Formal Hazard Analysis of Hybrid Systems in cTLA*

Peter Herrmann and Heiko Krumm
Universität Dortmund, Fachbereich Informatik, D-44221 Dortmund
{herrmann|krumm}@ls4.cs.uni-dortmund.de

Abstract

Hybrid systems like computer-controlled chemical plants are typical safety critical distributed systems. In present practice, the safety of hybrid systems is guaranteed by hazard analysis which is performed according to procedures (e.g., HazOp) where experts discuss a series of informal argumentations. Each argumentation considers a specific required system property. Formal property proofs can increase the reliability. They, however, have often to deal with very complex hybrid systems. Therefore, methods are needed which structure and decompose formal verification tasks into manageable substasks. With respect to this, our approach achieves a relatively direct translation of informal argumentations into formal proofs. Since the informal argumentations mostly do not refer to the system as a whole but do only address specific parts and aspects, the formal proofs also can deal with partial, less complex system models. In result, even very complex systems can be verified in well-manageable subtasks. The direct translation is supported by the characteristics of the specification technique applied. The temporal logic based technique cTLA supports the modular description of hybrid process systems. In particular, one can model a system as a composition of behavior constraints. Properties which are implied by a subsystem of constraints also are properties of the system as a whole. Therefore a subsystem can correspond to the parts and aspects addressed by an informal argumentation. We outline cTLA and introduce the formalization of hazard analysis argumentations by means of an hybrid example system. Additionally, we sketch a framework of specification modules and theorems which supports the formal hazard analysis of hybrid systems.

1. Introduction

Today, HazOps (hazard and operability studies) [20] are an established method in hazard analysis. In a systematic way, teams of experts examine descriptions of hybrid systems in order to detect subtle system faults and sources of danger. Systems are reduced into subsystems which are examined for faults. Furthermore, the teams detect the sources of a fault, predict possible consequences, and develop counter-measures. Various approaches to support the normally time-consuming and expensive HazOps have been developed. Besides expert systems assisting the examination of technical plants (cf. [5, 8, 22, 25]), mainly model-based approaches were investigated. For the plants abstract models are created, where the continuous flows of the plant are specified by a discrete set of states. The models are specified by qualitative equations [3, 27], Petri nets [23], or temporal logical formulas [21]. Simulation [6, 27], state space exploration [24], fault trees [7], and symbolic model checking [21] are used to detect faults in the models of the technical plants. Our approach is based on formal modelling and temporal logic specifications, too. In particular, we apply symbolic reasoning which is directly oriented at informal argumentations.

The informal argumentations of traditional hazard analysis procedures mostly supply well-understandable proof strategies. Moreover, very often, they induce an appropriate decomposition of the system analysis into a series of property proofs which only refer to smaller parts of the system. The reliability of the hazard analysis, however, would be enhanced by formal proofs, which on the other hand would require the costly development of complex proofs. Therefore approaches are of interest, which support the relatively direct translation of informal argumentations into formal proofs. They promise to provide for reliable formal proofs under reduced complexity and costs, since they consider subsystems only and already existing argumentations guide their development.

This objective demands for the application of a suitable formal specification and verification technique. Since the informal argumentations mainly consider the structure of a plant, i.e., its structuring into components and the coupling of the components, the formal technique has to support compositional specifications, which define a system accordingly by its set of components and their connections

*This work was funded by the German research foundation DFG.

(cf. [2, 14]). The components themselves are described by modular component specifications. Nevertheless, when proving interesting system properties, most compositional specifications have to be considered as a whole. It is not possible to reduce the scope of the proof to a small subsystem since the verification technique in general does not support the deduction of system properties from subsystems or since the special structuring of the system specification is not adequate. With respect to the first point, in system proofs many techniques abstract from component internals. They need, however, at least an abstract specification for each component, since in general each component of a system can influence all relevant system properties. With respect to the second point, many compositional specifications reflect the physical implementation structure of a system. The components of the specification correspond directly to real plant components which indeed have many different aspects and may influence a system in various ways. Thus, an interesting system property may depend on the proper cooperation of nearly all components. Our approach therefore applies so-called constraint-oriented specification structures (cf. [26]). The components of a specification do not directly model physical system components as a whole. Instead, specification components describe constraints which refer to single aspects of the behavior of physical parts. Thus, specifications have a finer structuring which is oriented at the logical connections in the system. Moreover, we combine constraint orientation with superposition (cf. [4]). The composition operation of our specification technique cTLA is equivalent to the consistent logical conjunction of component descriptions. Therefore, all relevant properties of components or subsystems are also properties of the system as a whole. In combination, one can restrict the proof of a system property to the consideration of a very small subsystem, if a suitable constraint-structured system specification exists.

The name of the specification technique cTLA [11, 12] stands for compositional TLA. cTLA is an extension of the Temporal Logic of Actions [19]. As in TLA, systems are modelled by state transition systems. A specification has the character of a linear time temporal logic formula which describes relevant properties of the system. Moreover, cTLA recognizes the TLA-extensions to realtime systems [1] and hybrid systems [18]. In difference to TLA, cTLA supports the explicit notion of processes. Processes act as modular specification components and can represent implementation parts as well as logical constraints. As in the ISO/OSI specification language LOTOS, a set of processes interact in a rendezvous-like way by performing actions jointly, and data parameters of the actions can model the communication of values between processes. Thus, cTLA processes are related to objects in DisCo [17]. They encapsulate private state components and interact via joint actions.

A first attempt to specify and verify hybrid systems based on cTLA was introduced in [10]. In the meanwhile, we studied a series of other examples and identified reusable specification and verification elements. In this paper, we describe the formal hazard analysis of an example system by using the pre-defined elements. These elements also serve as a basis for a framework consisting of libraries of specification components and theorems which is introduced below.

In the remainder of this paper we outline cTLA first. Thereafter we introduce the hybrid example system and describe its compositional specification. We sketch the informal argumentation of a safety property and outline the corresponding formal structured verification. Finally, we introduce the framework for hazard analysis and outline the application of a framework theorem for the example verification.

2. cTLA

The cTLA [11, 12] notation refers to TLA [19] which describes the safety and liveness properties of state transition systems by means of canonical formulas. The syntax of cTLA is oriented at programming languages. Moreover, cTLA omits the canonical parts of TLA formulas. Therefore the formula character of cTLA-specifications is not directly visible. Nevertheless, each specification corresponds to a TLA-formula. In difference to TLA, cTLA introduces the notion of processes. A specification is structured into modular definitions of process types. An instantiation of a process type forms a process which either may have the form of a simple process or that of a process composition. Simple processes directly refer to state transition systems.

Fig. 1 shows the example of the very simple process type *ReactOnAlarm*. The header declares the type name and generic module parameters (e.g., `initialstate`). The

```
React on a sensor alarm by switching an actor
PROCESS ReactOnAlarm
          (initialstate : {"ready","alarm"})
            controller initially active ?
BODY
  VARIABLES
    vastate : {"ready","alarm"}; actual alarm state
  INIT ≙ vastate = initialstate;
  ACTIONS
    alarm ≙ vastate′ = "alarm"; signal alarm
    switch ≙ vastate = "alarm" ∧
               vastate′ = "ready";
      react by switching an actor
END
```

Figure 1. Process Type *ReactOnAlarm*

body defines the state transition system which corresponds to an instantiation of the type. The state space is defined by the state variable `vastate`. The initial condition `INIT` is a condition on state variables and defines the set of starting states. Finally, the body declares actions each describing a set of state transitions (e.g., `alarm`, `switch`). An action can have action parameters. It is a condition on parameters and state variables where the state variables can also occur in so-called primed form (e.g., `vastate'`). The primed variables reference the successor state of a transition. The union of the actions forms the next state relation of the process. In the course of time, a process may perform action steps (i.e., it changes its state in accordance with an action) or stuttering steps (i.e., it does not change its state while the environment performs a state transition).

Processes of a form like *ReactOnAlarm* describe safety properties. Liveness properties are described by additional fairness assumptions on actions, e.g., `WF: switch` states that the action `switch` has to be performed weak fairly. Fairness forces the activity of a process. A fair action cannot be enabled for an infinite period of time without being executed. Weak fairness forces the execution if the action would be enabled incessantly otherwise. Strong fairness forces the execution even if the action is sometimes disabled. Unlike TLA, cTLA provides for conditional fairness assumptions to ensure the consistency of process compositions. A fairness statement refers to periods of time where an action is enabled as well as the environment of the process is ready to tolerate the action.

According to [1], realtime is represented by means of a real-valued state variable `now` which is incremented lively by a clock action `tick`. Unlike other variables which are private to exactly one process, `now` can be read by all processes of a system. The clock variable forms the basis for the definition of realtime and continuous properties. Additional constructs describe activity retarding and activity forcing realtime constraints. In accordance to [18], one can specify minimum waiting times and maximum reaction times for actions. Comparable to the distinction between weak and strong fairness, one may refer to volatile and to persistent enabling periods of an action. Also, the maximum reaction time construct is conditional. It forces an action only with respect to periods where the action is enabled and the environment does not block it. For instance, Fig. 2 shows the process type *ReactMaxTime*. It declares an action `switch` with a volatile maximum reaction time modelled by the generic module parameter `maxtime`.

Continuous properties of a process are expressed by means of an action with the special name `CONT`. All `CONT`-actions of all processes of a system and the $tick$-action of the clock are assumed to occur simultaneously. Thus, the series of `CONT`-steps of the system approximates the continuous behaviour (cf. [18]). Usually, the `CONT`-actions contain difference equations for continuous state variables. The time difference corresponding to an execution of `CONT` is expressed by `now'-now`. Action parameters model the inputs and outputs of continuous processes. As an example Fig. 3 shows the process *VesselVolume* declaring a continuous real-valued state variable `vvolume`. The `CONT`-action declares the continuous inputs and outputs and lists the difference equation for `vvolume`.

Several constructs stating safety, fairness, realtime, and continuous properties may be contained in the same process type definition. Thus, one can specify all relevant properties of a hybrid system component by one process. However, to support a fine-grained constraint-oriented system structure, we mainly use process types which concentrate on single properties of a single sort.

Systems and subsystems are described as compositions of concurrent processes. Each process encapsulates its variables and can change its state by atomic executions of its

```
Maximum time for a controller to switch an actor in [sec]
PROCESS ReactMaxTime ( maxtime : real )
                              maximum time
BODY
  ACTIONS
    switch;
  V MAX TIME switch : maxtime;
END
```

Figure 2. Process Type *ReactMaxTime*

```
Volume of fluid in a vessel in [m^3]
PROCESS VesselVolume ( capacity : real ;
                       maximum volume in vessel
                       initvolume : real )
                       initial volume in vessel
BODY
  VARIABLES
    vvolume : real; actual volume
  INIT ≙ vvolume = initvolume;
  ACTIONS
    CONT (INPUT  inflow, outflow : real;
                 inflow to and outflow from vessel
          OUTPUT volume : real) ≙
                 actual volume in vessel
      vvolume' =
        Max(Min(vvolume +
                (inflow - outflow) ·
                (now' - now),
                capacity),0) ∧
      volume = vvolume;
END
```

Figure 3. Process Type *VesselVolume*

actions. The system state is the vector of the state variables of the processes. State transitions of the system correspond to simultaneous process actions and process stuttering steps. Each process performs either exactly one action or a stuttering step. Thus, the system actions can be defined by conjunctions of process actions and process stuttering steps. Consequently, concurrency is modelled by interleaving and the coupling of processes corresponds to joint actions. A system specification at first declares the processes of the system as instantiations of process types. Thereafter it describes the coupling of the processes by the definition of the actions of the systems. An example of a system description is outlined in Fig. 6 (Sec. 6).

3. Example

As an example we use a chemical plant [15] employed as a benchmark system in the working group on discrete control of the German Society for Measurement and Automation (GMA). A discontinuous process produces a sodium chloride solution of a desired concentration by means of mixing a higher concentrated sodium chloride solution with water. After its utilization the solution is separated into its original ingredients by vaporization again. The system is sketched in Fig. 4. From the vessel B1 highly concentrated sodium chloride solution is introduced into vessel B3. To dilute the solution, water is introduced from vessel B2 to B3 afterwards. When the desired concentration is achieved, the solution is stored in vessel B4 for further usage.

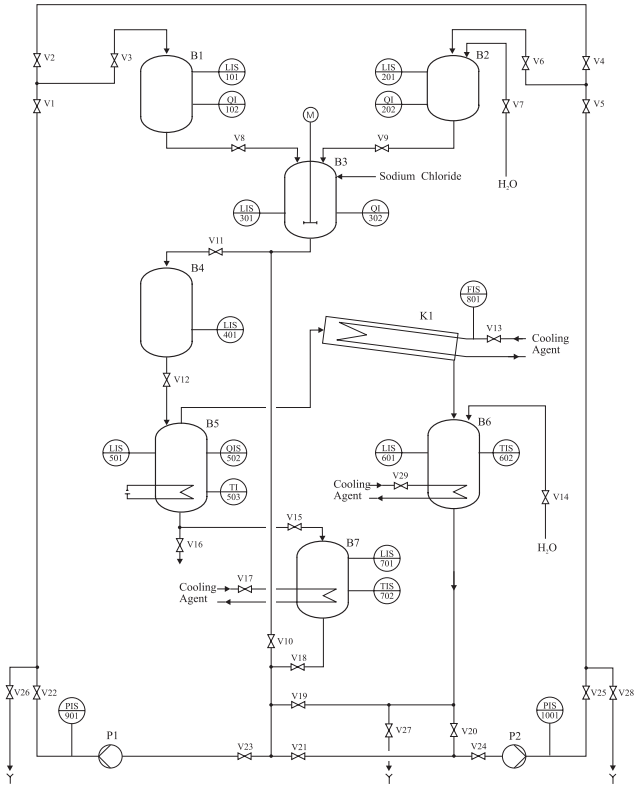

Figure 4. Sketch of the example system

Description	Function
BP2	Fill up vessel B2 with water
BP3	Produce sodium chloride solution of the desired concentration in B3
BP3K	Produce the highly concentrated sodium chloride solution in B3
BP3U	Pump the highly concentrated sodium chloride solution from B3 to B1 by P1
BP5	Vaporize sodium chlorid solution in B5
BP6	Cool the water in B6
BP6A	Empty water from B6
BP6S	Pump water from B6 to B2 by P1
BP6U	Pump water from B6 to B2 by P2
BP7	Cool sodium chloride solution in B7
BP7U	Pump solution from B7 to B1 by P1
SP1	Clean ring pipe with water from B2
SP2	Clean ring pipe with water from B6

Figure 5. Batch processes controlling the example system

To separate the solution, it is introduced from B4 to vessel B5, where as much water as necessary to produce the original highly concentrated solution is vaporized. The steam condenses in the heat exchanger K1 and drops into vessel B6. In B6 the water is cooled to the temperature of the environment and, afterwards, pumped back to the vessel B2 by the pump P2. The highly concentrated sodium chloride solution, produced by the vaporization, is introduced into the vessel B7 where it is cooled, too. Finally, the cooled solution is pumped back to the vessel B1 by means of the pump B1.

The process outlined above and additional processes as the initial production of the sodium chloride solution in B3 or the cleaning of the plant are controlled by the batch processes [16] listed in Fig. 5. For instance, the batch process BP5 controls the separation of water and solution in vessel B5. First, it triggers the introduction of solution from vessel B4 to B5. When B5 is filled, the process starts the cooling of vessel K1 and afterwards the heating of B5. During the vaporization it constantly checks the cooling. After a failure of the cooling the heating is switched off within 0.1 seconds. The heating and cooling are switched off when the desired concentration of the solution is reached. The batch process introduces the solution into vessel B7 and terminates after-

wards. If the solution in B5 falls below a minimum during the vaporiziation, the controller switches the heating and cooling off, too. However, it aborts without introducing the water into B7.

4. Specification

Our approach is based on a fine-grained constraint-oriented structuring of the system specification. Therefore, we develop a specification which is a composition of modular constraint processes each modelling a single constraint of a physical system component. For instance, the filling of a vessel in our example system can be viewed by the following constraints: the masses of water, steam, and dissolved sodium chloride in the vessel (cf. process type *VesselVolume* in Fig. 3), the temperature of the water/steam in the vessel, and the pressure in the vessel. Thus, we specify the vessel by five different cTLA-processes each describing only one constraint.

To model a pump, a valve, or an electrical heating/cooling in our example system, we use a hybrid process modelling as well the interface to the discrete control software as the actual actuator setting: the nominal amount of water to be pumped, the diameter of the valve, resp. the amount of power passed to or taken from the heated/cooled vessel.

The sensors of the example system detect the concentration of sodium chloride solution, the temperature of water, the amount of water in a vessel, the volume stream in a pipe, and the pressure of a pump. A sensor is specified by three processes: one process models a signal to the controller when it detects an alarm state. A second process models that the current value is measured on request of the controller. Since the reaction on an alarm state has to be performed within a certain period of time, we need a third process which guarantees a maximum reaction time.

The physical characteristics of the pipes linking the different vessels, actuators, and sensors are not modelled directly. Instead, we use processes describing the volumes of water, steam, and sodium chloride running from one vessel to another. To specify the flow of water and dissolved sodium chloride between two vessels of a different altitude, we use three processes. Two processes model the volumes of water and sodium chloride flowing from the upper vessel to the lower vessel in a certain amount of time. The volumes are calculated according to the flow formula of Torricelli, where the diameter of the pipe corresponds to the lowest diameter of all valves and pipes within the link. A third process specifies the temperature of the water flowing into the lower vessel. Similar processes are used to model pipes through which water is pumped. The flow depends on the amount of water pumped per time-unit. The flow of water or steam between two vessels due to different pressures is modelled by two processes according to the energy law of Bernoulli. The flow of water or steam and the temperature are described by two separate processes.

The batch process recipes controlling our example system are modelled by separate cTLA-processes, too. Moreover, since some batch processes execute a series of different tasks, one can split them into different cTLA-processes each modelling one task only. For instance, the batch process BP5 can be split into a series of cTLA-processes since it controls different phases: the introduction of sodium chloride solution from vessel B4 to vessel B5, the vaporization of water until a solution of the desired concentration is produced, and, finally, the introduction of the produced solution from vessel B5 to vessel B7. Moreover, in each phase certain system actions have to be linked with additional constraints. For example, an instance of the type *ReactOnAlarm* in Fig. 1 specifies that in the vaporization phase BP5 switches off the heating in vessel B5 after a failure of the cooling in K1. Since in our approach, safety, liveness, and realtime constraints are modelled independently, we need additional cTLA-processes describing that the commands of the batch processes are executed fairly or within certain time limits (e.g., an instance of process type *ReactMaxTime* — see Fig. 2 — guarantees that the heating of B5 is not switched off too late after a failure of the cooling in K1).

The complete specification of the example system is available in the WWW (`http://ls4-www.cs.uni-dortmund.de/RVS/P-HYSYS/`).

5. Informal Argumentation

A typical problem of the hazard analysis of our example is the question, whether the excess pressure in the vessels B5 and K1 can exceed a critical limit of $0.2\ Bar$ during the separation of the sodium chloride solution. Vessel B5 contains a heating in order to vaporize water. The steam produced here flows to vessel K1. In K1, the steam shall condense to water which drops into vessel B6.

The usual informal discussion of this question considers the failure of the cooling in K1 as a possible reason which can cause an increase of the excess pressure over the critical limit. The system may run too long in a critical state where the heating in B5 works but the cooling in K1 fails. Then only a part of the steam coming from B5 condenses in K1 and the pressure in K1 as well in B5 rises.

The argumentation that the system does not run too long in the critical state will be performed in three steps. First, we determine the maximum difference Δp_{max} between the pressure in B5 and K1. Based on Δp_{max} we calculate in a second step the maximum time t_{max} the system may be in the critical state without the pressure in one of the vessels exceeding the limit of $0.2\ Bar$. Finally, we check that the discrete controller guarantees that the example system does not remain in the critical state longer than t_{max}.

In order to calculate the maximum pressure difference Δp_{max} between B5 and K1, we take into consideration that after switching on the heating the pressure in B5 increases until the mass of steam vaporized equals the mass of steam running to K1. The maximum mass of steam $\dot{m}_{vap}$ created by vaporization depends on the maximum power of the heating and is equal to $2.66 \cdot 10^{-3} \frac{kg}{s}$. Since, the mass of steam running to K1 does not exceed this value, we can now calculate the maximum pressure difference Δp_{max} between B5 to K1 on the basis of the law of Bernoulli:

$$\Delta p_{max} = \frac{\dot{m}_{vap}^2}{2\ \varrho\ A^2} + g\ z\ \varrho$$

Since the density of steam ϱ is equal to $0.59 \frac{kg}{m^3}$, the diameter A of the pipe linking B5 with K1 to $28.27 \cdot 10^{-6} m^2$, the difference z between the altitudes of B5 and K1 to $0.3\ m$, and the acceleration of gravity g to $9.81 \frac{m}{s^2}$, the value of Δp_{max} equals to the value $0.07504\ Bar$. Consequently, the excess pressure in B5 does not exceed the critical limit if the pressure in K1 equals to the athmosperic pressure.

Now, we will calculate the maximum time t_{max}, the system may be in the critical state without the pressure in one of the vessels exceeding the limit of $0.2\ Bar$. Above, we calculated that the pressure in B5 is at most by Δp_{max} larger than that of K1. To guarantee that it does not climb over the limit, the pressure $p_{K1_{max}}$ in K1 must not exceed $0.12496\ Bar$. Next, we calculate the increase of pressure $\dot{p}_{K1}$ in vessel K1 in the worst case that the cooling fails completely and a maximum mass of $\dot{m}_{vap} = 2.66 \cdot 10^{-3} \frac{kg}{s}$ steam runs from B5 into K1. $\dot{p}_{K1}$ is calculated on the basis of the thermal state equation:

$$\dot{p}_{K1} = \frac{\dot{m}_{vap}\ R_i\ Z\ T}{V}$$

The individual gas constant R_i of steam is equal to the value $461.5 \frac{J}{kgK}$, the temperature T of the steam to $373.16K$, and the volume of vessel K1 to $7.854 \cdot 10^{-3} m^3$. Assuming a pressure less than $2\ Bar$ the real gas factor equals to 0.98. Thus, the value of $\dot{p}_{K1}$ is $0.5719 \frac{Bar}{s}$. The maximum time t_{max} is calculated by $\frac{p_{K1_{max}}}{\dot{p}_{K1}}$ and equals to $0.218\ s$.

In the last step we check that the disrete controller guarantees that the example system remains at most for $t_{max} = 0.218\ s$ in the critical state. In particular we outline, that either the system is in a non-critical state where the heating is switched off or the cooling works, or that the period of time, the system is in a critical state, does not exceed t_{max}. The controller consists of the batch processes listed in Fig. 5. However, we have to examine the batch process BP5 (cf. Sec. 3) only since in the other batch processes the heating is always switched off. Moreover, only one of the activities controlled by BP5 is critical, since the heating is switched off during the introduction of solution from vessel B4 to B5 and from B5 to B7. Therefore, only the vaporization activity has to be analysed further. Since, if the cooling works properly, the system state is non-critical, we concentrate on cooling failures. The batch process BP5 checks the cooling constantly and reacts on a failure within $0.1\ s$ by switching off the heating. Thus, the system remains at most for $0.1\ s$ in the critical state which is shorter than the maximum time $t_{max} = 0.218\ s$ tolerated by the continuous components.

6. Formal Verification

In order to analyze questions of hazard analysis formally, we translate informal argumentations straightforwardly into formal cTLA-based proofs. As an example, we refer to the informal argumentation of Sec. 5 and verify the last argumentation step that the discrete controller prevents the system to last in the critical state longer than t_{max}.

As mentioned in the introduction, we do not need the specification of the whole system to perform the proof of a property but only a subsystem specification containing only constraints relevant to the proof. We look at the informal argumentation for the property (cf. Sec. 5) and construct a subsystem *ReactOnCoolingFailure* which contains exactly these constraint processes modelling relevant assumptions of the argumentation. *ReactOnCoolingFailure* is listed in Fig. 6[1]. It consists of eight cTLA-processes. *B5Heat* models the heating with a maximum power of $6000\ Watt$. By *K1Power* we describe that, when the cooling fails, the cooling power decreases at most by $20\ Watt/s$. The sensor FIS801 which controls the cooling power is modelled by three constraints. *FIS801S* specifies that the sensor sends a signal to the discrete controller if the cooling power falls below a limit of $6005\ Watt$. *FIS801T* describes that this signal is sent within 0.1 seconds. *FIS801R* models that the sensor signals also the recovery of the cooling if the cooling power climbs above $6005\ Watt$ again. The relevant components of batch process BP5 are specified by three constraints. *BP5ROA* (cf. Fig. 1) models that the controller reacts on a signal from FIS801 indicating a cooling failure by switching off the heating of vessel B5. By *BP5ROAT* (cf. Fig. 2) we specifiy that the heating is switched off within 0.1 seconds. The constraint *BP5BH* guarantees that the heating is not switched on during the failure of the cooling.

As pointed out in the informal argumentation, the system has always to fulfill the condition that either the heating is switched off, the cooling supplies sufficient power, or the system lasts in the critical state shorter than $t_{max} = 0.218\,s$. We describe this invariant condition by the formula I:

$$I \triangleq \texttt{B5Heat.vstate} = \texttt{"off"} \vee \texttt{K1power.vpower} \geq 6000 \vee t_{fail} < 0.218$$

[1] To simplify the specification, we partly omitted to list stuttering processes in the description of the actions.

```
PROCESS BP5ReactOnCoolingFailure
  PROCESSES
  Heating B5
    B5Heat : Heating (6000, 20, 20);
    Maximum power: 6000 Watt, power inc/dec 20 Watt/sec
  Cooling K1
    K1Power : PowMaxDecrease (20);
      Constraint: cooling power decreases max. by 20 Watt/sec
  Sensor FIS801 to detect failure of cooling for vessel K1
    FIS801S : SenseMin (6005);
      Signal minimum condense power of 6005 Watt
    FIS801T : SenseMaxTime (0.1);
      Signal of cooling failurewithin 0.1 sec!
    FIS801R : SenseMax (6005);
      signal recovery of the cooling
  Batch process BP5
    BP5ROA : ReactOnAlarm ("ready");
      Switching off heating after a cooling failure
    BP5ROAT : ReactMaxTime (0.1);
      Switch off heating after a cooling failure within 0.1 sec
    BP5BH : BlockOnAlarm ("block");
      Block heating until the cooling is recovered

  ACTIONS
    CONT (OUTPUT B5Hpow, K1CPow) ≙
      Continuous flow: Output parameters B5Hpow, K1CPow:
                       heating/cooling powers
      B5Heat.CONT(; B5Hpow) ∧
      K1Power.CONT (; K1Cpow) ∧
      FIS801S.CONT(K1Cpow ;) ∧
      FIS801R.CONT(K1Cpow ;) ∧ ...;
    CoolingFailure ≙
      Signal failure of the cooling
      FIS801S.alarm ∧ FIS801T.alarm ∧
      BP5ROA.alarm ∧ BP5BH.alarm ∧
      B5Heat.stutter ∧ ...;
    CoolingRecovery ≙
      Signal recovery of the cooling after a failure
      FIS801R.alarm ∧ BP5BH.release ∧ ...;
    HeatingOffAfterCoolingFailure ≙
      Switch off heating after cooling fails
      B5Heat.off ∧ B5ROA.switch ∧
      B5ROAT.switch ∧ ...;
    HeatingOffOtherReason ≙
      Switch off heating for other reasons
      (e.g., fluid dissolved; too few fluid in B5)
      B5Heat.off ∧ ...;
    HeatingOn ≙
      Switch on heating only if cooling did not fail
      or is recovered
      B5Heat.on ∧ BP5BH.switch ∧ ...;
END
```

Figure 6. Subsystem Specification *ReactOnCoolingFailure*

The variable `vstate` of the cTLA-process *B5Heat* specifies the state of the heating. The variable `vpower` of the process *K1power* describes the power supply of the cooling. t_{fail} is an auxiliary variable[2] which indicates the time elapsed since the failure of the cooling. The proof that the formula I always holds corresponds to the TLA-theorem

$$ReactOnCoolingFailure \Rightarrow \Box I$$

The temporal operator $\Box$ means that the subsystem *ReactOnCoolingFailure* implies that I is always valid, i.e., I is an invariant of *ReactOnCoolingFailure*.

Yet, we cannot prove I directly but need a stronger invariant $I_h \triangleq I_1 \wedge I_2$ where

$$\begin{aligned} I_1 \triangleq\ & \texttt{B5Heat.vstate} = \texttt{"off"} \vee \\ & \texttt{K1power.vpower} \geq 6005 \vee \\ & (\texttt{FIS801S.vsense} = \texttt{"alarm"} \wedge \\ & \ 6000 \leq \texttt{K1power.vpower} < 6005) \vee \\ & (\texttt{BP5ROA.vastate} = \texttt{"alarm"} \wedge t_{fail} < 0.1) \\ I_2 \triangleq\ & \texttt{BP5BH.vastate} = \texttt{"block"} \vee \\ & \texttt{K1power.vpower} \geq 6005 \vee \\ & (\texttt{FIS801S.vsense} = \texttt{"alarm"} \wedge \\ & \ 6000 \leq \texttt{K1power.vpower} < 6005) \end{aligned}$$

I_h reflects that in our informal argumentation we assumed some implicit conditions which have to be proven formally now. By I_1, which implies I, we describe the different states passed while the system reacts to a cooling failure. If the cooling power is below the sensor limit of $6005\ Watt$ but the heating still works, either the sensor FIS801 or the discrete controller are in an alarm state. If FIS801 is active but, yet, did not send a signal to the controller (`FIS801S.vsense` $=$ `"alarm"`), the cooling power is still higher than the maximum heating power of $6000\ Watt$. If the controller received a signal but, yet, did not switched off the heating (`BP5ROA.vastate` $=$ `"alarm"`), at most 0.1 seconds passed since the cooling power falling below the limit of $6000\ Watt$. I_2 describes that the heating is not switched on during the failure of the cooling. In particular, either the cooling power is above $6005\ Watt$, FIS801 is in the alarm state, or the controller blocks the heating (`BP5BH.vastate` $=$ `"block"`).

Since I_1 implies the invariant I, it is sufficient to prove the theorem

$$ReactOnCoolingFailure \Rightarrow \Box I_h$$

Typically invariants are verified in two steps: The invariant must hold initially and the actions of the system must not falsify the invariant.

First, we verify that I_h holds in the inital state. Since initially the heating is switched off (`B5Heat.vstate`

[2] Auxiliary variables are used for verification purposes only and do not influence the behaviour of a system.

$=$ "off") and the controller blocks the heating (BP5BH.vastate $=$ "block"), I_1 and I_2 hold.

Second, we prove that the actions of the subsystem specification *ReactOnCoolingFailure* keep I_h. Exemplarily we outline the proof that the action HeatingOn does not falsify I_h which corresponds to the formula $I_h \wedge$ HeatingOn $\Rightarrow I_h'$.

In $I_h' \triangleq I_1' \wedge I_2'$ the variables occur in the primed form (e.g., B5Heat.vstate'; cf. Sec. 2). It describes that the invariant holds in the state after performing the action HeatingOn.

To prove that I_1' holds, we must take into consideration that I_h (in particular I_2) is true before performing the action. HeatingOn is only enabled if the controller does not block the heating (BP5BH.vastate $=$ "ready"). Due to I_2 in this state either the cooling power is also sufficient (K1power.vpower ≥ 6005) or the sensor FIS801 is in an alarm state (FIS801S.vsense $=$ "alarm" $\wedge$ $6000 \leq$ K1power.vpower < 6005). Since the action does not alter the variables K1power.vpower and FIS801S.vsense,

$$\begin{array}{l}\texttt{K1power.vpower'} \geq 6005 \vee \\ (\texttt{FIS801S.vsense'} = \texttt{"alarm"} \wedge \\ \ 6000 \leq \texttt{K1power.vpower'} < 6005)\end{array}$$

also holds. This formula, however, implies I_1'. I_2' is trivially true since I_2 holds before the execution of the HeatingOn and this action does not alter the variables BP5BH.vastate, K1power.vpower, and FIS801.vsense used in I_2.

Likewise, we can prove that the other actions of the subsystem specification *ReactOnCoolingFailure* keep I_h. Therefore I_h and I are always true in the subsystem modelled by *ReactOnCoolingFailure* and the subsystem does not last longer than $t_{max} = 0.218\ s$ in the critical system state.

Since the invariant proof includes the real time properties of the actions CoolingFailure and HeatingOffAfterCoolingFailure, however, we cannot generalize this proof to the whole example system directly. Furthermore, we have to ensure that these actions are not blocked by the environment of *ReactOnCoolingFailure*. Otherwise the verification that CONT keeps the invariant I_h might be spoiled due to the conditional character of realtime properties. This proof, however, is trivial since the other processes of the system either participate to CoolingFailure and HeatingOffAfterCoolingFailure with actions which are always enabled or with stuttering steps.

7. A framework for Hazard Analysis

Hybrid systems often contain similar subsystems. For instance, our example configuration of a boiler and a condenser is used in various plants to separate solutions. Consequently, during the hazard analysis of different plants, most questions and argumentations recur in a similar way. Therefore, we developed a framework that facilitates as well the design of formal system specifications as the proofs. It contains a library of modular component specifications modelled by cTLA process types. A system specification is developed by instantiating and composing component specifications to a system specification. Furthermore, the framework contains theorems, each stating that a subsystem pattern fulfills a safety property. These theorems are already verified in the way explained in Sec. 6. Thus, the user can prove safety properties by application of the theorems and has to check only that a pattern in a theorem is consistent to the specification of the technical system. These consistency checks are much simpler than the complete formal verification of a safety property. Comparable work in the field of computer communication protocols exists, showing that even difficult and very complex protocols can be verified quite easily [13].

The library of cTLA processes consists of three different kinds of specification modules. The first group contains processes of components and component constraints of a technical plant (fi., the processes *ReactOnAlarm*, *ReactMaxTime*, and *VesselVolume* introduced in figures 1–3). In a second group processes are listed that specify potential hazards. By these specifications one can introduce defective components like leaking valves or blocked lines to a system specification. The third group contains processes describing safety properties. For instance, the process type *VesselMaxPressure (maxpress)* describes that the pressure of a vessel does not exceed *maxpress*. Refering to this process type, we can express the property of our example — that the pressure in the boiler does not exceed $0.2\ Bar$ — by means of a process instance of the type *VesselMaxPressure (0.2)*.

Fig. 7 lists an example of a framework theorem which can be used to prove that the pressure in the boiler and the condenser does not exceed a certain value. The theorem exemplifies the general form of framework theorems which are implications. Each theorem states that a subsystem *Sys* of a certain pattern implies an interesting property if the subformulas *Pars* and *EnvCond* are true. The subsystem pattern *Sys* is structured similarly to cTLA system specifications (cf. fig. 6). The section PROCESSES lists the processes forming the pattern and the actual parameter settings. Besides processes specifying components of a technical system and besides hazard descriptions this section also may contain specifications of assumed system properties. The correctness of these assumed properties can be proven by other theorems of the framework. The coupling of the process actions of *Sys* is described in the section ACTIONS. *Pars* is a boolean condition guaranteeing that the processes of *Sys* and the property to be proven are parametrized con-

```
THEOREM MaxPressureBoilerCondenser
LET
  Sys ≜ PROCESSES  Subsystem
  Modular components of a technical plant:
    BoilerVol : VesselVolume
                (bcapacity, binitvolume);
    CondenserVol : VesselVolume
                   (ccapacity, cinitvolume);
    Vaporization : VesselVaporize
                   (fluiddense, gasdense,
                    enthalpy);
    ...;
  Already proven safety properties:
    CoolingPowerDecrease : PowerMaxDecrease
                           (cdecrate);
    BoilerMaxFluid : VesselMaxVolume
                     (bfluidmaxvolume);
    ...;
  ACTIONS
    CONT (INPUT ... ; OUTPUT ...) ≜ ...;
    continuous volume flow
    Alarm ≜ ...;  alarm cooling below minimum
    Release ≜ ...;  alarm cooling power above minimum
    HeatingOff ≜ ...;  switch off heating
    HeatingOn ≜ ...;  switch on heating
  Pars ≜ sensedpower > bmaxpower +
            cdecrate · sensetime ∧ ...;
  EnvCond ≜ Enabled(SensorAlarm.alarm) ⇒
              Sys.e_Alarm = "enab" ∧ ...;
IN  Sys ∧ Pars ∧ □EnvCond ⇒
      VesselMaxPressure (maxpress);
```

Figure 7. Theorem *MaxPressureBoilerCondenser*

sistently. *EnvCond* models an invariant to be kept by the environment. It assures that the environment of the pattern does not block the actions of *Sys* and therefore does not spoil liveness and realtime assumptions.

In order to prove a safety property, the user selects a suitable theorem from the framework and instantiates the parameters in the theorem according to the generic parameter replacements in the system specification. Since the theorems are already proven, one only has to check that *Sys* is a subsystem of the system specification, that the instantiated parameters fulfill the condition *Pars*, and that that the processes of the system specification not listed in *Sys* keep the environment condition *EnvCond*. The selection of theorems as well as the three checks are facilitated by the tool COAST [9]. Based on the system specification and the specifications of safety properties, COAST selects suitable theorems and performs the checks of *Sys* and *EnvCond* directly. Furthermore, COAST proves the condition *Pars* by means of a theorem prover.

8. Concluding remarks

We reported on the present results of a current research project funded by DFG which started with the proposal of "hybrid cTLA" as a TLA-based technique for modular temporal logic specifications of hybrid systems [12]. Thereafter, we investigated the suitability of constraint-oriented specification structures for the decomposition of formal verifications [10]. In the meanwhile, we identified recurring system patterns and hazard analysis problems and constructed the framework which we outlined in Sec. 7. Since the reuse of specification modules and theorems facilitates the formal specification of hybrid systems and the formal verification of safety properties, the framework is a useful means to support experts in performing HazOps. It is available via WWW (`http://ls4-www.cs.uni-dortmund.de/RVS/P-HYSYS/`).

In addition to COAST, a tool is in development examining specifications of hybrid systems for certain system faults and selecting, based on this examination, safety properties from the framework to be proven (fi., if the tool detects a vessel provided with a heating, it selects the safety property *VesselMaxPressure* to prevent the pressure in the vessel exceeding a maximum limit).

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.

[2] R. Alur, C. Courcoubetis, Th. A. Henzinger, and P.-H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736, pages 209–229. Springer Verlag, 1993.

[3] C. A. Catino and L. H. Ungar. A model-based approach to automated hazard identification of chemical plants. *AIChE Journal*, 41(3):97–109, 1995.

[4] K. M. Chandy and J. Misra. *Parallel Program Design — A Foundation*. Addison Wesley, 1988.

[5] M. Göring and H. G. Schecker. HAZEXPERT: An integrated expert system to support hazard analysis in process plant design. *Computers Chemical Engineering*, 17:429–434, 1993.

[6] H. Graf and H. Schmidt-Traub. A Model-Based Approach to Process Hazard Identification. In *Proceedings of 13th International Congress of Chemical and Process Engineering (CHISA)*, Prague, August 1998.

[7] K. M. Hansen, A. P. Ravn, and V. Stavridou. From Safety Analysis to Software Requirements. *IEEE Transactions on Software Engineering*, 24(7):573–584, July 1998.

[8] P. Heino, A. Poucet, and J. Soukas. Computer tools for hazard identification, modelling and analysis. *Journal of Hazardous Materials*, 29:445–463, 1992.

[9] P. Herrmann, O. Drögehorn, W. Geisselhardt, and H. Krumm. Tool-supported formal verification of highspeed transfer protocol designs. In *Proceedings of the 7th International Conference on Telecommunication Systems — Modelling and Analysis*, pages 531–541, Nashville, TN., USA, March 1999. ATSMA.

[10] P. Herrmann, G. Graw, and H. Krumm. Compositional Specification and Structured Verification of Hybrid Systems in cTLA. In *Proceedings of the 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC98)*, pages 335–340, Kyoto, April 1998. IEEE Computer Society Press.

[11] P. Herrmann and H. Krumm. Compositional Specification and Verification of High-Speed Transfer Protocols. In S. T. Vuong and S. T. Chanson, editors, *Protocol Specification, Testing, and Verification XIV*, pages 339–346, Vancouver, 1994. IFIP, Chapman & Hall.

[12] P. Herrmann and H. Krumm. Specification of Hybrid Systems in cTLA+. In *Proceedings of the 5th International Workshop on Parallel & Distributed Real-Time Systems (WPDRTS'97)*, pages 212–216, Geneva, 1997. IEEE Computer Society Press.

[13] P. Herrmann and H. Krumm. Modular Specification and Verification of XTP. *Telecommunication Systems*, 9(2):207–221, 1998.

[14] J. Hooman. A Compositional Approach to the Design of Hybrid Systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736, pages 121–148. Springer Verlag, 1993.

[15] S. Kowalewski, S. Engell, M. Fritz, R. Gesthuisen, G. Regner, and M. Stobbe. Modular discrete modelling of batch processes by means of condition/event systems. In *Workshop on Analysis and Design of Event-Driven Operations in Process Systems*, Imperial College, London, April 1995.

[16] S. Kowalewski and H.-M. Hanisch. Permissive control of boolean condition/event systems: synthesis and limits. In *IEEE Symposium on Intelligent Control*, pages 118–123, Ohio, 1994. IEEE.

[17] R. Kurki-Suonio. Fundamentals of object-oriented specification and modeling of collective behaviors. In H. Kilov and W. Harvey, editors, *Object-Oriented Behavioral Specifications*, pages 101–120. Kluwer Academic Publishers, 1996.

[18] L. Lamport. Hybrid Systems in TLA$^+$. In R. L. Grossmann, A. Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736, pages 77–102. Springer Verlag, 1993.

[19] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[20] H. G. Lawley. Operability Studies and Hazard Analysis. *Chemical Engineering Progress*, 70(4):45–56, April 1974.

[21] S. T. Probst. *Chemical Process Safety and Operability Analysis using Symbolic Model Checking*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, May 1996.

[22] Y. Shimada, K. Suzuki, and H. Sayama. Computer-aided operability study. *Computers Chemical Engineering*, 20(6/7):905–913, 1996.

[23] R. Srinivasan and V. Venkatasubramanian. Petri Net-Digraph models for automating HAZOP analysis of batch process plants. *Computers Chemical Engineering*, 20:719–725, 1996.

[24] O. Stursberg, H. Graf, S. Engell, and H. Schmidt-Traub. A concept for safety analyses of chemical plants based on discrete models with an adapted degree of abstraction. In *Proceedings of 4th International Workshop on Discrete Event Systems (WODES)*, Cagliari, August 1998.

[25] R. Vaidhyanathan and V. Venkatasubramanian. Experience with an expert system for automated HAZOP analysis. *Computers Chemical Engineering*, 20:1589–1594, 1996.

[26] C. A. Vissers, G. Scollo, and M. van Sinderen. Architecture and specification style in formal descriptions of distributed systems. In S. Agarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, pages 189–204, 1988. IFIP, Elsevier.

[27] A. Waters and J. W. Ponton. Qualitative simulation and fault propagation in process plants. *Chemical Engineering Research Descriptions*, 67:407–422, 1989.