Proving Safety Properties of the Steam Boiler Controller

Formal Methods for Industrial Applications: A Case Study

Gunter Leeb
leeb@auto.tuwien.ac.at
Vienna University of Technology
Department for Automation
Treitlstr. 3, A-1040 Vienna, Austria

Nancy Lynch
lynch@lcs.mit.edu
Massachusetts Institute for Technology
Laboratory for Computer Science
Technology Square 545, Cambridge, MA

Abstract

In this paper we model a hybrid system consisting of a continuous steam boiler and a discrete controller. Our model uses the Lynch-Vaandrager Timed Automata model to show formally that certain safety requirements can be guaranteed under the described assumptions and failure model. We prove incrementally that a simple controller model and a controller model tolerating sensor faults preserve the required safety conditions. The specification of the steam boiler and the failure model follow the specification problem for participants of the Dagstuhl Meeting "Methods for Semantics and Specification."

1 Introduction

The number of different formal methods for specifying, designing, and analyzing real-time systems has grown difficult to survey. For the purpose of comparison, some problems have been defined or borrowed from real-life applications. One such benchmark problem is the Steam Boiler Controller problem discussed in this paper. Another representative of this kind of problem is the Generalized Railroad Crossing (GRC) [Hei93]. Various approaches have been applied to the latter, e.g., [Cle93,Jah86,Sha93,Hoa93]. Many steps of the approach described here are similar to the steps described in [Hei94].

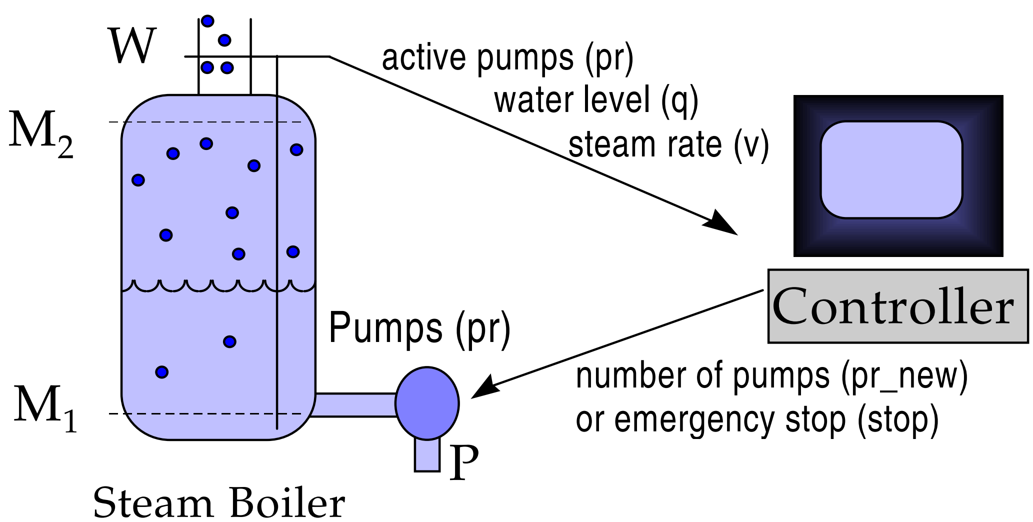

Figure 1: The steam boiler system. This picture shows the information flow between the controller and the steam boiler. It also gives some notion about the capacities of a pump (***P***), the limits for the steam rate (***W***) and the boundaries for the water level (***M_1*** and ***M_2***).

However, the Steam Boiler Controller represents a different kind of problem. Basically, it consists of a discrete control loop where several components may fail. The full version of this paper on the CD-ROM contains a condensed and informal description of the Steam Boiler Controller specification. The original specification can be found in [AS96]. Since even the original specification is informal and ambiguous, the condensed version on the CD-ROM summarizes our interpretation of the described problem.

The rest of this paper is organized as follows: After presenting an outline of our formal methods (Section 2), we state the assumptions we make for our model and show how the model is related to the physical model (Section 3). The following two sections describe the model of the boiler and a simple controller. In Section 6, we show some key model invariants. In Section 7, we present a similar controller which allows for sensor faults and we show its correctness incrementally based on the simpler controller model.

2 The Formal Framework

Applying formal methods to a system involves three steps: the system requirements specification, the design of an implementation, and the verification that the implementation satisfies the specification. The system requirements specification describes all acceptable system implementations [Hei94]. It has three parts:

1. A formal model describing the environment (e.g., the steam boiler) and its interface
2. A formal model describing the controller system and its interface at an abstraction level
3. Formal statements of the properties that the system must satisfy

The formal method we used to specify the steam boiler problem and to develop and verify a solution represents both the controller and the system environment as Timed Automata, according to the definition of Lynch and Vaandrager [Lyn91]. A Timed Automaton is a very general automaton, i.e., a labeled transition system. It is not finite-state: for example, the state can contain real-valued information, such as the current time or the current steam rate. This characteristic makes Timed Automata suitable for modeling not only discrete computer systems but also real-world entities such as the steam boiler. We base our work directly on an automaton model rather than on any particular specification language, programming language, or proof system, so that we may obtain the greatest flexibility in selecting specification and proof methods. The formal definition of a Timed Automaton appears in the CD-ROM version in Appendix A. Appendix B describes the Simulation Mapping method used for incremental reasoning about other increasingly specific instances of the model.

The Timed Automaton model supports the description of systems as collections of Timed Automata, interacting by means of common actions. In our example, we define separate Timed Automata for the steam boiler and the controller system; the common actions are sensors reporting the current state of some parameters of the boiler and actuators controlling the pumps of the boiler.

Actions change the state and, in particular, some variables of the state of an automaton. As a distinction between variables of the pre-state and the post-state, we write variables of the post-state (or the representation of the whole post-state) with a prime. In changing the state, actions perform a step or transition. Such a step or transition defines the change from one state *s* to another state *s'* by an action *a*, which is formally written as *(s, a, s')* or $s_A \xrightarrow{a}_A s'_A$, where the subscript A stands for the name of the particular automaton.

For the communication with other automata, we define input, output and internal actions. Such input actions will be enabled by output actions of another automaton. For example, the actuator output action in the controller model is synchronized with the actuator input action of the steam boiler model. The inherent flexibility of the method allows, for example, the introduction of a new automaton representing channel and message transfer characteristics to be employed in-between the boiler automaton and the controller automaton, interfacing with an input action from the controller and an output action to the steam boiler model. This allows us to model more complex systems without major changes to the previous automata. Furthermore, with this composition, we can reuse information we gained about the separate automata.

We describe the Timed Automata using precondition-effect notation. The precondition identifies particular states in which the system performs some actions. For any state fulfilling the precondition, the effect part describes how the state is changed by the particular action. This has several advantages. First of all, it is easy to understand. Even more important is that implementations can follow the abstract model description and even allow for simple validity checks in the code. In addition, all the invariants proved represent useful checks to be validated while running the final application. This approach will help to identify rare kinds of faults that are not even considered in the model. In this view, formal verification with Timed Automata is a constructive approach to systems development.

3 Further Considerations for Our Model

For our model, we need to know some more information about the physical behavior. Some of the following assumptions follow the informal specification of [AS96] or are intended to resolve some ambiguity. As suggested by [AS96], to simplify reasoning about the model, we ignore second order effects like the volume expansion of water when heated. This reasoning implies that a unit of water measured as steam can be replaced by pumping in exactly one unit of water.

Most important is some knowledge about how fast the steam rate may change over time. We assume a reasonable worst case situation where the steam rate increases at most with U_1 liters per second per second. In other words, the maximum gradient of increase of the steam rate is U_1 l/s^2. Symmetric to this, we know that the fastest decrease of the steam rate is denoted with U_2 l/s^2.

Furthermore, no pump supplies water unless activated and then it supplies a constant, exactly known amount of water per second denoted with P liters per second. The delay between reading the sensors and consequently changing the active pumps, denoted with S, is caused mainly by the slow reaction of the physical pumps. As a minor difference to the specification in [AS96], we assume the same delay for the activation and the deactivation of pumps. Since the pumps cause most of the delay S, we assume any boiler shut down is activated instantaneously and the whole process of shutting down the steam boiler is left to a later phase which we do not consider in this model. In the same way, we omit the initialization phase, which should force the boiler state into a particular acceptable set of start states before the boiler becomes fully operational. We assume all parameters of the start state for this model are already in their correct operational ranges. Moreover, we assume that the controller may decide to shut down the boiler any time it sets the new pumps. This assumption includes the possibility that the operator initiates an emergency stop, and it provides the flexibility to incorporate other reasons to shut down the boiler.

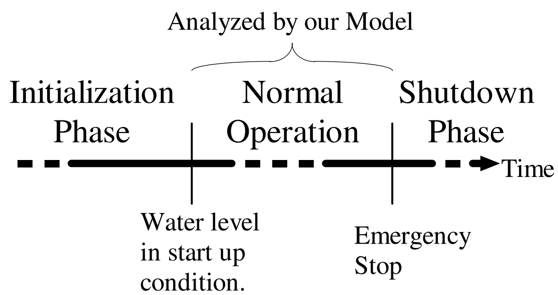

Figure 2: Our model only considers the time of normal operation. At the beginning, the initialization phase provides all parameters in the correct range and the shutdown phase is activated through setting parameter *stop* to ***true***.

Other helpful assumptions are correct and accurate sensor values or the detection of a sensor fault. Perfect fault detection and identification are necessary for our model but will not be available in reality. In this aspect our model might need improvement if it is necessary to study such general cases. For example, the techniques developed for probabilistic Timed Automata [Seg94] seem to be appropriate for a problem requiring the analysis of such probabilistic properties. Probabilistic Timed Automata would allow one to assign probabilities to certain actions, e.g., for a successful error detection, and to prove the probability of a certain system behavior.

As a further simplification, we choose a very simple fault model which, in fact, includes or is close to most common fault conditions. The fault model assumes that every pump may fail and stop pumping water into the boiler. As a minor simplification, we assume for our model that any pump fault only occurs at times when pumps may be activated or stopped. This happens periodically whenever the parameter *set* equals the current time (*now*). Thus, pumps, when successfully activated, supply water at least to the next instant where pumps might change their behavior. Moreover, we assume that the activation delay, i.e., the time from reading the sensor values until consequently the pumps change their behavior, is smaller than the time between two successive sensor readings ($\boldsymbol{S} < \boldsymbol{I}$).

The goal of modeling the steam boiler and the controller with Timed Automata is to show certain important properties. In this case, we want to verify that our controller model does not violate safety. Therefore, we have to show that neither the steam rate nor the water level crosses its critical limits.

Next, we summarize the information we have about the physical model.

3.1 The Physical Model

We assume the steam rate expressed as a function over time ($sr(t) \geq 0$) is differentiable. Furthermore, we know that

$$-U_2 \leq \dot{sr}(t) \leq U_1$$

and

$$wl(t) = wl(0) + \int_0^t pr(x)dx - \int_0^t sr(x)dx$$

where $\dot{sr}(t)$ represents the derivative of the steam rate function and $wl(t)$ the amount of water in the boiler at the time t and $pr(t)$ (≥ 0) the (discrete) pump rate function over time. We apply the following transformation to this information to make our model easier to follow.

We know $-U_2 \leq \dot{sr}(t)$, which implies $0 \leq \dot{sr}(t) + U_2$ and in general

$$\int \dot{sr}(t) + U_2 dt = sr(t) + t * U_2 + C.$$

Thus, we know that for all Δt,

$$sr(t + \Delta t) + U_2 * \Delta t \geq sr(t)$$

and symmetrically

$$sr(t + \Delta t) - U_1 * \Delta t \leq sr(t) .$$

In the following, we use s for $sr(t)$ and s_{new} for $sr(t + \Delta t)$. With a similar straightforward calculation as before, we get

$$wl(t + \Delta t) \geq wl(t) + \int_t^{t+\Delta t} pr(x)dx - \delta_{HIGH}\left(s, s_{new}, \Delta t\right)$$

and symmetrically

$$wl(t + \Delta t) \leq wl(t) + \int_t^{t+\Delta t} pr(x)dx - \delta_{LOW}\left(s, s_{new}, \Delta t\right)$$

with

$$\delta_{HIGH}\left(s, s_{new}, \Delta t\right) = \left(\frac{2\Delta t U_2 s + 2\Delta t U_1 s_{new} + \Delta t^2 U_1 U_2 - (s - s_{new})^2}{2U_1 + 2U_2}\right)$$

and

$$\delta_{LOW}\left(s, s_{new}, \Delta t\right) = \begin{cases} \left(\dfrac{2\Delta t U_1 s + 2\Delta t U_2 s_{new} - \Delta t^2 U_1 U_2 + (s - s_{new})^2}{2U_1 + 2U_2}\right) & \text{if } \left(\dfrac{s}{U_2} + \dfrac{s_{new}}{U_1}\right) > \Delta t \\ \left(\dfrac{s^2}{2U_2} + \dfrac{s_{new}{}^2}{2U_1}\right) & \text{otherwise} \end{cases}$$

δ_{HIGH} describes the maximum amount of water that could evaporate and δ_{LOW} the minimum amount of water. Obviously, δ_{LOW} depends on whether the steam rate might drop to 0 in the interval Δt. Figure 3 represents δ_{HIGH} and δ_{LOW} graphically for an arbitrary interval t. Figure 3 ignores the pump rate, and the shaded areas represent the water evaporated into steam until a certain point in time. In other words, δ_{HIGH} and δ_{LOW} represent the worst case amount of water that could evaporate into steam in interval Δt. Both depend on the knowledge of the steam rate at the beginning and the end of the interval. The basic dependencies shown in the following Lemma 1 are sufficient for all further proofs.

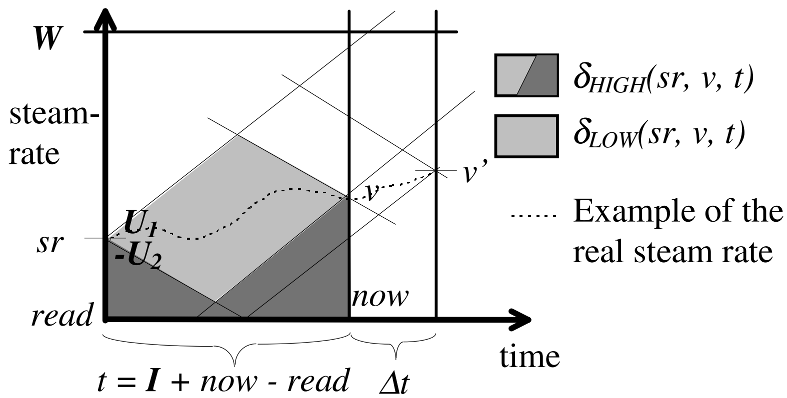

Figure 3: Example of what δ_{HIGH} and δ_{LOW} represent. For different intervals the maximum and minimum amount of water evaporated into steam depends on the steam rate at the beginning of the interval and at the end.

The following Lemma lists all necessary relations about the steam development functions δ_{HIGH} and δ_{LOW}. The intuition for this lemma can be gained from Figure 3. Obviously, two consecutive intervals can be joined and the minimum and maximum amount of water is smaller and bigger respectively or equal to the minimum/maximum water evaporated in both subintervals.

Lemma 1: For all $a, b, c \geq 0$, all constants > 0 and $t, u > 0$:

1) $\delta_{LOW}(a, b, u) \leq \delta_{HIGH}(a, b, u)$
2) $\delta_{LOW}(a, b, u) \geq \begin{cases} a^2/(2*\mathbf{U_2}) & \text{if } a < \mathbf{U_2} * u \\ a * u - \mathbf{U_2}*u^2/2 & \text{otherwise} \end{cases}$
3) $\delta_{LOW}(a, b, u) \geq \begin{cases} b^2/(2*\mathbf{U_1}) & \text{if } b < \mathbf{U_1} * u \\ b * u - \mathbf{U_1}*u^2/2 & \text{otherwise} \end{cases}$
4) $\delta_{LOW}(a, b, u) + \delta_{LOW}(b, c, t) \geq \delta_{LOW}(a, c, t + u)$
5) $(a + b)*u/2 \geq \delta_{LOW}(a, b, u)$
6) $\delta_{HIGH}(a, b, u) \leq (b * u + \mathbf{U_2}*u^2/2)$
7) $\delta_{HIGH}(a, b, u) + \delta_{HIGH}(b, c, t) \leq \delta_{HIGH}(a, c, u + t)$
8) $\delta_{HIGH}(a, b, u) \geq (a + b)*u/2$
9) $\delta_{HIGH}(a, b, u) \leq (a * u + \mathbf{U_1}*u^2/2)$

Proof: 1. - 9.: By calculus.

■

Based on this information, we can now model the steam boiler as a Timed Automaton.

4 The Boiler Model

Constants

Name	Type	Restriction	Unit	Description
I	positive real	$> \boldsymbol{S}$	s	time in-between periodical sensor readings
S	positive real	$< \boldsymbol{I}$	s	delay to activate pumps after the last sensor reading
$\boldsymbol{U_1}$	positive real		l/s^2	maximum gradient of the increase of the steam rate
$\boldsymbol{U_2}$	positive real		l/s^2	maximum gradient of the decrease of the steam rate
$\boldsymbol{M_1}$	real	$\geq 0, < \boldsymbol{M_2}$	l	minimum amount of water before boiler becomes critical
$\boldsymbol{M_2}$	positive real	$\leq \boldsymbol{C}, > \boldsymbol{M_1}$	l	maximum amount of water before boiler becomes critical
W	positive real		l/s	maximum steam rate before boiler becomes critical
P	positive real		l/s	exact rate at which one active pump supplies water to the boiler
#pumps	integer	> 0		number of pumps that can supply water to the boiler in parallel
C	positive real	$\geq \boldsymbol{M_2}$	l	capacity of the boiler

Table 1: Constants and their relations for the boiler and controller models

Variables

Name	Initial Value	Type	Values Range	Unit	Description
now	*0*	real	$[0 \ldots \infty)$	s	current time
pr	*0*	integer	{*0, …* ***#pumps***}		number of pumps actively supplying water to the boiler
q	$>> \boldsymbol{M_1}$, $<< \boldsymbol{M_2}$	real	$[0 \ldots \boldsymbol{C}]$	l	actual water level in the boiler
v	*0*	real	$[0 \ldots \infty)$	l/s	steam rate of the steam currently leaving the boiler
pr_new	*0*	integer	{*0, …* ***#pumps***}		number of pumps that are supposed to supply water after the activation delay
error	*0*	integer	{*0, …* ***#pumps***}		number of pumps that fail to supply water to the boiler after activation
do_sensor	***true***	boolean	{***true***, ***false***}		enable a single sensor reading
set	***S***	real	$[0 \ldots \infty)$	s	next time the pumps change to the new settings
read	*0*	real	$[0 \ldots \infty)$	s	next time the sensors will be read
stop	***false***	boolean	{***true***, ***false***}		flag that determines whether emergency shut-down is activated

Table 2: Variables of the steam boiler model. They represent the (initial) state of the steam boiler.

For providing a formal description of the steam boiler, we first define all constants and the state (Table 1 and Table 2). For all variables of the state, we provide the type, value range and description. Moreover, we describe the initial state which immediately forces the automaton to read the current sensor values and forwards them to the controller. The controller will provide an appropriate pump setting. The checks in the controller, which is described in the following section, require that there is a certain minimal amount of water between the critical limits or otherwise the controller would stop the steam boiler at once. Thus, a valid start condition of the water level and steam rate must be far enough from the critical boundaries not to force the controller to execute an emergency stop.

4.1 The Boiler Automaton

Expressing our interpretation of the informal specification more precisely leads to the following Timed Automaton:

Input Action

actuator (e_stop, pset)

Effect:

$pr\_new' = pset$
$stop' = e\_stop$
$do\_sensor' = \textbf{true}$
$read' = now + \boldsymbol{I}$

Output Action

sensor (s, w, p)

Precondition:

$now = read$
$do\_sensor = \textbf{true}$
$stop = \textbf{false}$
$w = q$
$s = v$
$p = pr$

Effect:

$do\_sensor' = \textbf{false}$

Internal Actions

activate

Precondition:

$now = set$
$stop = \textbf{false}$

Effect:

$set' = read + \boldsymbol{S}$
$0 \leq error' \leq pr\_new$
$pr' = pr\_new - error'$

$\nu(\Delta t)$

Precondition:

$stop = \textbf{false}$
$now + \Delta t \leq read$
$now + \Delta t \leq set$

Effect:

$v - \boldsymbol{U_2} * \Delta t \leq v' \leq v + \boldsymbol{U_1} * \Delta t$
$q + pr * \boldsymbol{P} * \Delta t - \delta_{HIGH}(v, v', \Delta t) \leq q'$
$q' \leq q + pr * \boldsymbol{P} * \Delta t - \delta_{LOW}(v, v', \Delta t)$
$now' = now + \Delta t$

This formal description of the steam boiler is easily readable: The steam boiler reads periodically the current water level and the current steam rate and forwards these values to the controller. In the addition, the controller learns about the number of pumps that currently actually supply water to the boiler. The controller evaluates the data and through the actuator supplies a new pump setting or enables the shut-down phase. After the activation delay, all non-faulty pumps of the new setting supply water to the boiler. In the meantime, water evaporates into steam unpredictably but limited by its worst case rules.

With the ***actuator*** action the boiler receives the new pump setting requested by the controller and learns whether the controller shuts down the boiler. Furthermore, it schedules and enables the next reading of the sensor values. After an emergency stop is executed by setting the variable ***stop*** to ***true***, our model ignores any further development.

As an internal action, the boiler changes the steam rate and the water level unpredictably over time. The purpose of the ***time-passage action*** denoted with $\nu(\Delta t)$ is to provide a method for describing formally a time-dependent process. Δt represents an arbitrary, non-empty interval of time. A possible value for the parameter Δt depends on the precondition. Obviously, Δt may be arbitrary as long as the next activation of the pumps and the next sensor reading occur. Formally, the time-passage action must follow some rules as described in CD-ROM version in the Appendix A, which we are going to verify in the next section.

The ***activate action*** occurs after the pump activation delay. It sets the new pump rate with respect to an arbitrary number of pumps that fail, expressed as *error*. We chose this rather strong fault model where all pumps might fail at the activation time regardless whether such a pump was already supplying water before. This can be as much as all pumps that should supply water for the next cycle. Finally, it schedules the next activation time. Periodically, the ***sensor action*** forwards the current amount of water, the current steam rate and the number of active pumps to the controller. To prevent the sensor action from happening multiple times, it disables itself by setting $do\_sensor = \textbf{\textit{false}}$.

4.2 Checking the Model

As described formally in the CD-ROM version in Appendix A (the complete definition can be found in [Lyn91]), each Timed Automaton has to follow five axioms. We need to show that the Boiler Model satisfies these axioms. Overall, these axioms are used to define the concept of time in Timed Automata. Axiom [A1] says that the current time is always 0 in a start state. Axiom [A2] says that non-time-passage steps do not change the time; that is, they occur "instantaneously", at a single point in time. Axiom [A3] says that time-passage steps must cause the time to increase. The fourth axiom [A4] enforces transitivity in the representation of time, i.e., transitivity of the time passage action. Whenever it is possible to describe a development over time with several succeeding time-passage steps it must be possible to describe this change in a single time-passage step. The fifth axiom [A5] describes trajectory consistency: Whenever the change from one state to another with the time-passage action can be expressed as a trajectory (or function), the change between any two states in this interval follows the same trajectory.

The CD-ROM version contains the details to these proofs. Basically, with these axioms fulfilled the Timed Automaton model allows us to combine automata through their input and output actions. We will combine the boiler model with a controller model, which we present in the next section.

4.3 Properties of the Boiler

Based on the automaton description, we can derive useful information about the boiler system. These intermediate results can be favorably employed for fault detection and consistency checks in any actual boiler implementation based on this model. This information is expressed in the form of logic expressions invariant in all possible executions of this boiler model. Therefore, these expressions are called *invariants*. In other words, no order of steps will produce a state in which any of these logical expressions is not true. All the statements and their proofs can be found in the CD-ROM version of this report. All proofs are by induction on the steps of the automaton.

5 The Controller Model

In order to solve the steam boiler problem, we have to find a controller that guarantees the required safety properties. For this purpose, we take advantage of a characteristic of the Timed Automaton model. First, we will show that a simple controller that cannot tolerate sensor faults guarantees the safety properties under described assumptions. Then, the Simulation Mapping technique is used to show incrementally that a different controller which allows for sensor failures preserves the safety properties.

Obviously, it is most important that the controller identifies water levels and steam rates that might cross their critical limits before the next sensor values arrive. In case such sensor values are identified the controller will enable the shut-down phase. In a non-critical case, the controller chooses an appropriate new setting for the pumps to adjust the water level and compensate for the amount of steam leaving the boiler.

5.1 The Controller Model

Definitions

Name	Type	Unit	Value	Description
max_pumps_after_set	integer		***#pumps***	maximum number of pumps that can supply water to the boiler after the delay considering the pump failure model
min_pumps_after_set	integer		*0*	minimum number of pumps that can supply water to the boiler after the delay considering the chosen pump failure model. For a different pump failure model, e.g., in which pumps might fail when activated or stopped, this constant may actually be a function of the change in the number of pumps.
min_steam_water(sr)	real	l	$sr^2/(2*\mathbf{U_2})$ *if* $sr < \mathbf{I} * \mathbf{U_2}$ $(sr - \mathbf{U_2} * \mathbf{I}/2)*\mathbf{I}$ *otherwise*	minimum amount of water that can evaporate into steam until the next sensor reading
max_steam_water(sr)	real	l	$(sr + \mathbf{U_1} * \mathbf{I}/2)*\mathbf{I}$	maximum amount of water that can evaporate into steam until the next sensor reading
min_steam_water_est(sr)	real	l	$sr^2/(2*\mathbf{U_1})$ *if* $sr < \mathbf{I} * \mathbf{U_1}$ $(sr - \mathbf{U_1} * \mathbf{I}/2)*\mathbf{I}$ *otherwise*	estimated minimum amount of water that has evaporated since the next sensor reading

Table 3: Definitions and abbreviations for the controller model

Variables

Name	Initial Value	Type	Value Range	Unit	Description
do_output	***false***	boolean	{***true, false***}		flag that enables the output. This represents a kind of program counter.
stopmode	***true***	boolean	{***true, false***}		flag to activate the shut down, initially true, since condition is not checked yet.
wl	*q*	real	[*0* ... ***C***]	l	current water level reading
sr	*0*	real	[*0* ... ***W***]	l/s	current steam rate reading
now	*0*	real	[*0* ... ∞)	s	current time
pumps	*0*	integer	{*0* ...***#pumps***}		number of currently active pumps supplying water to the boiler
px	*0*	integer	{*0* ...***#pumps***}		number of pumps that shell supply water next

Table 4: The state of the controller including all variables and their initial values

5.2 The Simple Controller Automaton

The input and output actions are complementary to the input and output actions of the steam boiler model.

Input Action

sensor (s, w, p)

Effect:

$sr' = s$
$wl' = w$
$pumps' = p$
$do\_output' = \mathbf{true}$

 # safety checks:
if $sr' \geq \mathbf{W} - \mathbf{U_1} * \mathbf{I}$ *or*
$wl' \geq \mathbf{M_2} - \mathbf{P} * (pumps' * \mathbf{S} + (max\_pumps\_after\_set) * (\mathbf{I} - \mathbf{S})) + min\_steam\_water(sr)$ *or*
$wl' \leq \mathbf{M_1} - \mathbf{P} * (pumps' * \mathbf{S} + (min\_pumps\_after\_set) * (\mathbf{I} - \mathbf{S})) + max\_steam\_water(sr)$
then
$stopmode' = \mathbf{true}$
else
$stopmode' = \{\mathbf{true}, \mathbf{false}\}$ *arbitrary*

Internal Actions

controller

Precondition:
true
Effect:
$0 \leq px' \leq \mathbf{\#pumps}$

ν(Δt)

Precondition:
true
Effect:
$now' = now + \Delta t$

Output Action

actuator (e_stop, pset)

Precondition:
$do\_output = \mathbf{true}$
$pset = px$
$e\_stop = stopmode$
Effect:
$do\_output' = \mathbf{false}$

With the ***sensor action***, the controller receives periodically the current steam rate, water level and number of activated pumps. Its primary purpose is to test if the current sensor values are “close” to either critical limit. In such a case the sensor action sets a flag for the actuator to initiate the shut-down. Likewise, external critical conditions are modeled by non-deterministically setting *stopmode* to true. Furthermore, the sensor action enables the actuator action. The test for what is "close" depends on the particular fault model used and controller capabilities. The controller can try to start all pumps every period and our fault model allows up to all pumps to fail. The point in time for the decision how many pumps actually supply water to the boiler is every *set* time. Therefore, we must choose all pumps for *max_pumps_after_set*. On the other hand, all pumps could fail and therefore *min_pumps_after_set* equals 0. Similarly, *min_steam_water* and *max_steam_water* express the minimum and maximum amounts of water that can evaporate into steam in the following period starting with given current steam rate, respectively. The test simply calculates the worst case situations for the water level and steam rate and compares the results with the critical limits $\boldsymbol{M_1}$, $\boldsymbol{M_2}$ and $\boldsymbol{W}$.

The ***controller action*** chooses an appropriate new pump setting. Actually, it can choose any pump setting. For our approach, we are not particularly interested in the performance of the controller. On the other hand, we are interested in generality. Therefore, we chose a controller model that can incorporate any possible control algorithm for setting the pumps. As a consequence, our results concerning the safety are valid for an arbitrary control algorithm. Although the choice of a new setting for the pumps is irrelevant to the safety of the steam boiler system, for a performance analysis the pump setting would be of major importance. The ***time-passage action*** ($\nu(\Delta t)$) allows time to pass. For the following proofs, we ignore these two actions, since they do not provide additional information and are irrelevant to the proofs.

Finally, the ***actuator action*** forwards the new pump setting and whether the boiler must be stopped to the boiler environment. Furthermore, it disables itself, by setting *do_output* back to false.

As suggested in the original specification, this controller model acts instantaneously. Therefore, the time-passage action is trivial and all five axioms for Timed Automata are satisfied. Moreover, there is no useful information gained from the controller model alone. So far the proofs have involved only either the steam boiler model or the controller model. Next, we use the composition property of Timed Automata for combining the two automata, and we prove the required safety properties.

6 Properties of the Combined Steam Boiler System

Following, we show in several steps that the combined model (formally a composition), consisting of the steam boiler model and the simple controller model together, guarantee the safety conditions. The first safety property requires that the steam rate must always stay below $\boldsymbol{W}$. Before the steam rate can cross this limit, the boiler must be shut down. Expressing this in terms of the state of the steam boiler system, we have to show

S1) $v < \boldsymbol{W}$ *or stop* = ***true***

The second safety property requires that the water level must always stay between its critical limits $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$. Before the water level can cross either limit, the boiler must be stopped. Thus, we have to show

S2) $\boldsymbol{M_1} < q < \boldsymbol{M_2}$ *or stop* = ***true***

6.2 Steam Boiler System Properties

The following lemmas lead us step-by-step toward proving the safety conditions. While we provide only some important statements in this version of this paper, all the statements (lemmas and theorems) and their proofs can be found on the CD-ROM. The numbering is consistent.

Coming up with the right invariants that lead to showing the safety properties is the most complicated task in working with Timed Automata. On the other hand, the proofs themselves are usually straightforward and follow well-established, stylized methods and the usual pattern for proving by induction. The main work for proving the safety properties is done by means of these invariants. All the proofs for our model are by induction on the model and can easily be verified using current mechanical proof technology.

The following lemma describes the conditions when the controller decides that the boiler needs to be emergency-stopped.

Lemma 3: In all reachable states of the controller model,

1) $\boldsymbol{M_2} > wl + \boldsymbol{P} * (pumps * \boldsymbol{S} + \boldsymbol{\#pumps} * (\boldsymbol{I} - \boldsymbol{S})) - min\_steam\_water(sr)$ *or stopmode* $=$ ***true***
2) $\boldsymbol{M_1} < wl + \boldsymbol{P} * pumps * \boldsymbol{S} - (sr * \boldsymbol{I} + \boldsymbol{U_1} * \boldsymbol{I}^2/2)$ *or stopmode* $=$ ***true***
3) $sr + \boldsymbol{U_1} * \boldsymbol{I} < \boldsymbol{W}$ *or stopmode* $=$ ***true***

Using the test conditions in Lemma 3.3, we can prove that the actual steam rate will stay under a certain limit depending on how long it takes until the next sensor reading. This lemma depends on $sr + \boldsymbol{U_1} * \boldsymbol{I} < \boldsymbol{W}$ *or stopmode* = ***true*** (**Lemma 3**.3) and *if do_output then now = read and sr = v* (Lemma 4).

Lemma 11: $v + \boldsymbol{U_1} * (read - now) < \boldsymbol{W}$ *or stop* = ***true***

The following lemma describes the amount of water above the lower limit of the water level depending on the current steam rate and pump rate. Lemma 12 depends on Lemma 1.9, $now \le set$ (Lemma 2), $wl > \boldsymbol{M_1} - \boldsymbol{P} * pumps * \boldsymbol{S} + (sr * \boldsymbol{I} + \boldsymbol{U_1} * \boldsymbol{I}^2/2)$ *or stopmode* = ***true*** (**Lemma 3**.2), *if do_output then now = read and wl = q and sr = v* (Lemma 4), $set = read + \boldsymbol{S}$ *or* $set = read - \boldsymbol{I} + \boldsymbol{S}$ (Lemma 5), $now \le read - \boldsymbol{I} + \boldsymbol{S}$ *or* $set = read + \boldsymbol{S}$ (Lemma 6), *if now < read then do_output* = ***false*** (Lemma 7) and Lemma 10: *if* $set = read + \boldsymbol{S}$ *and do_output* = ***false*** *then pr = pr_new - error else pr = pumps.*

Lemma 12: *if do_output* = ***false*** *then*

if $set = read - \boldsymbol{I} + \boldsymbol{S}$ *then*

$\boldsymbol{M_1} < q + \boldsymbol{P} * pumps * (set - now) - (v * (read - now) + \boldsymbol{U_1} * (read - now)^2/2)$ *or stop* = ***true***

else $\boldsymbol{M_1} < q - (v * (read - now) + \boldsymbol{U_1} * (read - now)^2/2)$ *or stop* = ***true***

Similar to the previous lemma, the following lemma describes the minimum amount of water in the boiler before the upper water level limit could be reached. This lemma depends on Lemma 1.2, $now \leq set$ (Lemma 2), $\mathbf{M_2} > wl + \mathbf{P}*(pumps*\mathbf{S} + \mathbf{\#pumps}*(\mathbf{I}-\mathbf{S})) - min\_steam\_water(sr)$ or $stopmode = \mathbf{true}$ with

$$min\_steam\_water(sr) = \begin{cases} sr^2/(2*\mathbf{U_2}) & \text{if } sr < \mathbf{U_2}*\mathbf{I} \\ (sr*\mathbf{I} - \mathbf{U_2}*\mathbf{I}^2/2) & \text{otherwise} \end{cases}$$

(Lemma 3.1), *if do_output then now = read and wl = q and sr = v* (Lemma 4), $set = read + \mathbf{S}$ or $set = read - \mathbf{I} + \mathbf{S}$ (Lemma 5), $now \leq read - \mathbf{I} + \mathbf{S}$ or $set = read + \mathbf{S}$ (Lemma 6), *if* $now < read$ *then* $do\_output = \mathbf{false}$ (Lemma 7) and Lemma 10: *if* $set = read + \mathbf{S}$ *and* $do\_output = \mathbf{false}$ *then* $pr = pr\_new - error$ *else* $pr = pumps$.

Lemma 13: *if* $do\_output = \mathbf{false}$ *then*

if $set = read - \mathbf{I} + \mathbf{S}$ *then*

$$\mathbf{M_2} > q + \mathbf{P}*(pumps*(set-now) + \mathbf{\#pumps}*(\mathbf{I}-\mathbf{S})) - steam \text{ or } stop = \mathbf{true}$$

else $\mathbf{M_2} > q + \mathbf{P}*\mathbf{\#pumps}*(read - now) - steam$ or $stop = \mathbf{true}$

$$\text{with } steam = \begin{cases} v^2/2*\mathbf{U_2} & \text{if } v < \mathbf{U_2}(read-now) \\ (v*(read-now) - \mathbf{U_2}*(read-now)^2/2) & \text{otherwise} \end{cases}$$

Lemma 14: $d(u)$ is convex:

$d(u) \geq min(0, d(S))$ for $S \geq u \geq 0$, $d(u) = A*u - B*u^2$ with A real and B positive real

6.3 Summarizing Theorems

The following theorems summarize the previous lemmas and translate them into the form in which the required safety properties were expressed.

Theorem 1: In all reachable states of boiler system, $v < \mathbf{W}$ or $stop = \mathbf{true}$.

Theorem 2: In all reachable states of boiler system, $\mathbf{M_1} < q < \mathbf{M_2}$ or $stop = \mathbf{true}$.

With above proofs, we have shown that the steam boiler model together with the controller model meets all the safety requirements. As a further step, we must modify the controller model to allow sensor faults. This is presented in the following section.

7 Sensor Fault-tolerant Controller

In this section, we extend the model of the controller to be tolerant to sensor faults. Rather than proving the safety properties all over again, we use a technique called *Simulation Mapping*. This technique is used to show consistency between abstraction levels. In particular, it provides a means to show that properties proved for an abstract model are preserved in a particular implementation. In this case, the previously described boiler system represents the specification and a new controller that tolerates sensor faults represents a possible implementation.

The two lemmas which supply additional information about the boiler system with the previous controller can be found in the CD-ROM version of this text. Next, we present the Timed Automaton model of the sensor fault-tolerant controller.

7.1 The Controller Model Allowing Sensor Faults

Variables

Name	Initial Value	Type	Value Range	Unit	Description
do_output	***false***	boolean	{***true, false***}		flag that activates the output; This parameter represents a kind of program counter.
stopmode	***true***	boolean	{***true, false***}		flag to activate the emergency stop, initially true, since condition is not checked yet.
wll	*q*	real	[0 ... **C**]	l	lower bound of the estimation of the current water level
srl	0	real	[0 ... **W**]	l/s	lower bound of the estimation of the current steam rate
wlh	*q*	real	[0 ... **C**]	l	upper bound of the estimation of the current water level
srh	0	real	[0 ... **W**]	l/s	upper bound of the estimation of the current steam rate
sr_ok	***true***	boolean	{***true, false***}		flag that tells whether the steam rate sensor has failed
wl_ok	***true***	boolean	{***true, false***}		flag that tells whether the water level sensor has failed
now	0	real	[0 ... ∞)	s	current time
pumps	0	integer	{0 ... ***#pumps***}		number of currently active pumps supplying water to the boiler
px	0	integer	{0 ... ***#pumps***}		number of pumps that shall supply water next

Table 5: The initial state of the fault-tolerant controller including all variable declarations

7.2 The Fault-tolerant Controller Automaton

Input Action

sensor (s, w, p)

Effect:

pumps' = p
do_output' = ***true***

\# estimate steam rate
if sr_ok then srh' = srl' = s
else $srh' = srh + \boldsymbol{U_1} * \boldsymbol{I}$
$srl' = srl - \boldsymbol{U_2} * \boldsymbol{I}$

\# estimate water level
if wl_ok then wlh' = wll' = w
else $wlh' = wlh - min\_steam\_water\_est(srl') + \boldsymbol{P} * pumps * \boldsymbol{S} + \boldsymbol{P} * pumps' * (\boldsymbol{I} - \boldsymbol{S})$
$wll' = wll - (srh' + \boldsymbol{U_2}*\boldsymbol{I}/2)*\boldsymbol{I} + \boldsymbol{P} * pumps * \boldsymbol{S} + \boldsymbol{P} * pumps' * (\boldsymbol{I} - \boldsymbol{S})$

\# safety checks
if $srh' \geq \boldsymbol{W} - \boldsymbol{U_1} * \boldsymbol{I}$ *or*
$wlh' \geq \boldsymbol{M_2} - \boldsymbol{P} *(pumps' * \boldsymbol{S} + (max\_pumps\_after\_set) * (\boldsymbol{I} - \boldsymbol{S})) + min\_steam\_water(srl)$ *or*
$wll' \leq \boldsymbol{M_1} + \boldsymbol{P} *(pumps' * \boldsymbol{S} + (min\_pumps\_after\_set) * (\boldsymbol{I} - \boldsymbol{S})) - max\_steam\_water(srh)$
then stopmode' = ***true***
*else stopmode' = {**true, false**} arbitrary*

Internal Actions

bad

Precondition:

true

Effect:

*sr_ok' = {**true**, **false**} arbitrary*
*wl_ok' = {**true**, **false**} arbitrary*

controller

Precondition:

true

Effect:

$0 \leq px' \leq \boldsymbol{\#pumps}$

ν(Δt)

Precondition:

true

Effect:

$now' = now + \Delta t$

Output Action

actuator (e_stop, pset)

Precondition:

do_output = ***true***
pset = px
e_stop = stopmode

Effect:

do_output' = ***false***

The controller model that allows sensor faults has the same structure as the simple controller. An additional action ***bad*** tell the controller whether a sensor has failed. The fault model allows arbitrary combinations of sensor break downs and fast or slow repairs. The ***sensor*** action expresses the strategy of the controller to cope with sensor faults. Basically, the strategy is to calculate an upper and lower limit for the missing value of the failed sensor, using its last recent value and the remaining sensor values. Even in the case that both sensors break, the controller still may allow the operation of the boiler and guarantee safety. In this respect, our controller definition is better than the one suggested in [AS96], since he suggests to shut down the boiler system whenever both steam rate and water level sensors fail.

The various operational modes (normal, degraded and rescue) as specified in [AS96] can be inferred from the variables *sr_ok*, *wl_ok* and the difference between *pumps* and *px*. In our model, these modes are not relevant to the safety of the boiler system and have therefore been ignored.

7.3 Proving the Safety Properties by Simulation Mapping

After composing the steam boiler automaton with the new fault-tolerant controller, we have to prove that the safety properties are satisfied in the new model.

We use a **Simulation Mapping** for proving that one Timed Automaton "implements" another. This technique shows that all possible traces* of the new automata are included in the traces of the already proven model. Therefore, all safety properties involving the states of the steam boiler with the simple controller are valid for the system with the fault-tolerant controller, too. A Simulation Mapping is most useful to show that an implementation actually preserves properties of the specification. This method can be applied repeatedly to get from a very abstract model, which is proven to fulfill the required properties, to a detailed implementation (maybe even the final implementation). Like invariants, the Simulation Mappings involve time deadline information, in particular, they include inequalities between time deadlines. Therefore, they are suitable for showing timing properties, too.

We apply a Simulation Mapping from states of the steam boiler system with the fault-tolerant controller (in short "fault-tolerant controller system") to the system with the simple controller ("simple controller system"). Appendix B of the CD-ROM version contains a formal definitions of the Simulation Mapping technique and the correctness properties it guarantees.

7.3.1 Simulation Relation

Theorem 3: The relation *f* as defined below is a Simulation Mapping from the states of the fault-tolerant controller system to the states of the simple controller system.

Let *s* denote a state of the simple controller system and *i* denote a state of the fault-tolerant controller system. We define *s* and *i* to be related by the relation *f* provided that:

1) *i.Boiler = s.Boiler*†
2) *i.do_output = s.do_output, s.px = i.px, s.pumps = i.pumps, s.now = i.now*
3) *i.srl* $\leq$ *s.sr* $\leq$ *i.srh*
4) *i.wll* $\leq$ *s.wl* $\leq$ *i.wlh*
5) *s.stopmode = i.stopmode*

Proof sketch: Let *i* lead to *i'* via action *a* in the fault-tolerant controller. We must find an *s'* such that *s' f i'* and there exists an execution fragment from *s* to *s'* with the same trace as *a*. Usually, we break by cases on the type of *a*. In the initial state *f* is fulfilled. For this proof it remains to show the case for the sensor action because all other actions are identical in the specification and implementation. It remains to show that there is an equivalent sensor step enabled in *s*, and *s'* relates to *i'* following the definition of *f*. In particular, we must show the three conditions in the definition of a Simulation Mapping in Appendix B in the CD-ROM version. The first condition, preservation of the *now* value, is immediate from the

* The exact meaning of "traces" is defined in Appendix A on the CD-ROM.
† This relation expresses that the entire boiler state is preserved.

definition of *f*. The second condition, correspondence of the start states, is also immediate, because *f* is fulfilled between the start states. The interesting condition is the induction step. As before, the detailed proof can be found in the full version of this text. The proof depends on Lemma 1.3&6, $s.now \leq s.read - \boldsymbol{I} + \boldsymbol{S}$ *or* $s.set = s.read + \boldsymbol{S}$ (Lemma 6), $-\boldsymbol{U_2}*(\boldsymbol{I} + s.now - s.read) \leq s.v - s.sr \leq \boldsymbol{U_1}*(\boldsymbol{I} + s.now - s.read)$ (Lemma 15) and *if* $s.do\_output = \textbf{\textit{false}}$ *then* $ps - \delta_{HIGH}(s.sr, s.v, t) \leq s.q - s.wl \leq ps - \delta_{LOW}(s.sr, s.v, t)$ with $ps =$ *if* $s.set = s.read + \boldsymbol{S} - \boldsymbol{I}$ *then* $\boldsymbol{P} * s.pumps * t$ *else* $\boldsymbol{P} * (s.pumps * \boldsymbol{S} + s.pr * (t - \boldsymbol{S}))$ and $t = (\boldsymbol{I} + s.now - s.read)$ (Lemma 16).

This Simulation Mapping maps every reachable state of the boiler system with the fault-tolerant controller to a corresponding reachable state in the system with the simple controller by the relation *f*. Therefore, the safety properties involving the states of the specification (simple controller) are valid for the implementation (fault-tolerant controller), too. Thus, we have shown that the steam boiler system with the fault-tolerant controller satisfies the required safety properties.

8 Conclusion

We have applied a formal method based on Timed Automata, invariant assertions and Simulation Mappings to the steam boiler model, and verified that our controller fulfills the required safety properties. In doing so we have made it possible to compare our techniques to other approaches.

Summarizing, the *Timed Automata*, *composition* and *Simulation Mapping* techniques present an excellent combination for system analysis. The main advantage of Timed Automata is their flexibility in modeling a hybrid system. Timed Automata allow us to combine a continuous environment that is fairly unpredictable over time with a discrete control system such as a computer. The composition and Simulation Mapping techniques supplement this specification tool for formal verification, for more flexibility in how to search for a solution and for the reuse of already gained knowledge. The composition technique lets you combine different automata and scale incrementally solutions from smaller problems to more complex ones. The Simulation Mapping technique provides a consistent transition between different abstraction layers.

This method seems to scale better than other formal verification techniques because of the possibility of applying this method to different abstraction layers, and applying various decomposition techniques [Wei96]. A Simulation Mapping can be used to prove that two abstraction layers preserve certain properties. Decomposition techniques provide modular and incremental verification. For instance, suppose that you have proved that a certain implementation of a shared register provides mutual exclusion. The automaton model together with already proved properties may then be composed into a bigger application without having to prove the mutual exclusion property again.

Constructing the proofs, though not difficult, requires significant work. The hardest parts were getting the details of the models right and finding the right invariants. Unfortunately, this seems to be an art rather than an automatic procedure. Nevertheless, our experience in this paper and others (e.g., [Hei94]) shows that this art is easily learnable even for application engineers. The techniques are very systematic and understandable. The description allows for much flexibility and is very powerful in describing the possible progression of a system.

The actual proofs of the invariants were tedious but routine work. Much work can be avoided by proving the required properties on a general model and using *Simulation Mappings* for more specialized models. Moreover, the characteristics of these techniques make them amenable for mechanical generation and verification of proofs. Related to this, we are currently considering the use of automatic provers such as Larch [Soe93] or PVS [Sha93] with the described techniques.

The only major disadvantage we encountered while working with Timed Automata and the Simulation Mapping technique is that we could not gain any information or any measurement towards the optimality of parameters of a solution. Although our controllers preserve provable safety, there are obviously better implementations. For example, on a steam rate sensor failure, the steam rate estimation could take into account the amount of water which has evaporated since the last sensor reading. Moreover, we like to note that more of the reality could be modeled formally with a more relaxed pump failure model and diverse pump controller algorithms. The latter might lead to interesting performance comparisons and tighter parameters such as the distance between $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$.

Future work includes applying this method to larger and more complex examples, and developing the appropriate computer assistance for carrying out and checking the proofs. On-going research in our group shows that the timed-automata method provides high potential for automating the generation of the proofs [Sha93], [Arc96].

Acknowledgments

We thank Anya Pogosyants and Roberto Segala for several useful comments as well as Angelika Leeb and Dave Evans for comments and proofreading.

References

[AS96] Abrial, J.-R.: A B-solution for the steam-boiler problem. Contains: Steam-boiler control specification problem for the meeting Methods for Semantics and Specification, Dagstuhl; See chapter AS in this LNCS volume.

[Arc96] Archer, M.; Heitmeyer, C.: Mechanical Verification of Timed Automata: A Case Study, To appear in the proceedings of RTAS, 1996

[Cle93] Cleaveland, R.; Parrow, J.; Steffen, B.: The concurrency workbench: A semantics-based tool for verification of concurrent systems. ACM Trans. on Prog. Lang. and Sys., 15(1):36-72, Jan. 1993

[Hei93] Heitmeyer, C.; Jeffords, R.; Labaw, B.: A benchmark for comparing different approaches for specifying and verifying real-time systems. In Proc., 10th Intern Workshop on Real-Time Operating Systems and Software, May, 1993

[Hei94] Heitmeyer, C.; Lynch, N.: The Generalized Railroad Crossing: A Case Study in Formal Verification of Real-Time Systems. In Proceedings of the 15th IEEE Real-Time Systems Symposium, San Juan, Puerto Rico, IEEE Computer Society Press, pages 120 -131, December 1994

[Hoa93] Hoare, C.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs, NJ, 1985

[Jah86] Jahanian, F.; Mok, A.: Safety analysis of timing properties in real-time systems. IEEE Trans. Software Engineering, SE-12(9), Sep. 1986

[Lyn91] Lynch, N.; Vaandrager, F.: Forward and backward simulations for timing-based systems. In Proceedings for REX Workshop: Real-Time: Theory in Practice, vol. 600 of Lecture Notes in Computer Science, p. 397-446, Mook, Netherlands, Springer-Verlag, June 1991

[Lyn94] Lynch, N.: Simulation Techniques for Proving Properties of Real-time Systems, In REX Workshop '93, Lecture Notes in Computer Science, Mook, the Netherlands, Springer Verlag, 1994

[Soe93] Soegaard-Anderson, J.; Garland, S.; Guttag, J.; Lynch, N.; Pogosyants, A.: Computer-assisted simulation proofs, In Costas Courcoubetis, Computer-Aided Verification: 5th International Conference, (CAV'93 Elounda, Greece, June/July 1993, Lecture Notes in Computer Science 697, p. 305-319, Springer-Verlag, 1993

[Seg94] Segala, R.; Lynch, N.: Probabilistic Simulations for Probabilistic Processes. In J. Parrow, Editor, Proceedings of CONCUR 94, Lecture Notes in Computer Science, volume 836, pages 481-496, Uppsala, Sweden, August 1994.

[Sha93] Shankar, N.: Verification of real-time systems using PVS. in Proc. Computer Aided Verification (CAV'93), pages 280-291. Springer-Verlag 1993

[Wei96] Weinberg, H.: Correctness of a Vehicle Control System: A Case Study, Master's Thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, 1996