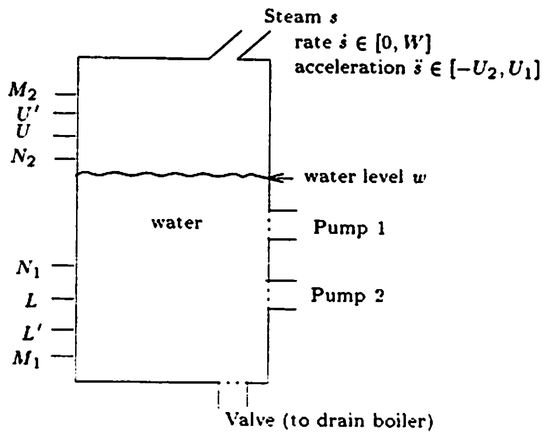


Fig. 1. Overview of the steam boiler

pumps and a simple fault-tolerant system with a degraded mode.

Steam-boiler analysis. We provide two models of the system, at two different levels of abstraction. For each model, we design controllers that ensure that whenever all physical components operate correctly, then the water level is maintained within the desired bounds, and the emergency stop mode is never entered. Our controllers rely on sensor values to determine how many pumps should be operating. The simpler the model of a system, the more complex the questions we are able to answer using HyTech. Our first model ignores all information about the second derivative of the steam output. For this simple model, we provide a controller whose decisions are based solely on the water level. We verify the controller, determine the minimal and maximal water levels that can occur, determine a safe upper bound on the time period Δ that separates consecutive rounds of communication, and determine constraints on the water-level thresholds that trigger decisions to turn a pump on or off. All of this is done completely automatically using HyTech.

The simpler the model, the more extraneous behaviors it admits. Therefore, a simple model may lead to an unnecessarily restrictive choice of control-parameter values such as water-level thresholds. Our second model achieves a closer approximation of the original system by taking into account the steam rate at the beginning and end of each round, and inferring bounds on the steam volume emitted during a round. A controller that utilizes this information is strictly superior to the previous one in that it requires the pumps to be on less often, while still maintaining the required water levels.

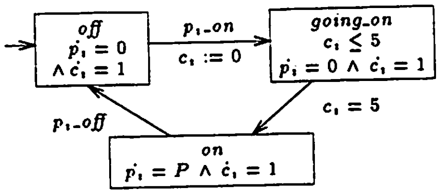


Fig. 2. Automaton for pump i

2 Hybrid Automata

We define *hybrid automata*, which are used to model mixed discrete-continuous systems [4]. Informally, a hybrid automaton consists of a finite set X of real-valued variables and a labeled multigraph (V, E). The edges E represent discrete transitions and are labeled with guarded assignments to X. The vertices V represent continuous activities and are labeled with constraints on the derivatives of the variables in X. The state of the automaton changes either instantaneously through a discrete transition or, while time elapses, through a continuous activity. We use the pump automaton in Figure 2 as an example accompanying the formal definition.

2.1 Definition

Given a set X of variables, a *predicate* over X is a boolean combination of inequalities between algebraic terms with free variables from X. A *hybrid automaton* A consists of the following components.

Variables. A finite ordered set $X = \{x_1, \ldots, x_n\}$ of real-valued *variables*, and a subset $Y = \{y_1, \ldots, y_k\} \subseteq X$ of *controlled variables*. For example, the pump automaton has two controlled variables, p_i and c_i, representing the pumping volume and a clock. A *valuation* is a point $\mathbf{a} = (a_1, \ldots, a_n)$ in the n-dimensional real space $\mathbb{R}^n$, or equivalently, a function mapping each variable x_i to its value a_i.

Control modes. A finite set V of vertices called *control modes*. The pump automaton has three control modes, *on*, *off*, and *going_on*, used for modeling the pump when it is on, off, and in the process of going on. A *state* $(v, \mathbf{a})$ of the hybrid automaton A consists of a control mode $v \in V$ and a valuation $\mathbf{a} \in \mathbb{R}^n$.

Invariant conditions. A labeling function *inv* that assigns to each control mode $v \in V$ an *invariant condition* $inv(v)$, which is a predicate over X. A state $(v, \mathbf{a})$ is *admissible* if $inv(v)[X := \mathbf{a}]$ is true. The automaton control may reside in control mode v only while the invariant condition $inv(v)$ is true; so the invariant conditions can be used to enforce progress. For example, in the pump automaton, the invariant condition $c_i \leq 5$ of the control mode *going_on* ensures that the automaton control must leave at the latest when the monotonically increasing clock c_i reaches the value 5.

Flow conditions. A labeling function *flow* that assigns to each control mode $v \in V$ a *flow condition* $flow(v)$, which is a predicate over the set $X \cup \dot{X}$ of variables, where $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$. Each dotted variable $\dot{x}_i$ represents the first derivative of the variable x_i with respect to time. While the automaton control resides in control mode v, the variables change along differentiable trajectories whose first derivatives satisfy the flow condition. Formally, for each real $\delta \geq 0$, we define the binary *flow relation* $\xrightarrow{\delta}$ on the admissible states such that $(v, \mathbf{a}) \xrightarrow{\delta} (v', \mathbf{a}')$ if $v' = v$, and there is a differentiable function $\rho\colon [0, \delta] \to \mathbb{R}^n$ such that

- the endpoints of the flow match those of ρ, *i.e.* $\rho(0) = \mathbf{a}$ and $\rho(\delta) = \mathbf{a}'$;
- the invariant condition is satisfied throughout the flow, *i.e.* for all reals $t \in [0, \delta]$, $inv(v)[X := \rho(t)]$ is true; and
- the flow condition is satisfied throughout the flow, *i.e.* for all reals $t \in (0, \delta)$, $flow(v)[X, \dot{X} := \rho(t), \dot{\rho}(t)]$ is true, where $\dot{\rho}(t) = (\rho_1(t)/dt, \ldots, \rho_n(t)/dt)$.

For example, in the pump automaton, the flow condition $\dot{p}_i = 0 \wedge \dot{c}_i = 1$ of the control mode *going_on* ensures that the pumping volume stays unchanged (at 0) and that the clock c_i measures the amount of elapsed time.

Initial conditions. A labeling function *init* that assigns to each control mode $v \in V$ an *initial condition* $init(v)$, which is a predicate over Y. The automaton control may start in control mode v when $init(v) \wedge inv(v)$ is true, where $\mathbf{a}[Y]$ denotes the restriction of $\mathbf{a}$ to the variables in Y. A state $(v, \mathbf{a})$ is *initial* if it is admissible and $init(v)[Y := \mathbf{a}[Y]]$ is true. In the graphical representation of automata, we omit initial conditions of the form *false*. For example, in the pump automaton, the initial condition of the control mode *off* is *true*, and the initial conditions of the other control modes are *false*.

Control switches. A finite multiset E of edges called *control switches*. Each control switch (v, v') has a source mode $v \in V$ and a target mode $v' \in V$. For example, the pump automaton has three control switches.

Jump conditions. A labeling function *jump* that assigns to each control switch $e \in E$ a *jump condition* $jump(e)$, which is a predicate over the set $X \cup Y'$ of variables, where $Y' = \{y'_1, \ldots, y'_k\}$. The unprimed symbol y_i refers to the value of the controlled variable before the control switch, and the primed symbol y'_i refers to its value after the control switch. Only controlled variables are updated by a control switch. Formally, for the control switch $e = (v, v')$, we define the binary *jump relation* $\xrightarrow{e}$ on the admissible states such that $(v, \mathbf{a}) \xrightarrow{e} (v', \mathbf{a}')$ iff

- $jump(e)[X, Y' := \mathbf{a}, \mathbf{a}'[Y]]$ is true; and
- for all uncontrolled (environment) variables $x_i \in X \setminus Y$, $\mathbf{a}'_i = \mathbf{a}_i$.

The control switch e is *enabled* in the valuation $\mathbf{a}$ if there exists a state $(v', \mathbf{a}')$ such that $(v, \mathbf{a}) \xrightarrow{e} (v', \mathbf{a}')$. We use nondeterministic guarded interval-valued assignments to write jump conditions. For example, we write $\phi \to y_i := [l, u]$ for the jump condition $\phi \wedge l \leq y'_i \leq u \wedge \bigwedge_{j \neq i} y'_j = y_j$, where l and u are predicates over X. Intuitively, a control switch is enabled in the valuation $\mathbf{a}$ if the guard is satisfied, *i.e.* $\phi[X := \mathbf{a}]$ is true. Then, the controlled variable y_i is updated nondeterministically to any value in the interval $[l[X := \mathbf{a}], u[X := \mathbf{a}]]$. In the graphical representation of automata, guards of the form *true* and identity assignments are omitted. For example, in the pump automaton, the control switch from *going_on* to *on* has the jump condition $c_i = 5 \wedge p_i' = p_i \wedge c_i' = c_i$.

Events. A finite set Σ of *visible events*, and a labeling function *event* that assigns to each control switch $e \in E$ either a visible event from Σ or the internal event τ. The *internal event* τ is contained neither in Σ, nor in the set of visible events of any other automaton. The event labels are used to define the parallel composition of automata. Internal events are omitted in the graphical representation of automata. For example, in the pump automaton, $\mathit{event}(\mathit{off}, \mathit{going\_on}) = p_i\_\mathit{on}$ and $\mathit{event}(\mathit{going\_on}, \mathit{on}) = \tau$.

2.2 Parallel composition

Nontrivial systems consist of several interacting components. We model each component as a hybrid automaton, and the components coordinate with each other through both shared variables and events. For example, the controller communicates with the i-th pump by synchronizing control switches with the events p_i_on and p_i_off. A hybrid automaton that models the entire system is obtained from the component automata using a product construction.

Let A_1 be the hybrid automaton $(X_1, Y_1, V_1, \mathit{inv}_1, \mathit{flow}_1, \mathit{init}_1, E_1, \mathit{jump}_1, \Sigma_1, \mathit{event}_1)$, and define A_2 similarly. In the product automaton $A_1 \times A_2$, two control switches e_1 and e_2 from the two component automata A_1 and A_2 occur simultaneously if $\mathit{event}_1(e_1) = \mathit{event}_2(e_2)$. They are interleaved if $\mathit{event}_1(e_1) \neq \mathit{event}_2(e_2)$ and neither $\mathit{event}_1(e_1)$ is a visible event of A_2, nor $\mathit{event}_2(e_2)$ is a visible event of A_1. Internal events may occur simultaneously or interleaved. Formally, provided Y_1 and Y_2 are disjoint, the *product* $A_1 \times A_2$ of A_1 and A_2 is the following hybrid automaton $A = (X_1 \cup X_2, Y_1 \cup Y_2, V_1 \times V_2, \mathit{inv}, \mathit{flow}, \mathit{init}, E, \mathit{jump}, \Sigma_1 \cup \Sigma_2, \mathit{event})$.

Control modes. Each control mode (v_1, v_2) in $V_1 \times V_2$ has the invariant condition $\mathit{inv}(v_1, v_2) = \mathit{inv}_1(v_1) \wedge \mathit{inv}_2(v_2)$, the flow condition $\mathit{flow}(v_1, v_2) = \mathit{flow}_1(v_1) \wedge \mathit{flow}_2(v_2)$, and the initial condition $\mathit{init}(v_1, v_2) = \mathit{init}_1(v_1) \wedge \mathit{init}(v_2)$.

Control switches. E contains the control switch $e = ((v_1, v_2), (v_1', v_2'))$ if

(1) $e_1 = (v_1, v_1') \in E_1$, $v_2' = v_2$, and $\mathit{event}_1(e_1) \notin \Sigma_2$; or
(2) $e_2 = (v_2, v_2') \in E_2$, $v_1' = v_1$, and $\mathit{event}_2(e_2) \notin \Sigma_1$; or
(3) $e_1 = (v_1, v_1') \in E_1$, $e_2 = (v_2, v_2') \in E_2$, and $\mathit{event}_1(e_1) = \mathit{event}_2(e_2)$.

In case (1), $\mathit{event}(e) = \mathit{event}_1(e_1)$ and $\mathit{jump}(e) = \mathit{jump}_1(e_1) \wedge \bigwedge_{y \in Y_2} y' = y$. In case (2), $\mathit{event}(e) = \mathit{event}_2(e_2)$ and $\mathit{jump}(e) = \mathit{jump}_2(e_2) \wedge \bigwedge_{y \in Y_1} y' = y$. In case (3), $\mathit{event}(e) = \mathit{event}_1(e_1) = \mathit{event}_2(e_2)$ and $\mathit{jump}(e) = \mathit{jump}_1(e_1) \wedge \mathit{jump}_2(e_2)$.

2.3 Verification

Let A be a hybrid automaton with n variables and the control graph (V, E). A subset of the state space $S = V \times \mathbb{R}^n$ is called a *region*. We define the binary *transition relation* $\rightarrow$ on S as $\bigcup_{e \in E} \xrightarrow{e} \cup \bigcup_{\delta \in \mathbb{R}_{\geq 0}} \xrightarrow{\delta}$. For a region R, the *successor region post*(R) is the set of states that are reachable from some state in R via a single transition, *i.e.* $post(R) = \{s' \mid \exists s \in R. s \rightarrow s'\}$. The *reachable region reach*(A) is the set of states that are reachable from some initial state via any finite number of transitions, *i.e.* $reach(A) = \bigcup_{i \geq 0} post^i(R)$ for the set R of initial states of A.

Safety and timing analysis. Safety and timing verification problems can be posed in a natural way as reachability problems. For this purpose, the system is composed with a special monitor automaton that "watches" the execution of the system and enters a violation state whenever the system violates a given safety or timing requirement. The automaton A is correct with respect to the region $T \subseteq S$ of violation states if $reach(A) \cap T$ is empty.

Parameter synthesis. A system description often contains symbolic (unknown) constants, which we call *system parameters*. We are interested in the problem of finding the values of the parameters for which the system is correct. When the correctness criterion is a safety or timing requirement, expressed via a monitor with violation states, then necessary and sufficient conditions on the parameter values can be discovered automatically using reachability analysis. We model a parameter as a controlled variable whose derivative is 0 in every control mode, and whose value is left changed by every control switch. Then, the system is correct for precisely those parameter values for which the region $reach(A) \cap T$ is empty.

2.4 Linear hybrid automata and the tool HyTech

Linear hybrid automata are a subclass of hybrid automata that can be analyzed algorithmically. A *linear term* is a linear combination of variables with rational coefficients. A *convex linear predicate* is a conjunction of inequalities between linear terms. A hybrid automaton A is *linear* if (1) all invariant, flow, initial, and jump conditions of A are convex linear predicates, and (2) no flow condition of A contains undotted variables, *i.e.* all flow conditions are convex linear predicates over the set $\dot{X}$ of first derivatives. Hence, in flow conditions, linear dependencies between the rates of variables can be expressed, but the current flow must be independent of the current state. The convexity restrictions can be achieved by splitting control modes and control switches, if necessary. For example, the pump automaton is a linear hybrid automaton.

A region is *linear* if it can be defined by a disjunction of convex linear predicates. The computation of the successor region $post(R)$ is effective for a linear region R, and yields again a linear region [5]. Therefore, the reachability problem, which can be solved by iterating successor computations, is semidecidable for linear hybrid automata. Our analysis of linear hybrid automata is performed using the symbolic model checker HyTech [17].[3]

2.5 Approximation

While it may be difficult to reason automatically about a complex automaton, in order to establish a particular property, it may suffice to prove a related property for a simpler approximating automaton. If the correctness criterion for an automaton A is a safety property, expressed via a set of violation states, then it suffices to consider an approximating automaton B such that $\mathit{reach}(A) \subseteq \mathit{reach}(B)$. Then, the safety property holds for A if it can be verified for B. The same idea applies to parametric analysis. Suppose that we can find necessary and sufficient parameter constraints for the correctness of the approximating automaton B. Since more states are reachable in B than in A, all parameter values that cause violations in A also cause B to be incorrect. Hence, the parameter constraints for the correctness of B are sufficient for the correctness of the original automaton A. However, it may not be the case that the conditions are necessary, and a closer approximation may yield more permissive conditions.

The use of HyTech demands approximation if the hybrid automaton A to be analyzed is nonlinear. In this case, we approximate A by a linear hybrid automaton B [13, 18]. For a system with several components, it may suffice to approximate only the nonlinear components. The steam-boiler system contains only one nonlinear component —the automaton that models the steam— and we replace it first by a very rough and then by a more accurate linear approximation.

3 Steam-boiler Description

We model the steam-boiler system at various levels of detail. The primary modeling issues are (1) modeling the nonlinear behavior of the steam exiting the boiler, (2) how many of the failure modes are considered, and (3) the design of the controller. To facilitate algorithmic analysis, we first restrict the state space by considering only two pumps and omitting the failure modes. We use the following variables: p_i for the volume of water pumped by pump i, c_i for a clock of pump i, v for the volume of water drained through the valve, w for the water level in the boiler, s for the volume of steam emitted from the boiler, r for the rate at which steam is emitted from the boiler, t_s for an auxiliary clock for modeling steam emission, and t for the clock of the control program.

Pumps. The automaton for pump i appears in Figure 2. Control is initially in the mode *off*, where the flow condition $\dot{p_i} = 0$ indicates that the pump is idle. The pump synchronizes with the controller automaton on the events p_i_on, through which the controller commands the pump to be turned on, and p_i_off, through which the pump is turned off. The pump takes 5 seconds to respond to the command p_i_on to begin pumping, modeled as follows. The variable c_i is a clock, with first derivative equal to 1 at all times. When the switch from mode

[3] HyTech can be obtained on the web at http://www.eecs.berkeley.edu/~tah.

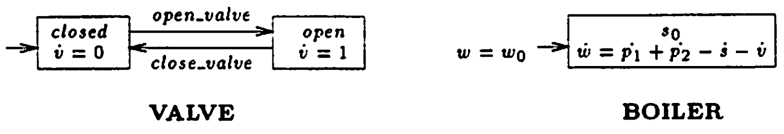


Fig. 3. Valve and boiler automata

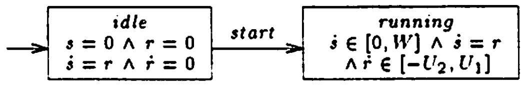

idle
$s = 0 \wedge r = 0$
$\dot{s} = r \wedge \dot{r} = 0$
start
running
$\dot{s} \in [0, W] \wedge \dot{s} = r$
$\wedge \dot{r} \in [-U_2, U_1]$

Fig. 4. Nonlinear steam model A^S

off to mode *going_on* occurs, the clock c_i is set to 0 and measures the delay before water is actually pumped. The invariant condition $c_i \leq 5$ together with the guard $c_i = 5$ on the switch to mode *on* ensures that the delay is exactly 5 seconds. The flow condition $\dot{p}_i = P$ in mode *on* reflects our assumption that when the pump is on, it operates at its maximal capacity P.

Valve. The valve automaton appears on the left of Figure 3. The controller automaton communicates with the valve through the events *open_valve* and *close_valve*. We assume that the valve opens and closes instantaneously, and we assume a draining rate of 1 liter per second. The valve is used only in the initialization phase.

Boiler. The boiler automaton, appearing on the right of Figure 3, uses a single control mode to model the relationship between the water level and the flow rates into and out of the boiler. The water level is initially w_0. For simplicity, we refer to the value of the water level and the water volume interchangeably, whereas in reality they are related via a formula dependent on the geometry of the boiler tank. Notice that the rate at which the water level changes and the pumping, steam, and drain rates, are all measured in liters per second.

Steam. The behavior of the steam is specified by the nonlinear automaton in Figure 4. The two control modes reflect whether the physical system is running or not. When the boiler system is active, the rate of the steam emission is bounded by 0 and W, *i.e.* $\dot{s} \in [0, W]$, and the acceleration of the steam emission is bounded by $-U_2$ and U_1, *i.e.* $\ddot{s} \in [-U_2, U_1]$. Since the hybrid-automaton model does not allow the direct modeling of higher derivatives, we represent information about the second derivative of the steam volume s using the auxiliary variable r, which represents the first derivative of s. Hence $\dot{s} = r$ and $\ddot{s} = \dot{r}$. The flow condition $\dot{s} = r$ is not linear, and cannot be directly analyzed using HYTECH. For automatic analysis, we will therefore approximate the steam automaton by

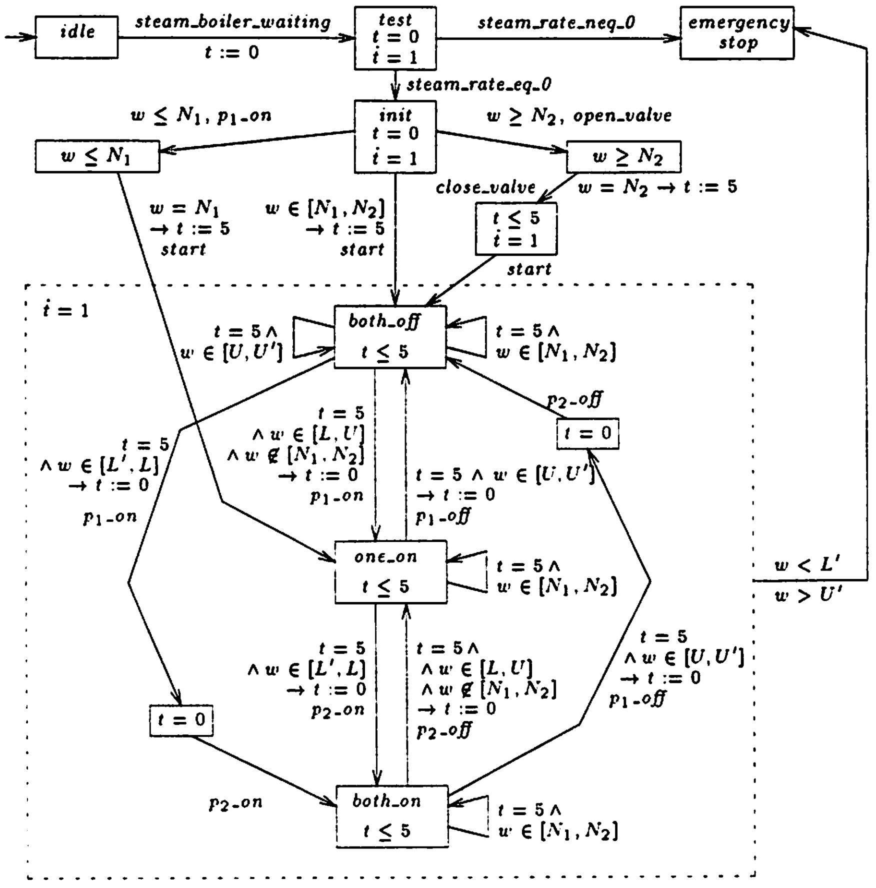


Fig. 5. First controller automaton C_A

linear models.

Controller. The controllers we use vary according to which model of steam is being used. Here, we describe the basic controller in Figure 5, which uses only information about the water level to decide how many pumps should be active. The controller assumes that the physical components never malfunction. It performs initialization, and then operates in normal mode, until disaster causes abortion to the emergency-stop mode.

Initially, control is in mode *idle*. The controller waits to receive a *steam_boiler_waiting* signal from the physical system, indicating that the steam boiler is ready and waiting for operation. It then checks whether the steam rate is zero (modeled by the event *steam_rate_eq_0*) or nonzero (modeled by the event *steam_rate_neq_0*), reads the initial water level, and if necessary, either opens the valve, or acti-

Water level w	Pumps
$w < L'$	emergency stop
$L' \leq w \leq L$	both on
$L \leq w \leq N_1$	one on, one off
$N_1 \leq w \leq N_2$	no control action
$N_2 \leq w \leq U$	one on, one off
$U \leq w \leq U'$	both off
$U' < w$	emergency stop

Fig. 6. Control strategy

vates the first pump. until the water level is within the normal operating range $[N_1, N_2]$.

During normal operation, the controller disregards all information except for the water level when making its control decision. It uses four parameters for the water level —L', L, U, and U'— to decide how many pumps should be active (see Figure 1). Normal operating mode contains three submodes, depending on whether zero, one, or two pumps are intended to be on. The table in Figure 6 shows the strategy of the controller. We assume that whenever only one pump is active, the system uses pump 1. Normal operating mode of the controller is indicated by the dashed box in Figure 5. All submodes within the dashed box inherit the flow condition $\dot{t} = 1$ and the outgoing control switches to the emergency-stop mode.

Safety requirements. The steam boiler should meet the requirement that the water level is always between M_1 and M_2, provided it is initially between L and U. We also require that the emergency-stop mode is never entered.[4] Thus the violation states are defined by the region containing all admissible states for which $w < M_1 \vee w > M_2$ is true, and all admissible states whose controller component is the emergency-stop mode.

4 Steam-boiler Analysis

We analyze two linear approximations of the nonlinear steam automaton. For the simpler steam model, A, we first verify the safety requirements for fixed values of all system constants. Then we perform three parametric analyses: we determine the minimal and maximal water levels that can occur; we determine the safe values of the sampling rate, holding all other constants fixed; and we determine constraints on the safe values for the control parameters L', L, U, and U', holding all other constants fixed. For the more accurate steam model, B, we show how information about the steam rate can be used to design a more flexible control algorithm that reduces the use of the pumps.

[4] This is a reasonable requirement given the assumption that no components fail.

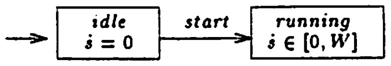


Fig. 7. First steam approximation A^S_A

4.1 Model A: ignoring the steam acceleration

Consider the simple model of the steam output shown in Figure 7. We assume that the rate $\dot{s}$ of the steam emission varies between the two bounds 0 and W, and we completely disregard all information about the acceleration $\ddot{s}$ of the steam emission. It is easy to see how this automaton is derived from that of Figure 4 by dropping all predicates that restrict the variable r. Clearly, $reach(A^S) \subseteq reach(A^S_A)$.

Verification. We fix the values of the system constants as follows: sampling time $\Delta = 5$ seconds, maximal steam rate $W = 6$ liters per second, pumping capacity $P = 4$ liters per second, interval of normal water levels $[N_1 = 100, N_2 = 150]$ liters, interval of acceptable water levels $[M_1 = 5, M_2 = 220]$ liters. Using HyTech, we verify that the controller with $L' = 25$, $L = 70$, $U = 170$, and $U' = 200$ maintains the water level within the required bounds.

Parametric analysis of the achievable water levels. We can use HyTech to determine the exact minimal and maximal bounds on the water level for the steam model A^S_A. For this purpose, we introduce two parameters, *min_level* and *max_level*. These play no role in the automaton descriptions, *i.e.* they are symbolic constants whose values never change. We perform reachability as before, and use as violating states all admissible states where $w \leq min\_level \vee w \geq max\_level$ is true. Then, there is a violation whenever the water level exceeds *max_level* or drops below *min_level*. HyTech outputs the following condition for unsafe parameter values

```
max_level <= 190 | min_level >= 30
```

from which we infer that the maximal water level reached is 190, and the minimal water level reached is 30. Since the steam model A^S_A is a conservative overapproximation of the nonlinear behaviors of the steam automaton A^S, we conclude that these bounds are true upper and lower bounds for the physical nonlinear system.

Parametric analysis of the sampling time. According to the steam-boiler specification, all communication and control occurs in rounds that are separated by intervals of 5 seconds. We can use HyTech to determine how much the delay between consecutive rounds can be stretched without violating the safety requirements. For this purpose, we introduce the parameter Δ, which represents the sampling time, *i.e.* the time between two consecutive rounds. Using the original set of violation states, HyTech outputs the (unsimplified) condition for unsafe sampling times as

```
2delta >= 15 | 4delta >= 25 & 100 >= 4delta
```

from which we infer that the system is correct as long as the sampling time is less than 25/4 seconds.

Parametric analysis of the control parameters. If the designer is free to choose values for the threshold parameters L', L, U, and U', HyTech can be used to determine which relationships between these control parameters ensure the safety requirements of the steam-boiler system. For this purpose, we introduce the four parameters L', L, U, and U'. We add the consistency assumptions $M_1 \leq L' \leq L \leq N_1$ and $N_2 \leq U \leq U' \leq M_2$ to the initial conditions of the system. Then HyTech automatically generates the following condition[5] as necessary and sufficient for the correctness of the abstracted system: $U' > U + 20 \wedge (U' > 190 \vee L < 60) \wedge L > L' + 20 \wedge (L < 40 \vee L' < 30 \vee L > L'+40) \wedge (L' < 40 \vee L < 70)$. We give an intuitive explanation of each of these conjuncts. Recall that the sampling time is $\Delta = 5$ seconds, the pumping rate is $P = 4$ liters per second per pump, and the maximal steam emission rate is $W = 6$ liters per second.

$U' > U + 20$: Suppose that the water level is exactly U liters and a single pump is running. If the steam emitted during the next round is minimal, *i.e.* 0 liters, then the water level may rise as high as $U + 20$ liters by the end of the round. If U' is not greater than this value, then the controller must enter the emergency-stop mode the next time the water level is checked.

$U' > 190 \vee L < 60$: This condition is equivalent to $L \geq 60 \Rightarrow U' > 190$. Assume that $L \geq 60$. We show that then $U' > 190$. Clearly there are situations in which both pumps will be turned on, because the water level can easily drop from the normal range above $N_1 = 100$ liters down to below 60 liters. Suppose that the controller has to make a decision when the water level is exactly L liters with both pumps active. It may leave both pumps running through the next round. Minimal steam emission over this period causes the next water-level reading to lie in the normal range [100, 150], because the net input of the two pumps is 8 liters per second. Within the range of normal water levels, no control action is taken, so both pumps are still active. It is therefore possible for a future water-level reading to occur at exactly $N_2 = 150$, and again both pumps remain operating. Minimal steam emission causes the water level to rise 40 more liters to 190 liters. Now if U' is not above 190, then the controller must enter the emergency-stop mode.

$\phi_0 : L > L' + 20$: The explanation here is similar to that for the constraint $U' > U+20$ above. The controller may decide to leave exactly one pump running when it reads the water level as L. The earliest time that the second pump could be active is 10 seconds later, because the next decision is made after 5 seconds, and there is a delay of 5 seconds before an activated pump begins pumping water. Within 10 seconds, the water level may drop by 20 liters.

$\phi_1 : L < 40 \vee L' < 30 \vee L > L' + 40$: Intuitively, this condition addresses the spacing between L and L' when the parameters are set close to the minimal

[5] HyTech's output is the conjunction of the consistency assumptions and the negation of this condition.

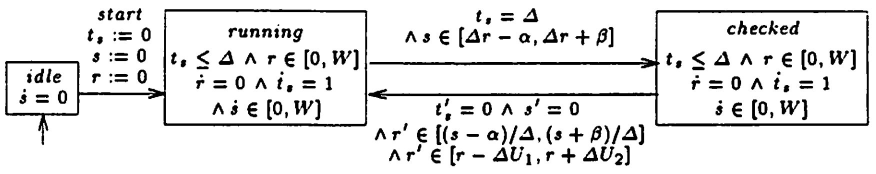

start
$t_s := 0$
$s := 0$
$r := 0$
idle
$\dot{s} = 0$
running
$t_s \leq \Delta \wedge r \in [0, W]$
$\dot{r} = 0 \wedge \dot{t}_s = 1$
$\wedge \dot{s} \in [0, W]$
$t_s = \Delta$
$\wedge s \in [\Delta r - \alpha, \Delta r + \beta]$
checked
$t_s \leq \Delta \wedge r \in [0, W]$
$\dot{r} = 0 \wedge \dot{t}_s = 1$
$\dot{s} \in [0, W]$
$t'_s = 0 \wedge s' = 0$
$\wedge r' \in [(s - \alpha)/\Delta, (s + \beta)/\Delta]$
$\wedge r' \in [r - \Delta U_1, r + \Delta U_2]$

Fig. 8. Second steam approximation A_B^S

normal water level N_1. Assume that $L' \geq 30$. We show that $L > L' + 40$. Consider the following scenario. The controller reads the water level when it is exactly $N_1 = 100$ with no pumps active, reads it again at 70 liters and activates only one pump. By the end of the next round, the water level drops to 40 liters, and then the first pump is active in the round after that, leading to a water level of 30 liters. If $L' \geq 30$, then the controller will enter the emergency-stop mode. To avoid this kind of situation, it is necessary for both pumps to be active before the water level drops to L'. In particular, the above scenario shows that L must be above 70. It is therefore possible for the controller to read the water level as L with no pumps active, and for the water level to drop to $L - 40$ over the next two rounds. Thus we conclude that $L' < L - 40$.

ϕ_2: $L' < 40 \vee L < 70$: First, observe that the conjunction $\phi_0 \wedge \phi_1 \wedge \phi_2$ is equivalent to $\phi_0 \wedge \phi_1 \wedge L' < 40$, because $\phi_0 \wedge \phi_1 \wedge L < 70$ implies $L' < 40$. The condition ϕ_2 can therefore be simplified to $L' < 40$, but currently HYTECH does not perform minimality checks on expressions in conjunctive normal form. It is easy to see that $L' < 40$ is a necessary condition. From a water level of $N_1 = 100$, it may take two rounds before a pump is activated, and during that time a total of 60 liters of water may be lost.

In hindsight, the necessity of each of the above conditions is not difficult to explain. However, it would have been nontrivial to manually generate each one, and justify that their conjunction is sufficient for the safe operation of the boiler. As explained in Subsection 2.5, the conditions are necessary and sufficient for the approximating linear system, but only sufficient for the correctness of the approximated nonlinear system.

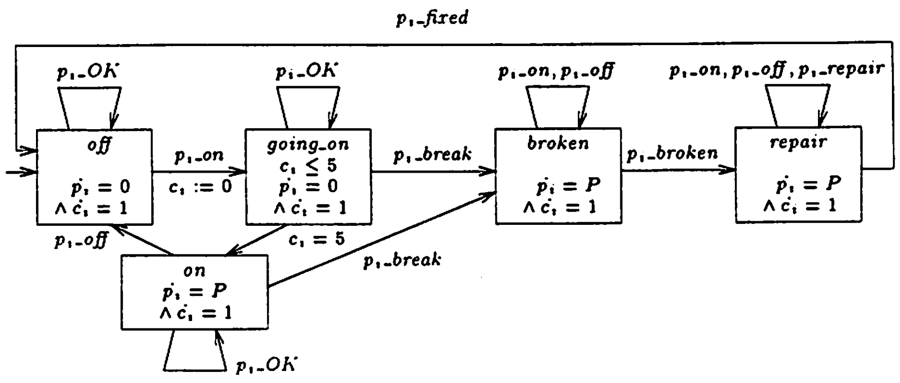
4.2 Model B: linear approximation of the steam acceleration

We now give a closer approximation of the steam automaton, which allows us to prove the correctness of a controller that lets the water level drop lower than previously before operating the pumps. We use the variable r to represent the steam rate at the end of each round. The consistency between the steam volume emitted during a round and the steam rate is maintained by ensuring that (1) the emitted steam volume is consistent with the steam rate at the beginning of the round and the possible steam acceleration, and (2) the steam rate at the end of the round is consistent with the emitted steam volume and the possible steam acceleration. The linear hybrid automaton for this steam model appears in Figure 8. The derivative of the variable r is always 0. If the steam rate at the beginning of a round is $\dot{s} = r_0$, and the steam emission accelerates constantly at k liters per second per second, then the volume of steam emitted over the next Δ seconds is $\int_0^{\Delta}(r_0 + kt)dt = \Delta r_0 + \frac{1}{2}k\Delta^2$. Thus the steam emitted lies in the range $[\Delta r_0 - \alpha, \Delta r_0 + \beta]$, where $\alpha = \frac{1}{2}\Delta^2 U_1$ and $\beta = \frac{1}{2}\Delta^2 U_2$. The jump condition on the control switch from mode *running* to mode *checked* enforces the consistency condition (1), and the jump condition from *checked* to *running* enforces the consistency condition (2).

Improved controller. If the controller has access to the measured steam rate r, in addition to the water level w, then it can bound more tightly the possible future behaviors of the boiler. It may therefore be able to maintain the water level correctly in situations where this is not possible without a steam-rate measurement. In this subsection, we assume that the controller keeps the pumps either both active or both idle at any given time. The key control decision is when to turn on the pumps. The controller (see the appendix) makes this decision at the end of each round based on the water level w and the steam rate r. As before, when the water level falls below the control parameter L', or rises above U', then the controller enters the emergency-stop mode. For water levels between L' and N_1, the controller activates both pumps if $w \leq \varphi(r)$, and determines that the pumps should be idle if $w > \varphi(r)$, where the values of the function φ are sufficiently high to avoid the water level dropping to L' over the next two rounds. The controller turns off both pumps whenever the water level is between N_2 and U', and aborts when the level is above U'.

We define the function φ. A decision not to turn on the pumps means that at least two rounds will pass before the pumps begin actively filling the boiler tank. The controller must ensure that no disaster can occur during this period. Leaving the pumps inactive will not cause the water level to drop to L' if $w - \mathit{max2}(r) > L'$, where $\mathit{max2}(r)$ is the maximal amount of steam emitted during the next 2Δ seconds, given a current steam rate of r. However, $\mathit{max2}(r)$ is quadratic, and therefore cannot be expressed in a linear hybrid automaton. We instead use the upper bound $\mathit{Max2}(r)$ on $\mathit{max2}(r)$, which is defined as $\mathit{Max2}(r) = \min(2W\Delta, 2r\Delta + 2\Delta^2 U_2)$. Thus the function φ is defined by $\varphi(r) = L' + \mathit{Max2}(r)$.

Verification of the improved controller. Given the bounds $U_1 = U_2 = 2/5$ on the steam acceleration, HyTech verifies that the controller C_B, with the control parameters set to $L' = 25$ and $U' = 200$, guarantees the safety requirements. This controller is more flexible than a comparable controller that relies on the simple steam model A_A^S. Consider a simple controller C'_A (see the appendix) that activates both pumps between L' and L, and neither pump between L and N_1 and between N_2 and U'. Parametric analysis of C'_A for $L' = 25$ and $U' = 200$ determines that the safe values of L are characterized by $L \geq 85$. The improved controller C_B is more relaxed than the simple controller C'_A, in that it does not activate the pumps as often. For a given steam rate r, the threshold value for turning on the pumps is $\varphi(r) = L' + \mathit{Max2}(r)$. When the steam rate is high



p_i_fixed
p_i_OK
p_i_OK
p_i_on, p_i_off
p_i_on, p_i_off, p_i_repair
off
$\dot{p}_i = 0 \wedge \dot{c}_i = 1$
p_i_on
$c_i := 0$
going_on
$c_i \leq 5$
$\dot{p}_i = 0 \wedge \dot{c}_i = 1$
p_i_break
broken
$\dot{p}_i = P \wedge \dot{c}_i = 1$
p_i_broken
repair
$\dot{p}_i = P \wedge \dot{c}_i = 1$
p_i_off
$c_i = 5$
p_i_break
on
$\dot{p}_i = P \wedge \dot{c}_i = 1$
p_i_OK

Fig. 9. Automaton for pump i, assuming possible failure

(*i.e.* above 4), then $\varphi(r) = 85$, and both pumps are activated just as in C'_A. But when the steam rate r is lower than 4, then the threshold value $\varphi(r)$ is only $L' + 10r + 50U_2 = 45 + 10r$. In particular, if $r = 0$, then the improved controller allows the water level to drop as low as 45 before turning on the pumps.

4.3 A fault-tolerant system

Finally, we describe a simple fault-tolerant model of a steam-boiler system where pumps may fail. The new model of pump i appears in Figure 9. It includes a *broken* mode and a *repair* mode. The pump may fail at any time other than when it is in the *off* mode. It remains in the *broken* mode until the controller is informed of the status of the pump via the events p_i*_OK* and p_i*_broken*, at which point the pump may be repaired and restored to the *off* mode. While broken or in the process of being repaired, the pump delivers water at any rate between 0 and P.

The controller (see the appendix) attempts to maintain the water level within the required bounds by using operational pumps wherever possible. It aborts to the emergency-stop mode if it cannot guarantee a safe water level for the next two rounds, assuming that any active pump may break at any time. The control decisions are based on five control parameters: if the water level is below *lo_abort*, then the controller aborts because of a danger of reaching the minimal allowable level; it activates all functional pumps if the water level is between *lo_abort* and L, and at most one functional pump if the water level is between L and N_1; if the water level is above *hi_abort_both_off* with both pumps off, or it is above *hi_abort_one_off* with exactly one pump on, or it is above *hi_abort_none_off* with both pumps on, then the controller also aborts.

It makes no sense to check the same safety requirements as before, because in the presence of pump faults it is impossible to avoid extreme water levels.

We instead show two properties of the fault-tolerant system. First, if no pumps ever break (*i.e.* we remove the p_i_break edges), then the system never enters the emergency-stop mode, and the water level remains within the required bounds. Second, if the system can be completely shutdown (*i.e.* no steam being emitted and no water being pumped) within 5 seconds of entering the emergency-stop mode, then the water level is always safe. HyTech verifies both these properties for the system with two pumps and the simple steam model A_A^S.

4.4 Computational data

The analysis reported in this paper was performed on a Sun Sparcstation 5 with 32 MBytes of main memory and additional swap space. For the simple steam model A_A^S, HyTech requires 5 seconds and 3 MBytes of memory for verification, 7s (5MB) to generate bounds on the water level, 11s (7MB) for synthesizing the sampling time, and 62s (8MB) for analyzing the control parameters. For the more sophisticated steam model B, HyTech requires 119s (9MB) for verification.

The complexity of our models has been restricted by HyTech's computational capacity. For a boiler with four pumps and the simple steam model A, HyTech completes the verification in 33s (15MB). The fault-tolerant system with a degraded mode strains resources, taking 123 seconds and 39 MBytes. However. for neither system does a parametric analysis of the allowable sampling rates complete.

5 Evaluation and Comparison

Hybrid automata enable a natural, yet mathematically precise, modeling of the steam-boiler system. Both discrete and continuous phenomena are modeled directly in one integrated formalism. The theory of abstractions for this formalism ensures that the automatic analysis results apply to the original mixed discrete-continuous system. Because our modeling language includes continuous information about the physical behavior of the boiler, we were able to discover nontrivial control constraints, *e.g.* a lower bound on the safe delay between rounds of communication, and parametric correctness criteria for the water thresholds used by the controller. The advantage of an expressive model comes at the cost of computationally expensive automated analysis. We presented fault-free components as well as a simple fault-tolerant system because of restrictions on the size of the model that HyTech can currently handle.

Designing the nonlinear hybrid automata was reasonably straightforward, even for the full fault-tolerant system (not shown in this paper). We expect an average programmer with some background in a state-machine formalism to have little difficulty understanding our model, and to be able to produce a similar solution after a couple of weeks' training[6]. The majority of time developing the

[6] A U.C. Berkeley graduate student took three weeks to learn the formalism from scratch, acquaint herself with HyTech, and generate a solution to a nonlinear control problem of similar complexity.

solution was spent in the iterative process of abstracting the nonlinear model to linear models, designing a suitable controller, testing whether HYTECH is able to produce useful results, and then redesigning the abstraction and the controller. The simplest abstraction adn its controller was easy to derive, but produces coarse results that may be too conservative to establish system correctness for safe parameter combinations. On the other hand, more accurate approximations sometimes cause HYTECH to fail, due to either memory overflow, arithmetic overflow in solving linear constraints, or a nonterminating sequence of successor computations. Finding a useful balance between detail and abstraction, and modeling the system to optimize the use of HYTECH's analysis algorithms, is still somewhat of an art.

Our goal is the *direct* and *automatic* analysis of mixed discrete-continuous systems such as the steam boiler. Among the papers in this volume, many abstract the continuous behavior into a discrete system (*e.g.* [8, 11, 22]) or into a simple timed system (*e.g.* [21, 24]), which may then be subjected to automated analysis. Others analyze the continuous behavior deductively with the possible assistance of a mechanical proof checker (*e.g.* [7, 12, 20, 23]). This paper is unique in that we retain continuous information about the boiler's physical quantities during the automated analysis. While computational complexities force us to keep our models simpler and smaller compared to discrete-state based methods, our approach allows a fully automatic synthesis of constraints on parameters that control continuous behavior.

References

1. J.-R. Abrial, E. Börger. and H. Langmaack. The steam-boiler case study project. An introduction. This volume.
2. J.-R. Abrial. Steam-boiler control specification problem. This volume.
3. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
4. R. Alur, C. Courcoubetis, T.A. Henzinger. and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems I*, Lecture Notes in Computer Science 736, pp. 209–229. Springer-Verlag, 1993.
5. R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Engineering*, 22:181–201, 1996.
6. R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. *Proc. Symp. Theory of Computing*, pp. 592–601. ACM Press, 1993.
7. R. Buessow and M. Weber. A steam-boiler control specification with STATECHARTS and Z. This volume.
8. T. Cattel and G. Duval. The steam-boiler problem in LUSTRE. This volume.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. *Proc. Symp. Principles of Programming Languages*. ACM Press, 1977.

10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. *Proc. Symp. Principles of Programming Languages*. ACM Press, 1978.
11. G. Duval and T. Cattel. Specifying and verifying the steam-boiler problem with SPIN. This volume.
12. G. Leeb and N. Lynch. Proving safety properties of the steam-boiler controller. This volume.
13. T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. *Computer-aided Verification*, Lecture Notes in Computer Science 939, pp. 225–238. Springer-Verlag, 1995.
14. T.A. Henzinger and P.-H. Ho. HYTECH: The Cornell Hybrid Technology Tool. *Hybrid Systems II*, Lecture Notes in Computer Science 999, pp. 265–293. Springer-Verlag, 1995.
15. T.A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. *Hybrid Systems II*, Lecture Notes in Computer Science 999, pp. 252–264. Springer-Verlag, 1995.
16. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: The next generation. *Proc. Real-time Systems Symp.*, pp. 56–65. IEEE Computer Society Press, 1995.
17. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HYTECH. *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1019, pp. 41–71. Springer-Verlag, 1995.
18. T.A. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pp. 377–388. Springer-Verlag, 1995.
19. P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. *Computer-aided Verification*, Lecture Notes in Computer Science 939, pp. 381–394. Springer-Verlag, 1995.
20. X.-S. Li and J. Wang. Specifying optimal design of a steam-boiler system. This volume.
21. P.C. Olveczky, P. Kosiuczenko, and M. Wirsing. An object-oriented algebraic steam-boiler control specification. This volume.
22. C. Schinagl. VDM specification of the steam-boiler control using RSL notation. This volume.
23. J. Vitt and J. Hooman. Assertional specification and verification using PVS of the steam-boiler control system. This volume.
24. A. Willig and I. Schieferdecker. Specifying and verifying the steam-boiler control system with time extended LOTOS. This volume.

A Appendix: HyTech Input Files

We provide the HyTech input files for the following analyses:

1. Verification of the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S
2. Parametric analysis of the achievable water levels of the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S
3. Parametric analysis of the sampling time for the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S
4. Parametric analysis of the control parameters for the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S
5. Verification of the fault-free four-pump steam-boiler system with the steam model A_A^S
6. Verification of the fault-tolerant two-pump steam-boiler system with steam model A_A^S
7. Parametric analysis of the control parameter L in the dual-pump fault-free steam-boiler system with controller C'_A and steam model A_A^S
8. Verification of the dual-pump fault-free steam-boiler system with controller C_B and steam model A_B^S

All but the last analysis use the simple model of the steam that ignores the steam acceleration.

A.1 Verification of the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S

```
---------------------------
-- HyTech input file
--
-- Steam boiler
--

define(P_rate,4)
define(W1,0) -- min steam rate
define(W2,6) -- max steam rate

define(MIN,5)
define(param_l_prime,25)
define(param_l,70)
define(N_1,100)
define(N_2,150)
define(param_u,170)
define(param_u_prime,200)
define(MAX,220)

define(delta,5)
```

```
var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        p2,     -- pump volume from Pump 2 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        t,      -- controller's clock
        t1,     -- pump controller 1's clock
        t2      -- pump controller 2's clock
          : clock;

--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw = dp1 + dp2 - dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

synclabs: p_1_on, p_1_off;
initially off ;

loc off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto going_on;

loc going_on: while t1<=5 wait {dp1=0}
        when t1=5 do {t1'=t1'} goto on;

loc on: while True wait {dp1=P_rate}
        when True sync p_1_off goto off;
end


--------------------------------------------------

automaton pump_cont_2

synclabs: p_2_on, p_2_off;
initially off ;

loc off: while True wait {dp2=0}
        when True sync p_2_on do {t2'=0} goto going_on;
```

```
loc going_on: while t2<=5 wait {dp2=0}
        when t2=5  do {t2'=t2'} goto on;

loc on: while True wait {dp2=P_rate}
        when True sync p_2_off goto off;
end

--------------------------------------------------

automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain=0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain=1}
        when True sync close_valve goto closed;
end

--------------------------------------------------

automaton steam

synclabs: start ;
initially idle  ;

loc idle: while True wait {dsteam=0}
          when True sync start goto running;

loc running: while True wait {dsteam in [W1,W2]}

end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do

synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on, p_1_off, p_2_on, p_2_off,
        open_valve, close_valve;

initially idle & t=0;
```

```
-- the initialization mode

loc idle: while True wait {}
        when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
        when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
        when w>=N_2 sync open_valve goto wait_till_drained;
        when w<=N_1 sync p_1_on goto wait_till_fill;
        when N_1<=w & w<=N_2 sync start  do {t'=delta} goto off_off;

loc wait_till_drained: while w>=N_2 wait {}
        when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
        when asap  sync start  do {t'=delta} goto off_off;

loc wait_till_fill: while w<=N_1 wait {}
        when w=N_1 sync start do {t'=delta} goto on_off;

-- the normal operating mode

loc off_off: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l
                do {t'=0} sync p_1_on
                goto going_on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_1_on
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_1_on
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime do {t'=0}
                goto off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto off_off;

loc on_off: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l
                do {t'=0} sync p_2_on
                goto on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0}
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0}
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime
                do {t'=0} sync p_1_off
                goto off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_off;


loc on_on: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l do {t'=0}
                goto on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime
                do {t'=0} sync p_1_off
                goto going_off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_on;


loc going_on_on: while True wait {}
        when asap sync p_2_on goto on_on;


loc going_off_off: while True wait {}
        when asap sync p_2_off goto off_off;


-- emergency stop mode

loc emergency_stop: while t<=delta  wait {}
end


--------------------
-- analysis commands

var
   init_reg, final_reg, reached, reached_final: region;

init_reg :=       loc[steam_boiler]=s0
                & param_l<=w & w<=param_u
                & loc[pump_cont_1]=off
                & loc[pump_cont_2]=off
                & loc[valve]=closed
                & loc[steam]=idle
                & loc[controller]=idle
```

```
                  & MIN<=param_l_prime & param_l_prime<=param_l
                  & param_l<=N_1
                  & N_2<=param_u
                  & param_u<=param_u_prime & param_u_prime<=MAX
                  & delta>=0
                  ;


final_reg :=  w>=MAX
            | w<=MIN
            | loc[controller]=emergency_stop;

reached := reach forward from init_reg endreach;

reached_final := reached & final_reg;

if empty(reached_final)
    then prints "Water level maintained between bounds MIN and MAX";
    else prints "Water level NOT maintained between bounds MIN and MAX";
         prints "Violating states";
         print reached_final;
         prints "End of reached and final";
         print trace to final_reg using reached;
endif;
```

A.2 Parametric analysis of the achievable water levels of the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S

```
---------------------------
-- HyTech input file
--
-- Steam boiler
--
-- Parametric analysis of achievable water levels

define(P_rate,4)
define(W1,0) -- min steam rate
define(W2,6) -- max steam rate

define(MIN,5)
define(param_l_prime,25)
define(param_l,70)
define(N_1,100)
define(N_2,150)
define(param_u,170)
define(param_u_prime,200)
define(MAX,220)
define(delta,5)

var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        p2,     -- pump volume from Pump 2 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        t,      -- controller's clock
        t1,     -- pump controller 1's clock
        t2      -- pump controller 2's clock
          : clock;
        min_param,
        max_param: parameter;

--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw = dp1 + dp2 - dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

synclabs: p_1_on, p_1_off;
initially off ;

loc off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto going_on;

loc going_on: while t1<=delta wait {dp1=0}
        when t1=delta do {t1'=t1'} goto on;

loc on: while True wait {dp1=P_rate}
        when True sync p_1_off goto off;
end


--------------------------------------------------

automaton pump_cont_2

synclabs: p_2_on, p_2_off;
```

```
initially off ;

loc off: while True wait {dp2=0}
        when True sync p_2_on do {t2'=0} goto going_on;

loc going_on: while t2<=delta wait {dp2=0}
        when t2=delta  do {t2'=t2'} goto on;

loc on: while True wait {dp2=P_rate}
        when True sync p_2_off goto off;
end

--------------------------------------------------

automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain=0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain=1}
        when True sync close_valve goto closed;
end

--------------------------------------------------

automaton steam

synclabs: start ;
initially idle  ;

loc idle: while True wait {dsteam=0}
          when True sync start goto running;

loc running: while True wait {dsteam in [W1,W2]}

end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do

synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on, p_1_off, p_2_on, p_2_off,
        open_valve, close_valve;

initially idle & t=0;

-- the initialization mode

loc idle: while True wait {}
        when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
        when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
        when w>=N_2 sync open_valve goto wait_till_drained;
        when w<=N_1 sync p_1_on goto wait_till_fill;
        when N_1<=w & w<=N_2 sync start  do {t'=delta} goto off_off;

loc wait_till_drained: while w>=N_2 wait {}
        when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
        when asap  sync start  do {t'=delta} goto off_off;

loc wait_till_fill: while w<=N_1 wait {}
        when w=N_1 sync start do {t'=delta} goto on_off;

-- the normal operating mode

loc off_off: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l
                do {t'=0} sync p_1_on
                goto going_on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_1_on
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_1_on
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime do {t'=0}
                goto off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto off_off;

loc on_off: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l
                do {t'=0} sync p_2_on
                goto on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0}
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0}
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime
                do {t'=0} sync p_1_off
                goto off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_off;


loc on_on: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l do {t'=0}
                goto on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime
                do {t'=0} sync p_1_off
                goto going_off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_on;


loc going_on_on: while True wait {}
        when asap sync p_2_on goto on_on;


loc going_off_off: while True wait {}
        when asap sync p_2_off goto off_off;


-- emergency stop mode

loc emergency_stop: while t<=delta  wait {}
end


--------------------
-- analysis commands

var
   init_reg, final_reg_param, reached, reached_final: region;

init_reg :=       loc[steam_boiler]=s0
                & param_l<=w & w<=param_u
                & loc[pump_cont_1]=off
                & loc[pump_cont_2]=off
                & loc[valve]=closed
                & loc[steam]=idle
                & loc[controller]=idle
                & MIN<=param_l_prime & param_l_prime<=param_l
                & param_l<=N_1
                & N_2<=param_u
                & param_u<=param_u_prime & param_u_prime<=MAX
                & delta>=0;


final_reg_param :=   w>=max_param
                   | w<=min_param
                  | loc[controller]=emergency_stop;

reached := reach forward from init_reg endreach;

prints "Parametric constraints on min and max water levels";
print omit all locations
        hide non_parameters in reached & final_reg_param endhide;
```

A.3 Parametric analysis of the sampling time for the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S

```
----------------------------
-- HyTech input file
--
-- Steam boiler
--
-- Parametric analysis of sampling time delta

define(P_rate,4)
define(W1,0) -- min steam rate
define(W2,6) -- max steam rate

define(MIN,5)
define(param_l_prime,25)
define(param_l,70)
define(N_1,100)
define(N_2,150)
define(param_u,170)
define(param_u_prime,200)
define(MAX,220)

var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        p2,     -- pump volume from Pump 2 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        t,      -- controller's clock
        t1,     -- pump controller 1's clock
        t2      -- pump controller 2's clock
          : clock;
        delta
                : parameter;

--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw=dp1 + dp2 - dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

synclabs: p_1_on, p_1_off;
initially off ;

loc off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto going_on;

loc going_on: while t1<=5 wait {dp1=0}
        when t1=5 do {t1'=t1'} goto on;

loc on: while True wait {dp1=P_rate}
        when True sync p_1_off goto off;
end


--------------------------------------------------

automaton pump_cont_2

synclabs: p_2_on, p_2_off;
initially off ;

loc off: while True wait {dp2=0}
        when True sync p_2_on do {t2'=0} goto going_on;

loc going_on: while t2<=5 wait {dp2=0}
```

```
        when t2=5  do {t2'=t2'} goto on;

loc on: while True wait {dp2=P_rate}
        when True sync p_2_off goto off;
end

--------------------------------------------------

automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain=0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain=1}
        when True sync close_valve goto closed;
end

--------------------------------------------------

automaton steam

synclabs: start ;
initially idle  ;

loc idle: while True wait {dsteam=0}
          when True sync start goto running;

loc running: while True wait {dsteam in [W1,W2]}

end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do

synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on, p_1_off, p_2_on, p_2_off,
        open_valve, close_valve;

initially idle & t=0;

-- the initialization mode
```

```
loc idle: while True wait {}
       when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
       when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
       when w>=N_2 sync open_valve goto wait_till_drained;
       when w<=N_1 sync p_1_on goto wait_till_fill;
       when N_1<=w & w<=N_2 sync start  do {t'=delta} goto off_off;

loc wait_till_drained: while w>=N_2 wait {}
       when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
       when asap  sync start  do {t'=delta} goto off_off;

loc wait_till_fill: while w<=N_1 wait {}
       when w=N_1 sync start do {t'=delta} goto on_off;

-- the normal operating mode

loc off_off: while t<=delta wait {}
       when t=delta & w<=param_l_prime goto emergency_stop;
       when t=delta & w>=param_u_prime goto emergency_stop;
       when t=delta & param_l_prime<=w & w<=param_l
               do {t'=0} sync p_1_on
               goto going_on_on;
       when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_1_on
               goto on_off;
       when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_1_on
               goto on_off;
       when t=delta & param_u<=w & w<=param_u_prime do {t'=0}
               goto off_off;

       when t=delta & N_1<=w & w<=N_2 do {t'=0} goto off_off;

loc on_off: while t<=delta wait {}
       when t=delta & w<=param_l_prime goto emergency_stop;
       when t=delta & w>=param_u_prime goto emergency_stop;
       when t=delta & param_l_prime<=w & w<=param_l
               do {t'=0} sync p_2_on
               goto on_on;
       when t=delta & param_l<=w & w<=N_1 do {t'=0}
               goto on_off;
       when t=delta & N_2<=w & w<=param_u do {t'=0}
               goto on_off;
       when t=delta & param_u<=w & w<=param_u_prime
               do {t'=0} sync p_1_off
```

```
                goto off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_off;


loc on_on: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l do {t'=0}
                goto on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime
                do {t'=0} sync p_1_off
                goto going_off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_on;


loc going_on_on: while True wait {}
        when asap sync p_2_on goto on_on;


loc going_off_off: while True wait {}
        when asap sync p_2_off goto off_off;


-- emergency stop mode

loc emergency_stop: while t<=delta  wait {}

end


--------------------
-- analysis commands

var
   init_reg, final_reg, reached, reached_final: region;

init_reg :=       loc[steam_boiler]=s0
                & param_l<=w & w<=param_u
                & loc[pump_cont_1]=off
                & loc[pump_cont_2]=off
                & loc[valve]=closed
                & loc[steam]=idle
                & loc[controller]=idle
                & MIN<=param_l_prime & param_l_prime<=param_l
                & param_l<=N_1
                & N_2<=param_u
                & param_u<=param_u_prime & param_u_prime<=MAX
                & delta>=5
                ;


final_reg :=  w>=MAX
            | w<=MIN
            | loc[controller]=emergency_stop;

reached := reach forward from init_reg endreach;
reached_final := reached & final_reg;

if empty(reached_final)
    then prints "Water level maintained between bounds MIN and MAX";
         prints " for all values of delta";
    else prints "Violating values for delta";
         print omit all locations
             hide non_parameters in reached_final endhide;
endif;
```

A.4 Parametric analysis of the control parameters for the fault-free two-pump steam-boiler system with controller C_A and steam model A_A^S

```
---------------------------
-- HyTech input file
--
-- Steam boiler
--
-- Parametric synthesis of controller cut-off values


define(P_rate,4)
define(W1,0) -- min steam rate
define(W2,6) -- max steam rate

define(MIN,5)
define(N_1,100)
define(N_2,150)
define(MAX,220)

define(delta,5)

var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        p2,     -- pump volume from Pump 2 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        t,      -- controller's clock
        t1,     -- pump controller 1's clock
        t2      -- pump controller 2's clock
          : clock;
        param_l,       -- parameter L
        param_l_prime, -- parameter L'
        param_u,       -- parameter U
        param_u_prime  -- parameter U'
          : parameter;


--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw = dp1 + dp2 -dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

synclabs: p_1_on, p_1_off;
initially off ;

loc off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto going_on;

loc going_on: while t1<=5 wait {dp1=0}
        when t1=5 do {t1'=t1'} goto on;

loc on: while True wait {dp1=P_rate}
        when True sync p_1_off goto off;
end


--------------------------------------------------

automaton pump_cont_2

synclabs: p_2_on, p_2_off;
initially off ;
```

```
loc off: while True wait {dp2=0}
        when True sync p_2_on do {t2'=0} goto going_on;

loc going_on: while t2<=5 wait {dp2=0}
        when t2=5  do {t2'=t2'} goto on;

loc on: while True wait {dp2=P_rate}
        when True sync p_2_off goto off;
end

--------------------------------------------------

automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain=0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain=1}
        when True sync close_valve goto closed;
end

--------------------------------------------------

automaton steam

synclabs: start ;
initially idle  ;

loc idle: while True wait {dsteam=0}
          when True sync start goto running;

loc running: while True wait {dsteam in [W1,W2]}

end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do

synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on, p_1_off, p_2_on, p_2_off,
        open_valve, close_valve;
```

```
initially idle & t=0;

-- the initialization mode

loc idle: while True wait {}
        when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
        when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
        when w>=N_2 sync open_valve goto wait_till_drained;
        when w<=N_1 sync p_1_on goto wait_till_fill;
        when N_1<=w & w<=N_2 sync start  do {t'=delta} goto off_off;

loc wait_till_drained: while w>=N_2 wait {}
        when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
        when asap  sync start  do {t'=delta} goto off_off;

loc wait_till_fill: while w<=N_1 wait {}
        when w=N_1 sync start do {t'=delta} goto on_off;

-- the normal operating mode

loc off_off: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l
                do {t'=0} sync p_1_on
                goto going_on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_1_on
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_1_on
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime do {t'=0}
                goto off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto off_off;

loc on_off: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l
                do {t'=0} sync p_2_on
                goto on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0}
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0}
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime
                do {t'=0} sync p_1_off
                goto off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_off;


loc on_on: while t<=delta wait {}
        when t=delta & w<=param_l_prime goto emergency_stop;
        when t=delta & w>=param_u_prime goto emergency_stop;
        when t=delta & param_l_prime<=w & w<=param_l do {t'=0}
                goto on_on;
        when t=delta & param_l<=w & w<=N_1 do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & N_2<=w & w<=param_u do {t'=0} sync p_2_off
                goto on_off;
        when t=delta & param_u<=w & w<=param_u_prime
                do {t'=0} sync p_1_off
                goto going_off_off;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto on_on;


loc going_on_on: while True wait {}
        when asap sync p_2_on goto on_on;


loc going_off_off: while True wait {}
        when asap sync p_2_off goto off_off;


-- emergency stop mode

loc emergency_stop: while t<=delta  wait {}
end


--------------------
-- analysis commands

var
   init_reg, final_reg, reached, reached_final: region;

init_reg :=       loc[steam_boiler]=s0
                & param_l<=w & w<=param_u
                & loc[pump_cont_1]=off
                & loc[pump_cont_2]=off
                & loc[valve]=closed
```

```
                  & loc[steam]=idle
                  & loc[controller]=idle
                  & MIN<=param_l_prime & param_l_prime<=param_l
                  & param_l<=N_1
                  & N_2<=param_u
                  & param_u<=param_u_prime & param_u_prime<=MAX;


final_reg :=   w>=MAX
             | w<=MIN
             | loc[controller]=emergency_stop;

reached := reach forward from init_reg endreach;

reached_final := reached & final_reg;

prints "Safety requirements maintained unless";
print omit all locations
   hide non_parameters in reached_final endhide;
```

A.5 Verification of the fault-free four-pump steam-boiler system with steam model A_A^S

```
---------------------------
-- HyTech input file
--
-- Steam boiler
--
-- using 4 pumps
-- abort below param_l_4
-- activate
--      4 between param_l_4 and param_l_3
--      3 between param_l_3 and param_l_2
--      two between param_l_2 and param_l_1,
--      one between param_l_1 and M
--      none between M and N
--
--      one on between N and c_u_1
--      none between c_u_1 and c_u_0
--
--  always turn on, going up in sequence, turn off going down in sequence
--

define(P_rate,2)
define(W1,0) -- min steam rate
define(W2,6) -- max steam rate
define(param_l_4,25)
define(param_l_3,40)
define(param_l_2,60)
define(param_l_1,80)
define(param_u_1,170)
define(param_u_0,200)


define(MIN,5)
define(N_1,100)
define(N_2,150)
define(MAX,220)

define(delta,5)

var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        p2,     -- pump volume from Pump 2 for a time slot
        p3,     -- pump volume from Pump 3 for a time slot
        p4,     -- pump volume from Pump 3 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        t,      -- controller's clock
        t1,     -- pump controller 1's clock
        t2,     -- pump controller 2's clock
        t3,     -- pump controller 3's clock
        t4      -- pump controller 4's clock
          : clock;

--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw = dp1 + dp2 + dp3 + dp4 - dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

synclabs: p_1_on, p_1_off;
initially off ;
```

```
loc off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto going_on;

loc going_on: while t1<=5 wait {dp1=0}
        when t1=5 do {t1'=t1'} goto on;

loc on: while True wait {dp1=P_rate}
        when True sync p_1_off goto off;
end


-------------------------------------------------

automaton pump_cont_2

synclabs: p_2_on, p_2_off;
initially off ;

loc off: while True wait {dp2=0}
        when True sync p_2_on do {t2'=0} goto going_on;

loc going_on: while t2<=5 wait {dp2=0}
        when t2=5  do {t2'=t2'} goto on;

loc on: while True wait {dp2=P_rate}
        when True sync p_2_off goto off;
end

-------------------------------------------------

automaton pump_cont_3

synclabs: p_3_on, p_3_off;
initially off ;

loc off: while True wait {dp3=0}
        when True sync p_3_on do {t3'=0} goto going_on;

loc going_on: while t3<=5 wait {dp3=0}
        when t3=5  do {t3'=t3'} goto on;

loc on: while True wait {dp3=P_rate}
        when True sync p_3_off goto off;
end
-------------------------------------------------

automaton pump_cont_4

synclabs: p_4_on, p_4_off;
initially off ;
```

```
loc off: while True wait {dp4=0}
        when True sync p_4_on do {t4'=0} goto going_on;

loc going_on: while t4<=5 wait {dp4=0}
        when t4=5  do {t4'=t4'} goto on;

loc on: while True wait {dp4=P_rate}
        when True sync p_4_off goto off;
end

--------------------------------------------------

automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain=0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain=1}
        when True sync close_valve goto closed;
end

--------------------------------------------------

automaton steam

synclabs: start ;
initially idle;

loc idle: while True wait {dsteam=0}
        when True sync start goto running;

loc running: while True wait {dsteam in [W1,W2]}

end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do

synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on, p_1_off,
        p_2_on, p_2_off,
        p_3_on, p_3_off,
        p_4_on, p_4_off,
        open_valve, close_valve;

initially idle & t=0;

-- the initialization mode

loc idle: while True wait {}
        when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
        when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
        when w>=N_2 sync open_valve goto wait_till_drained;
        when w<=N_1 sync p_1_on goto wait_till_fill;
        when N_1<=w & w<=N_2 sync start  do {t'=delta} goto op_0000;

loc wait_till_drained: while w>=N_2 wait {}
        when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
        when asap  sync start  do {t'=delta} goto op_0000;

loc wait_till_fill: while w<=N_1 wait {}
        when w=N_1 sync start do {t'=delta} goto op_1000;

-- the normal operating mode

loc op_0000: while t<=delta wait {}
        when t=delta & w<=param_l_4 goto emergency_stop;
        when t=delta & w>=param_u_0 goto emergency_stop;
        when t=delta & param_l_4<=w & w<=param_l_3
                do {t'=0} sync p_1_on
                goto going_1111_2;
        when t=delta & param_l_3<=w & w<=param_l_2
                do {t'=0} sync p_1_on
                goto going_1110_2;
        when t=delta & param_l_2<=w & w<=param_l_1
                do {t'=0} sync p_1_on
                goto going_1100_2;
        when t=delta & param_l_1<=w & w<=N_1
                do {t'=0} sync p_1_on
                goto op_1000;
        when t=delta & N_2<=w & w<=param_u_1
                do {t'=0} sync p_1_on
                goto op_1000;
        when t=delta & param_u_1<=w & w<=param_u_0 do {t'=0}
```

```
                goto op_0000;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto op_0000;

loc op_1000: while t<=delta wait {}
        when t=delta & w<=param_l_4 goto emergency_stop;
        when t=delta & w>=param_u_0 goto emergency_stop;
        when t=delta & param_l_4 <=w & w<=param_l_3
                do {t'=0} sync p_2_on
                goto going_1111_3;
        when t=delta & param_l_3 <=w & w<=param_l_2
                do {t'=0} sync p_2_on
                goto going_1110_3;
        when t=delta & param_l_2 <=w & w<=param_l_1
                do {t'=0} sync p_2_on
                goto op_1100;
        when t=delta & param_l_1<=w & w<=N_1 do {t'=0}
                goto op_1000;
        when t=delta & N_2<=w & w<=param_u_1 do {t'=0}
                goto op_1000;
        when t=delta & param_u_1<=w & w<=param_u_0
                do {t'=0} sync p_1_off
                goto op_0000;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto op_1000;


loc op_1100: while t<=delta wait {}
        when t=delta & w<=param_l_4  goto emergency_stop;
        when t=delta & w>=param_u_0 goto emergency_stop;
        when t=delta & param_l_4 <=w & w<=param_l_3
                do {t'=0} sync p_3_on
                goto going_1111_4;
        when t=delta & param_l_3 <=w & w<=param_l_2
                do {t'=0} sync p_3_on
                goto op_1110;
        when t=delta & param_l_2 <=w & w<=param_l_1
                do {t'=0}
                goto op_1100;
        when t=delta & param_l_1<=w & w<=N_1
                do {t'=0} sync p_2_off
                goto op_1000;
        when t=delta & N_2<=w & w<=param_u_1
                do {t'=0} sync p_2_off
                goto op_1000;
        when t=delta & param_u_1<=w & w<=param_u_0
                do {t'=0}  sync p_2_off
                goto going_0000_1;

        when t=delta & N_1<=w & w<=N_2
```

```
                do {t'=0} goto op_1100;


loc op_1110: while t<=delta wait {}
        when t=delta & w<=param_l_4  goto emergency_stop;
        when t=delta & w>=param_u_0 goto emergency_stop;
        when t=delta & param_l_4 <=w & w<=param_l_3
                do {t'=0} sync p_4_on
                goto op_1111;
        when t=delta & param_l_3 <=w & w<=param_l_2
                do {t'=0}
                goto op_1110;
        when t=delta & param_l_2 <=w & w<=param_l_1
                do {t'=0} sync p_3_off
                goto op_1100;
        when t=delta & param_l_1<=w & w<=N_1
                do {t'=0} sync p_3_off
                goto going_1000_2;
        when t=delta & N_2<=w & w<=param_u_1
                do {t'=0} sync p_3_off
                goto going_1000_2;
        when t=delta & param_u_1<=w & w<=param_u_0
                do {t'=0}  sync p_3_off
                goto going_0000_2;

        when t=delta & N_1<=w & w<=N_2 do {t'=0} goto op_1110;


loc op_1111: while t<=delta wait {}
        when t=delta & w<=param_l_4  goto emergency_stop;
        when t=delta & w>=param_u_0 goto emergency_stop;
        when t=delta & param_l_4 <=w & w<=param_l_3
                do {t'=0}
                goto op_1111;
        when t=delta & param_l_3 <=w & w<=param_l_2
                do {t'=0} sync p_4_off
                goto op_1110;
        when t=delta & param_l_2 <=w & w<=param_l_1
                do {t'=0} sync p_4_off
                goto going_1100_3;
        when t=delta & param_l_1<=w & w<=N_1
                do {t'=0} sync p_4_off
                goto going_1000_3;
        when t=delta & N_2<=w & w<=param_u_1
                do {t'=0} sync p_4_off
                goto going_1000_3;
        when t=delta & param_u_1<=w & w<=param_u_0
                do {t'=0}  sync p_4_off
                goto going_0000_3;
        when t=delta & N_1<=w & w<=N_2
                do {t'=0} goto op_1111;


-- intermediate states

loc going_1111_2: while True wait {}
        when asap sync p_2_on goto going_1111_3;

loc going_1111_3: while True wait {}
        when asap sync p_3_on goto going_1111_4;


loc going_1111_4: while True wait {}
        when asap sync p_4_on goto op_1111;

loc going_1110_2: while True wait {}
        when asap sync p_2_on goto going_1110_3;

loc going_1110_3: while True wait {}
        when asap sync p_3_on goto op_1110;


loc going_1100_2: while True wait {}
        when asap sync p_2_on goto op_1100;


-- turning off

loc going_1100_3: while True wait {}
        when asap sync p_3_off goto op_1100;

loc going_1000_3: while True wait {}
        when asap sync p_3_off goto going_1000_2;

loc going_1000_2: while True wait {}
        when asap sync p_2_off goto op_1000;

loc going_0000_3: while True wait {}
        when asap sync p_3_off goto going_0000_2;

loc going_0000_2: while True wait {}
        when asap sync p_2_off goto going_0000_1;

loc going_0000_1: while True wait {}
        when asap sync p_1_off goto op_0000;


-- emergency stop mode
```

```
loc emergency_stop: while t<=delta  wait {}
end


--------------------
-- analysis commands

var
        init_reg, final_reg, reached, reached_final: region;

init_reg :=       loc[steam_boiler]=s0
                & param_l_1<=w & w<=param_u_1
                & loc[pump_cont_1]=off
                & loc[pump_cont_2]=off
                & loc[valve]=closed
                & loc[steam]=idle
                & loc[controller]=idle
                & MIN<=param_l_3 & param_l_3<=param_l_1
                & param_l_1<=N_1
                & N_2<=param_u_1
                & param_u_1<=param_u_0 & param_u_0<=MAX;


final_reg :=  w>=MAX
            | w<=MIN
            | loc[controller]=emergency_stop;

reached := reach forward from init_reg endreach;

reached_final := reached & final_reg;

if empty(reached_final)
  then prints "Water level maintained between bounds MIN and MAX";
else prints "Water level NOT maintained between bounds MIN and MAX";
     prints "Violating states";
     print reached_final;
     prints "End of reached and final";
     print trace to final_reg using reached;
endif;
```

A.6 Verification of the fault-tolerant two-pump steam-boiler system with steam model A_A^S

```
---------------------------
-- HyTech input file
--
```

```
-- Steam boiler
--
-- fault-tolerant version.
-- pumps can break at any time, except when in off mode
-- broken pumps use rate [0,P_rate]
-- repair them and reinstate in off mode
-- controller uses local variable(s) to record status of pumps
--   from each location, at end of time, it performs interchange of info
--   with each pump, and updates its local variables

-- uses constants, lo_abort =          50
--                 param_1 =           80
--                 N_1 =              100
--                 N_2 =              150
--                 hi_abort_none_off = 185
--                 hi_abort_one_off =  215
--                 hi_abort_both_off = 245

-- when both pumps OK
--      use: in range (-inf, lo_abort] -- abort
--           [lo_abort, param_1] --      both on
--           [param_1, N_1] --           one on
-- when one OK, one broken.
--      use: in range (-inf, lo_abort] -- abort
--           [lo_abort,100] --           one on.
-- when both broken,
--      use: (-inf, lo_abort] --         abort
--           [lo_abort = 50,100] --      wait

-- in range: [N_1,N_2] --                no action

-- with both pump known to be off
--      use: in range [N_2,hi_abort_both_off] -- no action
--           [hi_abort_both_off,inf) --         abort
-- with one pump known to be off
--      use: in range [N_2,hi_abort_one_off] --  no action
--           [hi_one_off,inf_abort) --          abort
-- with no pumps known to be off
--      use: in range [N_2,hi_abort_none_off] -- no action
--           [hi_abort_none_off,inf) --         abort



-- controller mode names off_on, etc, indicate the expected state
--      of each pump at the end of the coming cycle.  when a pump
--      breaks we assume it is off since, once it is fixed, it is
--      in off state.


define(P_rate,3)
define(W1,0) -- min steam rate
define(W2,4) -- max steam rate


define(MIN,5)
define(lo_abort,50)
define(param_1,80)
define(N_1,100)
define(N_2,150)
define(hi_abort_both_off,245)
define(hi_abort_one_off,215)
define(hi_abort_none_off,185)
define(MAX,250)

define(delta,5)

var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        p2,     -- pump volume from Pump 2 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        t,      -- controller's clock
        t1,     -- pump controller 1's clock
        t2      -- pump controller 2's clock
          : clock;
        OK_1,
        OK_2
          : discrete;

--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw = dp1 + dp2 - dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

-- pump can spontaneously break from any OK mode.
-- broken pump outputs at any rate between 0 and P_rate
-- it is detected at next cycle of communication.
-- nondeterministic time to repair
```

```
synclabs:
        p_1_on,          -- command from controller to turn on
        p_1_off,         -- command from controller to turn off
        p_1_OK,          -- tell controller pump is OK
        p_1_broken,      -- tell controller pump is broken
        p_1_repairing    -- tell controller pump is being repaired
        ;

initially OK_off ;

loc OK_off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto OK_going_on;
        when True sync p_1_OK goto OK_off;

loc OK_going_on: while t1<=5 wait {dp1=0}
        when t1=5 do {t1'=t1'} goto OK_on;
        when True sync p_1_OK goto OK_going_on;
        when True sync p_1_off goto OK_off;
        when True do {t1'=t1'} goto broken;

loc OK_on: while True wait {dp1=P_rate}
        when True sync p_1_off goto OK_off;
        when True sync p_1_OK goto OK_on;
        when True do {t1'=t1'} goto broken;


-- broken mode
loc broken: while True wait {dp1 in [0,P_rate]}
        -- p_1_broken can be used to model communication to controller
        --  and the controller's command to repair the pump
        when True sync p_1_broken goto repair;
        when True sync p_1_on goto broken;
        when True sync p_1_off goto broken;

-- repairing mode
loc repair: while True wait {dp1 in [0,P_rate]}
        when True sync p_1_repairing goto repair;
        when True goto OK_off;

        when True sync p_1_on goto repair;
        when True sync p_1_off goto repair;

end


--------------------------------------------------
automaton pump_cont_2

-- pump can spontaneously break from any OK mode.
-- broken pump outputs at any rate between 0 and P_rate
-- it is detected at next cycle of communication.
-- nondeterministic time to repair

synclabs:
        p_2_on,         -- command from controller to turn on
        p_2_off,        -- command from controller to turn off
        p_2_OK,         -- tell controller pump is OK
        p_2_broken,     -- tell controller pump is broken
        p_2_repairing   -- tell controller pump is being repaired
        ;

initially OK_off ;

loc OK_off: while True wait {dp2=0}
        when True sync p_2_on do {t2'=0} goto OK_going_on;
        when True sync p_2_OK goto OK_off;


loc OK_going_on: while t2<=5 wait {dp2=0}
        when t2=5 do {t2'=t2'} goto OK_on;
        when True sync p_2_OK goto OK_going_on;
        when True do {t2'=t2'} goto broken;
        when True sync p_2_off goto OK_off;

loc OK_on: while True wait {dp2=P_rate}
        when True sync p_2_off goto OK_off;
        when True sync p_2_OK goto OK_on;
        when True do {t2'=t2'} goto broken;


-- broken mode
loc broken: while True wait {dp2 in [0,P_rate]}
        when True sync p_2_broken goto repair;

        when True sync p_2_on goto broken;
        when True sync p_2_off goto broken;

-- repairing mode
loc repair: while True wait {dp2 in [0,P_rate]}
        when True sync p_2_repairing goto repair;
        when True goto OK_off;

        when True sync p_2_on goto repair;
        when True sync p_2_off goto repair;

end

--------------------------------------------------
```

```
automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain=0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain=1}
        when True sync close_valve goto closed;
end

--------------------------------------------------

automaton steam
synclabs: start ;
initially idle;

loc idle: while True wait {dsteam=0}
        when True sync start
                goto running;

loc running: while True wait {dsteam in [W1,W2]}
end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do
-- some redundancy in off_on/on_off modes, since the normal range
-- is adjacent to other regions where no action is taken


synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on, p_1_off,
        p_1_OK, p_1_broken, p_1_repairing,
        p_2_on, p_2_off,
        p_2_OK, p_2_broken, p_2_repairing,
        open_valve, close_valve;

initially idle & t=0;

-- the initialization mode

loc idle: while True wait {}
```

```
        when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
        when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
        when w>=N_2 sync open_valve goto wait_till_drained;
        when w<=N_1 sync p_1_on goto wait_till_fill;
        when N_1<=w & w<=N_2 sync start  do {t'=delta} goto off_off;

loc wait_till_drained: while w>=N_2 wait {}
        when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
        when asap  sync start  do {t'=delta} goto off_off;

loc wait_till_fill: while w<=N_1 wait {}
        when w=N_1 sync start do {t'=delta} goto on_off;

-- the normal operating mode

loc off_off: while t<=delta wait {}
        -- find status of pumps
        when t=delta sync p_1_OK do {OK_1'=1,t'=t'} goto off_off_b;
        when t=delta sync p_1_broken do {OK_1'=0,t'=t'} goto off_off_b;


loc off_off_b: while True wait {}
        when asap sync p_2_OK do {OK_2'=1, t'=0} goto off_off_d;
        when asap sync p_2_broken do {OK_2'=0,t'=0} goto off_off_d;
        when asap sync p_2_repairing do {t'=0} goto off_off_d;

loc off_off_d: while t=0 wait {}
        -- normal mode
        when OK_1=1 & OK_2=1 &
                w<=lo_abort
                goto emergency_stop;
        when OK_1=1 & OK_2=1 &
                lo_abort<=w & w<=param_1 sync p_1_on
                goto going_on_on;
        when OK_1=1 & OK_2=1 &
                param_1<=w & w<=N_1 sync p_2_on
                goto off_on;
        when OK_1=1 & OK_2=1 &
                param_1<=w & w<=N_1  sync p_1_on
                goto on_off;
        when OK_1=1 & OK_2=1 &
                N_1<=w & w<=N_2 goto off_off;
        when  OK_1=1 & OK_2=1 &
                 N_2<=w & w<=hi_abort_both_off goto off_off;
```

```
when  OK_1=1 & OK_2=1 &
        hi_abort_both_off<=w
        goto emergency_stop;

-- one broken, OK_vector=10
when OK_1=1 & OK_2=0 &
        w<=lo_abort
        goto emergency_stop;
when OK_1=1 & OK_2=0 &
        lo_abort<=w & w<=N_1 sync p_1_on
        goto on_off;
when OK_1=1 & OK_2=0 &
        N_1<=w & w<=N_2
        goto off_off;
when OK_1=1 & OK_2=0 &
        N_2<=w & w<=hi_abort_one_off
        goto off_off;
when OK_1=1 & OK_2=0 &
        hi_abort_one_off<=w
        goto emergency_stop;

-- one broken, OK_vector= 01
when OK_1=0 & OK_2=1 &
        w<=lo_abort
        goto emergency_stop;
when OK_1=0 & OK_2 =1 &
        lo_abort<=w & w<=N_1 sync p_2_on
        goto on_off;
when OK_1=0 & OK_2=1 &
        N_1<=w & w<=N_2
        goto off_off;
when OK_1 =0  & OK_2=1 &
        N_2<=w & w<=hi_abort_one_off
        goto off_off;
when OK_1=0 & OK_2=1 &
        hi_abort_one_off<=w
        goto emergency_stop;


-- both broken, OK_vector=00
when OK_1=0 & OK_2=0 &
        w<=lo_abort
        goto emergency_stop;
when OK_1=0 & OK_2=0 &
        lo_abort<=w & w<=N_1
        goto off_off;
when OK_1=0 & OK_2=0 &
        N_1<=w & w<=N_2
        goto off_off;
when OK_1 =0  & OK_2=0 &
```

```
                N_2<=w & w<=hi_abort_none_off
                goto off_off;
        when OK_1=0 & OK_2=0 &
                hi_abort_none_off<=w
                goto emergency_stop;


loc off_on: while t<=delta wait {}
        -- find  status of pumps
        when t=delta sync p_1_OK do {OK_1'=1, t'=t'} goto off_on_b;
        when t=delta sync p_1_broken do {OK_1'=0,t'=t'} goto off_on_b;
        when t=delta sync p_1_repairing do {t'=t'} goto off_on_b;

loc off_on_b: while True wait {}
        when asap sync p_2_OK do {OK_2'=1, t'=0} goto off_on_d;
        when asap sync p_2_broken do {OK_2'=0,t'=0} goto off_on_d;
        when asap sync p_2_repairing do {t'=0} goto off_on_d;

loc off_on_d: while t=0 wait {}
        -- normal mode
        when  OK_1=1 & OK_2=1 &
                w<=lo_abort
                goto emergency_stop;
        when  OK_1=1 & OK_2=1 &
                lo_abort<=w & w<=param_1  sync p_1_on
                goto on_on;
        when  OK_1=1 & OK_2=1 &
                param_1<=w & w<=N_1
                goto off_on;
        when  OK_1=1 & OK_2=1 &
                 N_1<=w & w<=N_2  goto off_on;
        when  OK_1=1 & OK_2=1 &
                N_2<=w & w<=hi_abort_one_off sync p_2_off
                goto off_off;
        when  OK_1=1 & OK_2=1 &
                hi_abort_one_off<=w
                goto emergency_stop;

        -- one broken, OK_vector=10
        when OK_1=1 & OK_2=0 &
                w<=lo_abort
                goto emergency_stop;
        when OK_1=1 & OK_2=0 &
                lo_abort<=w & w<=N_1 sync p_1_on
                goto on_off;
        when OK_1=1 & OK_2=0 &
                N_1<=w & w<=N_2
                goto off_off;
        when OK_1=1 & OK_2=0 &
                N_2<=w & w<=hi_abort_one_off
                goto off_off;
        when OK_1=1 & OK_2=0 &
                hi_abort_one_off<=w
                goto emergency_stop;

        -- one broken, OK_vector=01
        when OK_1=0 & OK_2=1 &
                w<=lo_abort
                goto emergency_stop;
        when OK_1=0 & OK_2=1 &
                lo_abort<=w & w<=N_1 sync p_2_on
                goto off_on;
        when OK_1=0 & OK_2=1 &
                N_1<=w & w<=N_2
                goto off_on;
        when OK_1 =0  & OK_2=1 &
                N_2<=w & w<=hi_abort_none_off sync p_2_off
                goto off_off;
        when OK_1=0 & OK_2=1 &
                hi_abort_none_off<=w
                goto emergency_stop;


        -- both broken, OK_vector=00
        when OK_1=0 & OK_2=0 &
                w<=lo_abort
                goto emergency_stop;
        when OK_1=0 & OK_2=0 &
                lo_abort<=w & w<=N_1
                goto off_off;
        when OK_1=0 & OK_2=0 &
                N_1<=w & w<=N_2
                goto off_off;
        when OK_1 =0  & OK_2=0 &
                N_2<=w & w<=hi_abort_none_off
                goto off_off;
        when OK_1=0 & OK_2=0 &
                hi_abort_none_off<=w
                goto emergency_stop;


loc on_off: while t<=delta wait {}
        -- find  status of pumps
        when t=delta sync p_1_OK do {OK_1'=1,t'=t'} goto on_off_b;
        when t=delta sync p_1_broken do {OK_1'=0,t'=t'} goto on_off_b;
        when t=delta sync p_1_repairing do {t'=t'} goto on_off_b;

loc on_off_b: while True wait {}
        when asap sync p_2_OK do {OK_2'=1,t'=0} goto on_off_d;
        when asap sync p_2_broken do {OK_2'=0,t'=0} goto on_off_d;
        when asap sync p_2_repairing do {t'=0} goto on_off_d;

loc on_off_d: while t=0 wait {}
        -- normal mode
        when  OK_1=1 & OK_2=1 &
                w<=lo_abort
                goto emergency_stop;
        when  OK_1=1 & OK_2=1 &
                lo_abort <=w & w<=param_1  sync p_2_on
                goto on_on;
        when  OK_1=1 & OK_2=1 &
                param_1<=w & w<=N_1
                goto on_off;
        when   OK_1=1 & OK_2=1 &
                N_1<=w & w<=N_2  goto on_off;
        when   OK_1=1 & OK_2=1 &
                N_2<=w & w<=hi_abort_one_off  sync p_1_off
                goto off_off;
        when   OK_1=1 & OK_2=1 &
                hi_abort_one_off<=w
                goto emergency_stop;


        -- one broken, OK_vector=10
        when OK_1=1 & OK_2=0 &
                w<=lo_abort
                goto emergency_stop;
        when OK_1=1 & OK_2=0 &
                lo_abort<=w & w<=N_1 sync p_1_on
                goto on_off;
        when OK_1=1 & OK_2=0 &
                N_1<=w & w<=N_2
                goto on_off;
        when OK_1=1 & OK_2=0 &
                N_2<=w & w<=hi_abort_none_off sync p_1_off
                goto off_off;
        when OK_1=1 & OK_2=0 &
                hi_abort_none_off<=w
                goto emergency_stop;

        -- one broken, OK_vector=01
        when OK_1=0 & OK_2=1 &
                w<=lo_abort
                goto emergency_stop;
        when OK_1=0 & OK_2=1 &
                lo_abort<=w & w<=N_1 sync p_2_on
                goto off_on;
        when OK_1=0 & OK_2=1 &
                N_1<=w & w<=N_2
                goto off_on;
        when OK_1=0  & OK_2=1 &
                N_2<=w & w<=hi_abort_one_off sync p_1_off
                goto off_off;
        when OK_1=0 & OK_2=1 &
                hi_abort_one_off<=w
                goto emergency_stop;


        -- both broken, OK_vector=00
        when OK_1=0 & OK_2=0 &
                w<=lo_abort
                goto emergency_stop;
        when OK_1=0 & OK_2=0 &
                lo_abort<=w & w<=N_1
                goto off_off;
        when OK_1=0 & OK_2=0 &
                N_1<=w & w<=N_2
                goto off_off;
        when OK_1=0  & OK_2=0 &
                N_2<=w & w<=hi_abort_none_off
                goto off_off;
        when OK_1=0 & OK_2=0 &
                hi_abort_none_off<=w
                goto emergency_stop;



loc on_on: while t<=delta wait {}
        -- find  status of pumps
        when t=delta sync p_1_OK do {OK_1'=1,t'=t'} goto on_on_b;
        when t=delta sync p_1_broken do {OK_1'=0,t'=t'} goto on_on_b;
        when t=delta sync p_1_repairing do {t'=t'} goto on_on_b;

loc on_on_b: while True wait {}
        when asap sync p_2_OK do {OK_2'=1,t'=0} goto on_on_d;
        when asap sync p_2_broken do {OK_2'=0,t'=0} goto on_on_d;
        when asap sync p_2_repairing do {t'=0} goto on_on_d;

loc on_on_d: while t=0 wait {}
        -- normal mode
        when  OK_1=1 & OK_2=1 &
                w<=lo_abort
                goto emergency_stop;
        when  OK_1=1 & OK_2=1 &
                lo_abort  <=w & w<=param_1
                goto on_on;
        when   OK_1=1 & OK_2=1 &
                param_1<=w & w<=N_1  sync p_2_off
                goto on_off;
```

```
when  OK_1=1 & OK_2=1 &
          N_1<=w & w<=N_2  goto on_on;
when  OK_1=1 & OK_2=1 &
          N_2<=w & w<=hi_abort_none_off   sync p_1_off
        goto going_off_off;
when  OK_1=1 & OK_2=1 &
        hi_abort_none_off<=w
        goto emergency_stop;

-- one broken, OK_vector=10
when OK_1=1 & OK_2=0 &
        w<=lo_abort
        goto emergency_stop;
when OK_1=1 & OK_2=0 &
        lo_abort<=w & w<=N_1 sync p_1_on
        goto on_off;
when OK_1=1 & OK_2=0 &
        N_1<=w & w<=N_2
        goto on_off;
when OK_1=1 & OK_2=0 &
        N_2<=w & w<=hi_abort_none_off sync p_1_off
        goto off_off;
when OK_1=1 & OK_2=0 &
        hi_abort_none_off<=w
        goto emergency_stop;

-- one broken, OK_vector=01
when OK_1=0 & OK_2=1 &
        w<=lo_abort
        goto emergency_stop;
when OK_1=0 & OK_2=1 &
        lo_abort<=w & w<=N_1 sync p_2_on
        goto off_on;
when OK_1=0 & OK_2=1 &
        N_1<=w & w<=N_2
        goto off_on;
when OK_1=0  & OK_2=1 &
        N_2<=w & w<=hi_abort_none_off sync p_2_off
        goto off_off;
when OK_1=0 & OK_2=1 &
        hi_abort_none_off<=w
        goto emergency_stop;


-- both broken, OK_vector=00
when OK_1=0 & OK_2=0 &
        w<=lo_abort
        goto emergency_stop;
when OK_1=0 & OK_2=0 &
        lo_abort<=w & w<=N_1
```

```
                goto off_off;
        when OK_1=0 & OK_2=0 &
                N_1<=w & w<=N_2
                goto off_off;
        when OK_1=0  & OK_2=0 &
                N_2<=w & w<=hi_abort_none_off
                goto off_off;
        when OK_1=0 & OK_2=0 &
                hi_abort_none_off<=w
                goto emergency_stop;

loc going_on_on: while True wait {}
        when asap sync p_2_on goto on_on;


loc going_off_off: while True wait {}
        when asap sync p_2_off goto off_off;



-- emergency stop mode

loc emergency_stop: while t<=delta  wait {}
end


--------------------
-- analysis commands

var
        init_reg, final_reg, reached, reached_final,
        emergency_reg, bounds_viol_reg : region;

-- initially t=delta, so that a control decision is made immediately
init_reg :=       loc[steam_boiler]=s0
                & N_1 -20<=w & w<=N_2 + 20
                & loc[pump_cont_1]=OK_off
                & loc[pump_cont_2]=OK_off
                & loc[valve]=closed
                & loc[steam]=idle
                & loc[controller]=idle
                & OK_1=1
                & OK_2=1
                ;


emergency_reg :=  loc[controller]=emergency_stop;

bounds_viol_reg := w>=MAX | w<=MIN;
```

```
reached := reach forward from init_reg endreach;

reached_final := reached & emergency_reg;

if empty(reached_final)
  then prints "Never enters emergency state";
  else prints "Can enter emergency state";
        print omit all locations
                hide p1, p2, steam, drain, t, t1, t2, OK_1, OK_2
                     in reached_final endhide ;
endif;

prints "==================================";

reached_final := reached & bounds_viol_reg;

if empty(reached_final)
    then prints "Water level maintained between bounds MIN and MAX";
    else prints "Water level NOT maintained between bounds MIN and MAX";
         prints "Violating states";
         print trace to final_reg using reached;
endif;
```

A.7 Parametric analysis of the control parameter L in the dual-pump fault-free steam-boiler system with controller C'_A and steam model A^S_A

```
---------------------------
-- HyTech input file
--
-- Steam boiler
--
--
-- time slot is 5 units long


define(P_rate,4)
define(W1,0) -- min steam rate
define(W2,6) -- max steam rate

define(param_l_prime,25)
--Ndefine(param_l,70)
define(param_u_prime,200)

define(N_1,100)
define(N_2,150)

define(MIN,5)
define(MAX,220)

var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        p2,     -- pump volume from Pump 2 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        t,      -- controller's clock
        t1,     -- pump controller 1's clock
        t2      -- pump controller 2's clock
          : clock;
        param_l     -- parameter L (above L , leave pumps idle,
                    --              below activate both)
          : parameter;


--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw = dp1 + dp2 - dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

synclabs: p_1_on, p_1_off;
initially off ;

loc off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto going_on;

loc going_on: while t1<=5 wait {dp1 = 0}
        when t1=5 do {t1'=t1'} goto on;

loc on: while True wait {dp1=P_rate}
        when True sync p_1_off goto off;
end


--------------------------------------------------

automaton pump_cont_2
```

```
synclabs: p_2_on, p_2_off;
initially off ;

loc off: while True wait {dp2=0}
        when True sync p_2_on do {t2'=0} goto going_on;

loc going_on: while t2<=5 wait {dp2 = 0}
        when t2=5  do {t2'=t2'} goto on;

loc on: while True wait {dp2=P_rate}
        when True sync p_2_off goto off;
end

--------------------------------------------------

automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain = 0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain = 1}
        when True sync close_valve goto closed;
end

--------------------------------------------------

automaton steam

synclabs: start ;
initially idle;

loc idle: while True wait {dsteam = 0}
        when True sync start goto running;

loc running: while True wait {dsteam in [W1,W2]}

end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do
-- some redundancy in off_on/on_off modes, since the normal range
-- is adjacent to other regions where no action is taken
```

```
synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on, p_1_off, p_2_on, p_2_off,
        open_valve, close_valve;

initially idle & t=0;

-- the initialization mode

loc idle: while True wait {}
        when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
        when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
        when w>=N_2 sync open_valve goto wait_till_drained;
        when w<=N_1 sync p_1_on goto going_wait_till_fill;
        when N_1<=w & w<=N_2 sync start  do {t'=5} goto off_off;

loc wait_till_drained: while w>=N_2 wait {}
        when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
        when asap  sync start  do {t'=5} goto off_off;

loc going_wait_till_fill: while True wait {}
        when asap sync p_2_on  goto wait_till_fill;

loc wait_till_fill: while w<=N_1 wait {}
        when w=N_1 sync start do {t'=5} goto on_on;

-- the normal operating mode

loc off_off: while t<=5 wait {}
        when t=5 & w<param_l_prime goto emergency_stop;
        when t=5 & param_l_prime<=w & w<=param_l do {t'=0} sync p_1_on
                goto going_on_on;
        when t=5 & param_l<=w & w<=N_1 do {t'=0}
                goto off_off;
        when t=5 & N_1<=w & w<=N_2 do {t'=0} goto off_off;
        when t=5 & N_2<=w & w<=param_u_prime do {t'=0}
                goto off_off;
        when t=5 & w>param_u_prime goto emergency_stop;


loc on_on: while t<=5 wait {}
```

```
        when t=5 & w<param_l_prime  goto emergency_stop;
        when t=5 & param_l_prime<=w & w<=param_l do {t'=0} goto on_on;
        when t=5 & param_l<=w & w<=N_1 do {t'=0} sync p_1_off
                goto going_off_off;
        when t=5 & N_1<=w & w<=N_2 do {t'=0} goto on_on;
        when t=5 & N_2<=w & w<=param_u_prime do {t'=0}  sync p_1_off
                goto going_off_off;
        when t=5 & w>param_u_prime goto emergency_stop;



loc going_on_on: while True wait {}
        when asap sync p_2_on goto on_on;


loc going_off_off: while True wait {}
        when asap sync p_2_off goto off_off;


-- emergency stop mode

loc emergency_stop: while t<=5  wait {}
end


--------------------
-- analysis commands

var
        assumption,
        init_reg_loc,
        init_reg,
        final_reg,
        reached,
        reached_final: region;

init_reg_loc :=   loc[steam_boiler] = s0
                & loc[pump_cont_1] = off
                & loc[pump_cont_2] = off
                & loc[valve] = closed
                & loc[steam] = idle
                & loc[controller] = idle ;


assumption := MIN<=param_l_prime & param_l_prime<=param_l
                & param_l<=N_1
                & N_2<=param_u_prime & param_u_prime<=MAX;

-- initially t = 5, so that a control decision is made immediately
init_reg :=       init_reg_loc
```

```
                & param_l<= w & w<=param_u_prime
                & assumption;


final_reg :=  w>=MAX
            | w<=MIN
            | loc[controller]=emergency_stop;

reached := reach forward from init_reg endreach;

reached_final := reached & final_reg;

prints "Parametric constraints causing violation of MIN and MAX bounds";
print omit all locations
        hide non_parameters in reached_final endhide;
```

A.8 Verification of the dual-pump fault-free steam-boiler system with controller C_B and steam model A^S_B

```
---------------------------
-- HyTech input file
--
-- Steam boiler
--
-- Steam model A^S_B. Verification of controller C_B
--
-- time slot is 5 units long

define(P_rate,8) -- pump rate for both pumps combined
define(W1,0)     -- min steam rate
define(W,6)      -- max steam rate

define(U1,2/5)  -- max gradient of steam rate
define(U2,2/5)  -- - min gradient of steam rate

define(delta,5)

define(delta_U1,2)
define(delta_U2,2)

define(acc_loss,5)             -- delta^2/2.U2
define(acc_gain,5)             -- delta^2/2.U1
define(two_round_acc_gain,20) -- actually 4*acc_gain, i.e. gain over
                                -- double time period

define(MIN,5)
define(param_l_prime,25)
define(init_l,70)
define(N_1,100)
define(N_2,150)
define(init_u,170)
define(param_u_prime,200)
define(MAX,220)

var
        w,      -- water level
        p1,     -- pump volume from Pump 1 for a time slot
        steam,  -- steam volume for a time slot
        drain
          : analog;
        steam_rate
          : discrete;
        t,      -- controller's clock
        tsteam, -- clock for monitoring steam level is consistent
                --      with steam acceleration rate
        t1      -- pump controller 1's clock
          : clock;

--------------------------------------------------

automaton steam_boiler

synclabs: ;
initially s0;

loc s0: while True wait {dw = dp1 - dsteam - ddrain}

end

--------------------------------------------------

automaton pump_cont_1

synclabs: p_1_on, p_1_off;
initially off ;

loc off: while True wait {dp1=0}
        when True sync p_1_on do {t1'=0} goto going_on;

loc going_on: while t1<=5 wait {dp1=0}
        when t1=5 do {t1'=t1'} goto on;

loc on: while True wait {dp1=P_rate}
        when True sync p_1_off goto off;

end
```

```
--------------------------------------------------


automaton valve

synclabs: open_valve, close_valve;
initially closed;

loc closed: while True wait {ddrain=0}
        when True sync open_valve goto open;

loc open: while True wait {ddrain=1}
        when True sync close_valve goto closed;
end

--------------------------------------------------


automaton steam

-- bounds on steam's double derivative loosely bounded by checking
--      last steam  rate, and calculating loose lower and upper bounds
--      on next volume of steam.
-- real relationships use quadratic formulae
--      If the steam volume is not consistent, then time stops.
-- the model allows for more liberal steam during running, which will
--      result in incorrect global bounds on w.  If necessary, this
--      can be corrected by measuring water levels only in the
--      checked_steam mode

synclabs: start;
initially idle & steam=0 & steam_rate=0;

loc idle: while True wait {dsteam=0}
        when True sync start
                do {tsteam'=0,steam'=0,steam_rate'=0}
                goto running;

loc running: while tsteam<=delta & steam_rate>=0 & steam_rate<=W
                 wait { dsteam in [0,W] }
        when tsteam=delta & t=delta
                -- i.e. BEFORE the controller reads the values
                &  steam>=delta steam_rate - acc_loss
                &  steam<=delta steam_rate + acc_gain
                goto checked_steam;




loc checked_steam: while tsteam=delta & steam_rate>=0 & steam_rate<=W
                  wait { dsteam=0 }
        when tsteam=delta  -- i.e. BEFORE the controller reads the values
                do {
                        steam_rate'<=steam_rate + delta_U2,
                        steam_rate'>=steam_rate - delta_U1,
                        delta steam_rate'<=steam +  acc_gain,
                        delta steam_rate'>=steam -  acc_loss,
                        steam'=0,
                        tsteam'=0
                }
                goto running;
end

--------------------------------------------------

automaton controller

-- in normal mode, look at water level only, and decide what to do
synclabs:
        steam_boiler_waiting,
        steam_rate_zero,
        start,  -- to turn on all systems, esp. the boiler
        p_1_on,  p_1_off,
        open_valve, close_valve;

initially idle & t=0;

-- the initialization mode

loc idle: while True wait {}
        when True sync steam_boiler_waiting do {t'=0} goto test;

loc test: while t=0 wait {}
        when True sync steam_rate_zero goto init;

loc init: while t=0 wait {}
        when w>=N_2 sync open_valve goto wait_till_drained;
        when w<=N_1 sync p_1_on goto wait_till_fill;
        when N_1<=w & w<=N_2 sync start  do {t'=delta} goto off;

loc wait_till_drained: while w>=N_2 wait {}
        when w=N_2 sync close_valve goto wait_till_drained_b;

loc wait_till_drained_b: while True wait {}
        when asap  sync start  do {t'=delta} goto off;

loc wait_till_fill: while w<=N_1 wait {}
        when w=N_1 sync start do {t'=delta} goto on;

-- the normal operating mode
```

```
loc off: while t<=delta wait {}
        when t=delta & tsteam=0 & w<param_l_prime goto emergency_stop;

        when t=delta & tsteam=0
                & param_l_prime<=w
                & w<=N_1
                & 10 steam_rate + two_round_acc_gain >=60
                & w<=60 + param_l_prime
                do {t'=0} sync p_1_on
                goto on;
        when t=delta & tsteam=0
                & param_l_prime<=w
                & w<=N_1
                & 10 steam_rate + two_round_acc_gain<=60
                & w<=param_l_prime + 10 steam_rate + two_round_acc_gain
                do {t'=0} sync p_1_on
                goto on;
        when t=delta & tsteam=0
                & param_l_prime<=w
                & w<=N_1
                & 10 steam_rate + two_round_acc_gain >=60
                & w>=param_l_prime + 60
                do {t'=0}
                goto off;
        when t=delta & tsteam=0
                & param_l_prime<=w
                & w<=N_1
                & 10 steam_rate + two_round_acc_gain <=60
                & w>=param_l_prime + 10 steam_rate + two_round_acc_gain
                do {t'=0}
                goto off;

        when t=delta & tsteam=0 & N_1<=w & w<=N_2 do {t'=0} goto off;

        when t=delta & tsteam=0
                & N_2<=w
                & w<=param_u_prime
                do {t'=0}
                goto off;

        when t=delta & tsteam=0 & w>param_u_prime goto emergency_stop;


loc on: while t<=delta wait {}
        when t=delta & tsteam=0 & w<param_l_prime  goto emergency_stop;

        when t=delta & tsteam=0
                & param_l_prime<=w
                & w<=N_1
                do {t'=0}
                goto on;

        when t=delta & tsteam=0 & N_1<=w & w<=N_2 do {t'=0} goto on;

        when t=delta & tsteam=0
                & N_2<=w
                & w<=param_u_prime
                do {t'=0}
                sync p_1_off
                goto off;

        when t=delta & tsteam=0 & w>param_u_prime  goto emergency_stop;

-- emergency stop mode

loc emergency_stop: while t<=delta  wait {}
end



--------------------
-- analysis commands

var
    init_reg, final_reg, reached, reached_final: region;

init_reg :=       loc[steam_boiler]=s0
                & init_l<=w & w<=init_u
                & loc[pump_cont_1]=off
                & loc[valve]=closed
                & loc[steam]=idle
                & steam=0
                & steam_rate=0
                & loc[controller]=idle
                & MIN<=param_l_prime
                & param_l_prime<=N_1
                & N_2<=param_u_prime
                & param_u_prime<=MAX;


final_reg := loc[controller]=emergency_stop | w<=MIN | w>=MAX;

reached := reach forward from init_reg endreach;

reached_final := reached & final_reg;

if empty(reached_final)
    then prints "Water level maintained between bounds MIN and MAX";
    else prints "Water level NOT maintained between bounds MIN and MAX";
         prints "Violating states";
```

```
          print reached_final;
          print trace to final_reg using reached;
endif;
```