INSTITUT FÜR INFORMATIK UND PRAKTISCHE MATHEMATIK

LEHRSTUHL FÜR SOFTWARETECHNOLOGIE

An Abstract Model of VHS Case Study 1 (Experimental Batch Plant)

Ben Lukoschus

Bericht Nr. TR-ST-99-2

May 1999

CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL

An Abstract Model of VHS Case Study 1* (Experimental Batch Plant)

Ben Lukoschus
(bls@informatik.uni-kiel.de)

Chair of Software Technology
Institute of Computer Science and Applied Mathematics
Christian Albrechts University of Kiel

May 1999

Abstract

In this work we describe the verification of safety properties for the experimental batch plant of case study 1 using the model checker SMV. Discrete condition/event systems are used to describe the physical parts of the plant and the control programs. This model is transformed into SMV code, and the properties are checked.

1 Introduction

The experimental batch plant [Kow98] can be divided into two parts, the physical part consisting of tanks, valves, pumps, the heater, the condenser, cooling units, piping, and sensors (temperature, water level, concentration, pressure), and the control part, namely the distributed control system with the 13 control routines. The control part receives sensor readings from the physical part and sends control commands to the physical devices. This communication structure is shown in Figure 1.

The goal of this work is to verify some properties of the control programs on a formal basis. We focus on safety requirements which help to ensure that the control programs will not cause any damages to the plant, like, e.g., by switching on the heater in an empty tank.

*This work was partially supported by the European Community in the Esprit Long-Term Research Project 26270 – VHS (Verification of Hybrid Systems).

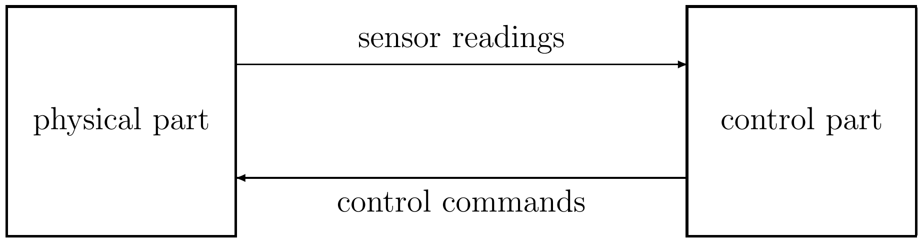

Figure 1: Communication structure of the experimental batch plant.

In order to check the requirements formally, we need a modeling framework which allows formal verification. Our approach uses *discrete condition/event systems* (DCESs) [SK91] for both the physical and the control part. A modular structure is used; we define one DCES for each valve, pump, tank, heater, and also one DCES for each of the 13 control programs. This allows compositional reasoning in the verification process.

Since DCESs are finite-state, we can apply an enumerative verification technique known as *model checking* [QS82, CES83]. We use the tool SMV (Symbolic Model Verifier) [McM93, McM92] to perform this verification. A compiler prototype translates a set of given DCESs into an SMV input file, and each safety property is written as a CTL formula [CE81], which is then verified by SMV.

2 The Model

We use a discrete, untimed model. Although it is quite simple, many safety aspects of the interaction of the plant and the control programs can be examined. As the modeling paradigm we use *discrete condition/event systems* (DCESs), which are introduced in [SK91]. To illustrate the connections between DCESs more clearly, we assign names to their input and output components, and thus extend them to *named* DCESs, a notion which is introduced in Chapter 4 of [Luk99].

Based on the plant description in [Kow98], our model consists of two parts. The first part contains the physical devices of the plant, like valves, pumps, tanks, etc. The second part introduces one DCES for each of the control programs, which are given as sequential function charts (SFCs) in the description.

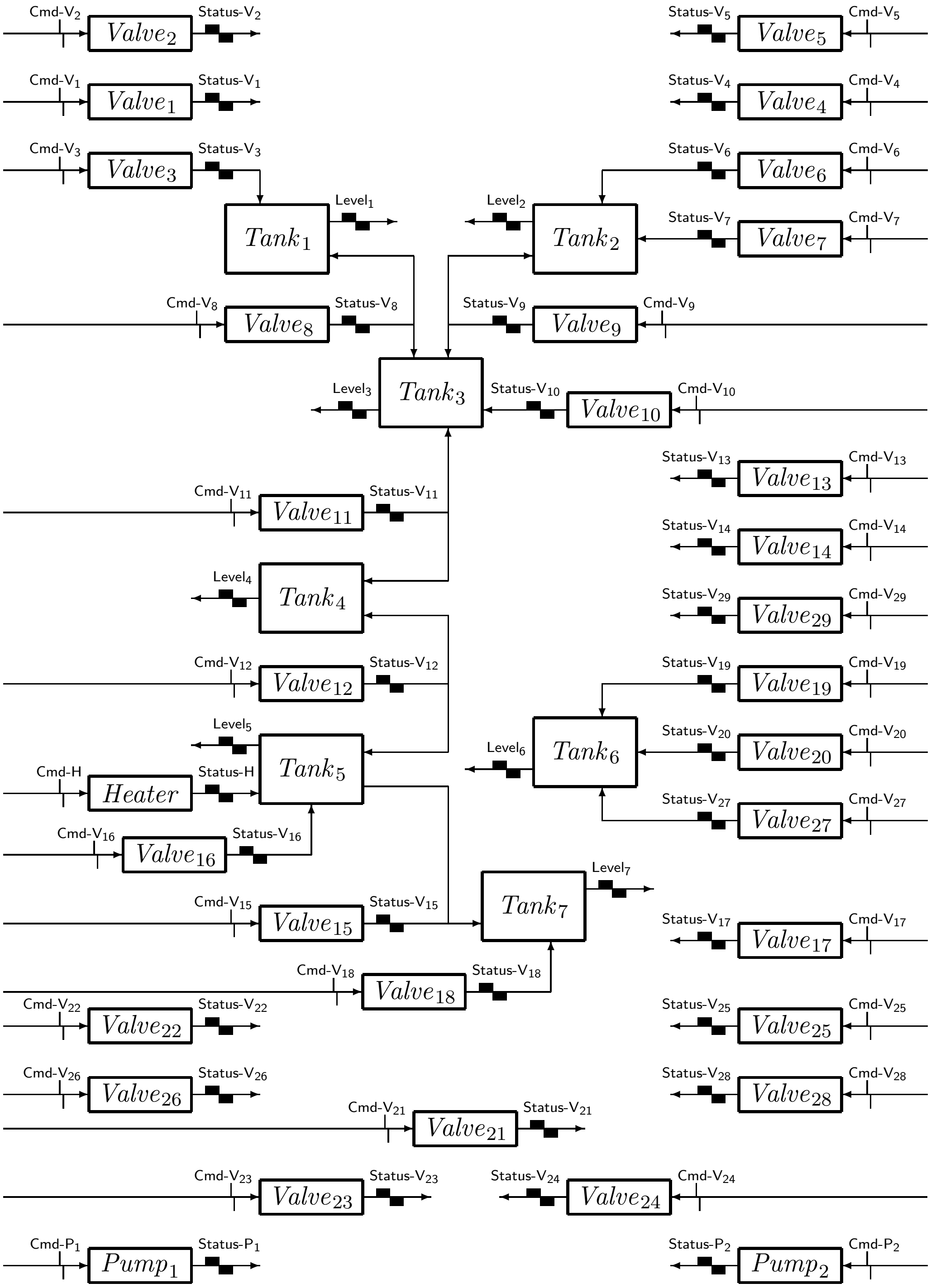

Figure 2: Block diagram of the physical devices modeled as DCESs.

2.1 The Physical Devices

We model the following physical devices of the plant:

- The 29 valves. These are either closed or open, and discrete events can be used to force them to change their state.
- The 2 pumps. They are either not pumping or pumping, and discrete events can force them to change their state.
- The 7 tanks. Each of these models the quantity of liquid in a tank. The water level, which depends on the states of the valves connected to the tank, is given as one of two (tank 5: three) discrete values.
- The heating unit. It is either off or on, and discrete events are used to change its state.

Figure 2 shows a block diagram of the interconnections of all physical devices. The input event connections which can be accessed by the control programs are at the left and right hand side of the diagram.

Since we do not model different concentrations or temperatures of the salt solutions or any pressure differences in pipes or tanks, and since we always assume that the condenser is working properly, the following devices of the plant are not modeled: The mixing unit in tank 3, the condenser, the cooling units, and all kinds of sensors (pressure, concentration, temperature).

For each of the devices, the input and output components with their associated alphabets of the respective DCES are shown in a block diagram. The states and transitions of the DCES, which define the input/output behavior, are shown in a transition diagram. The precise semantics of such a transition diagram is given in Chapter 6 of [Luk99].

2.1.1 The Valves

We have one DCES $Valve_i$ for each valve i, $i \in \{1, \ldots, 29\}$. Since all the valves are isomorphic, we only describe $Valve_1$ here.

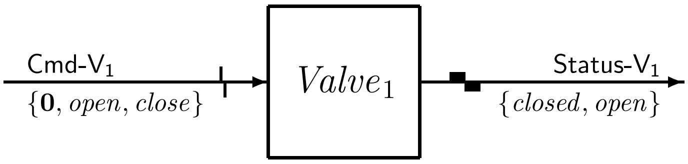

Figure 3: Block diagram of $Valve_1$.

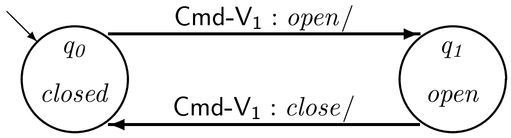

Figure 4: Transition diagram of $Valve_1$.

The valve has one event input named $\mathsf{Cmd\text{-}V_1}$ with the associated alphabet $\{\mathbf{0}, open, close\}$. Initially, the valve is closed. The state of $Valve_1$ can be controlled by sending a $\mathsf{Cmd\text{-}V_1} : open$ or $\mathsf{Cmd\text{-}V_1} : close$ event. The current state of the valve can be observed at the condition output named $\mathsf{Status\text{-}V_1}$, which can take on the values *closed* or *open*.

In our model, the valves have no switching delays or malfunctions.

2.1.2 The Pumps

The two pumps are modeled by the DCESs $Pump_1$ and $Pump_2$. Since both are isomorphic, only pump 1 is shown.

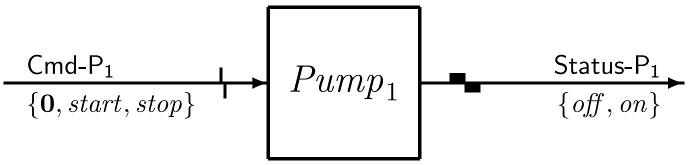

Figure 5: Block diagram of $Pump_1$.

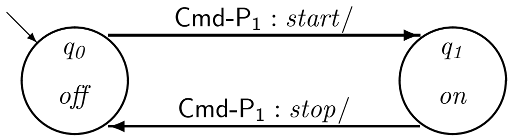

Figure 6: Transition diagram of $Pump_1$.

The pump has one event input named $\mathsf{Cmd\text{-}P_1}$ with the associated alphabet $\{\mathbf{0}, start, stop\}$. Initially, the pump is off. The state of $Pump_1$ can be controlled by sending a $\mathsf{Cmd\text{-}P_1} : start$ or $\mathsf{Cmd\text{-}V_1} : stop$ event. The current state of the pump can be observed at the condition output named $\mathsf{Status\text{-}P_1}$, which can take on the values *off* or *on*.

The pumps also don't have switching delays or malfunctions. The pressure limit switches of the pumps are not modeled; we do however verify later that the control programs never switch on a pump that would pump against a closed valve.

2.1.3 The Tanks

Each DCES model of a tank is used to provide information about the water level in the respective tank of the plant. We use two different levels (*empty* and *full*) in our model, only tank 5 has a third level (*half*). These are output condition symbols of the DCES tank models. Whenever a control program needs to access a water level sensor, it checks the condition output of the respective tank. We do not model the water level sensors themselves.

A change of the water level of a tank depends on the amount of water flowing in and out of the tank. This again depends on pressure differences and the state of the valves. In this model a simple abstraction is chosen: Whenever a valve controlling an input pipe of a tank is open, water may flow into the tank, and the tank's level may change from *empty* to *full*. And whenever a valve controlling an output pipe of a tank is open, water may leave the tank, and the tank's level may change from *full* to *empty*. It is obvious that this is an abstraction of the real plant behavior.

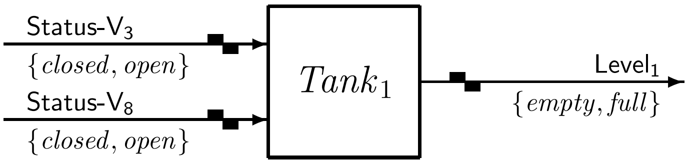

Figure 7: Block diagram of $Tank_1$.

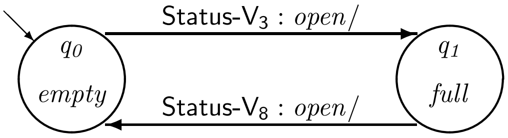

Figure 8: Transition diagram of $Tank_1$.

Tank 1 can be filled via valve 3 and emptied via valve 8. Thus, Status-V_3 and Status-V_8 are condition input names of $Tank_1$. Status-V_3 : *open* enables the transition from q_0 to q_1 (when taken, this transition yields the condition output change from Level_1 : *empty* to Level_1 : *full*), and $\mathsf{Status\text{-}V}_8$: *open* enables the transition from q_1 to q_0, and if this transition is taken, the condition output changes from Level_1 : *full* to Level_1 : *empty*.

Note that these transitions need not to be taken when they are enabled; in fact they may never be taken, since condition changes cannot force any transitions. This behavior is intended, making this an obvious abstraction of the real plant behavior.

The control programs can read the condition output Level_1 to obtain the current water level in tank 1.

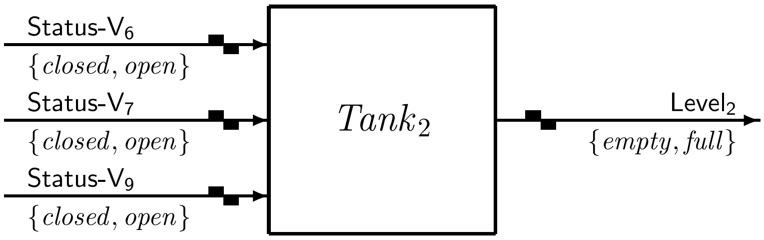

Figure 9: Block diagram of $Tank_2$.

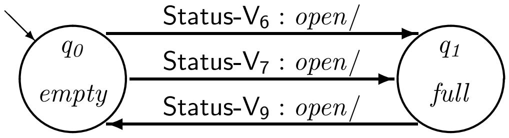

Figure 10: Transition diagram of $Tank_2$.

Tank 2 can be filled via valve 6 or valve 7, and it can be emptied via valve 9. The water level is represented by the condition output Level_2.

Tank 3 can be filled via valve 8 or valve 9, and it can be emptied via valve 10 or valve 11. The water level is represented by the condition output Level_3. The mixing unit and the manual input of salt is not modeled.

Tank 4 can be filled via valve 11 and emptied via valve 12. The water level is represented by the condition output Level_4.

Tank 5 can be filled via valve 12 and emptied via valve 15 or valve 16. The water level is represented by the condition output Level_5, which can take on the values *empty*, *half*, and *full*. We use three values to have a representation of the water level that suits the structure of control program B5 better than a two-valued representation.

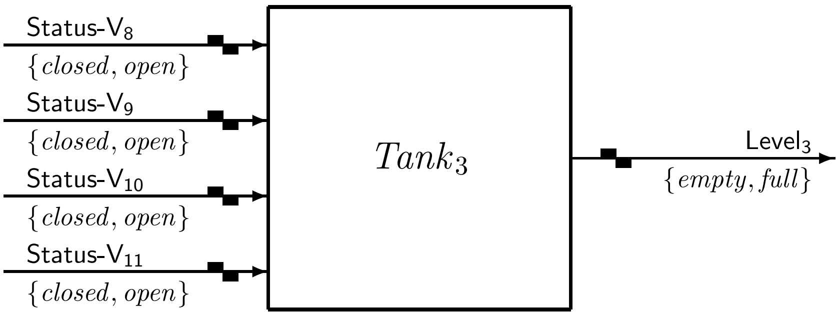

Figure 11: Block diagram of $Tank_3$.

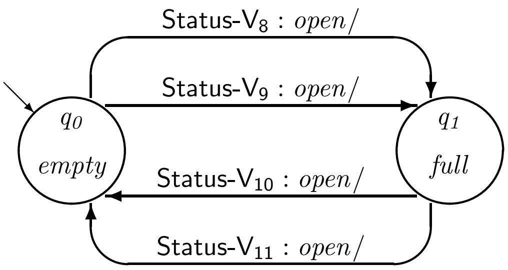

Figure 12: Transition diagram of $Tank_3$.

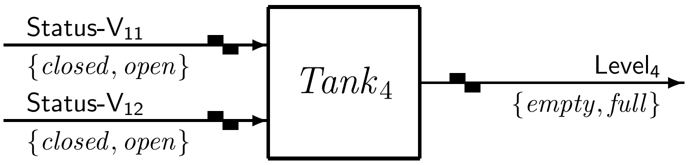

Figure 13: Block diagram of $Tank_4$.

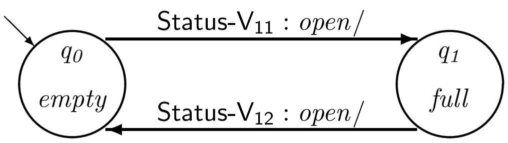

Figure 14: Transition diagram of $Tank_4$.

If the heater is working, steam may be produced, decreasing the water level in tank 5. This is modeled by the two transitions enabled by the Status-H : *on* condition.

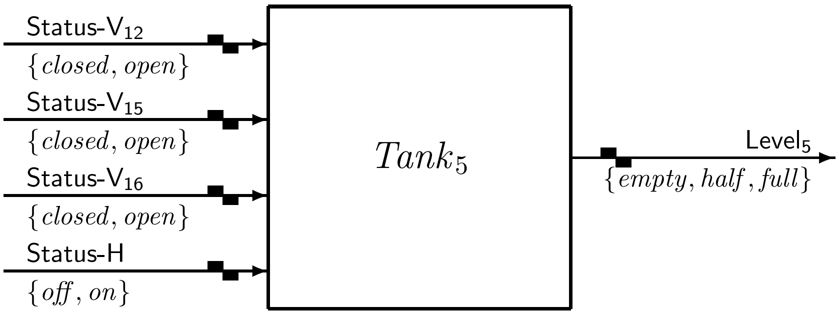

Figure 15: Block diagram of $Tank_5$.

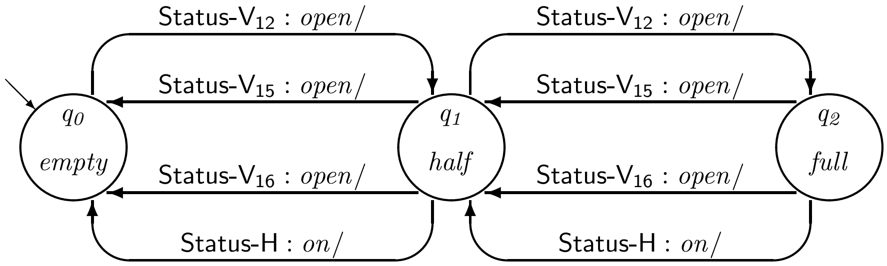

Figure 16: Transition diagram of $Tank_5$.

Tank 6 can be filled via valve 14 or by water pouring out of the condenser, and it can be emptied via valve 19, valve 20, or valve 27. The water level is represented by the condition output Level_6. We assume that water can leave the condenser at any time, even if the heater is not working, since it takes some time to cool down the steam and since there is a delay between switching off the heater and the end of steam production in tank 5. Therefore, we have an unconditional transition from state q_0 to state q_1. As a side effect of this we do not need Status-V_{14} as a condition input, since an additional transition from q_0 to q_1 enabled by Status-V_{14} : *open* would not change the semantics of the DCES $Tank_6$. The cooling unit of tank 6 is not modeled.

Tank 7 can be filled via valve 15 and emptied via valve 17. The water level is represented by the condition output Level_7. The cooling unit of tank 7 is not modeled.

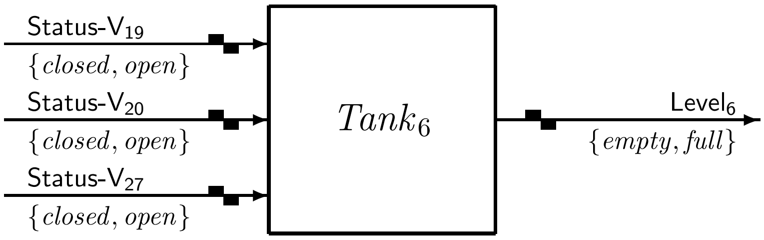

Figure 17: Block diagram of $Tank_6$.

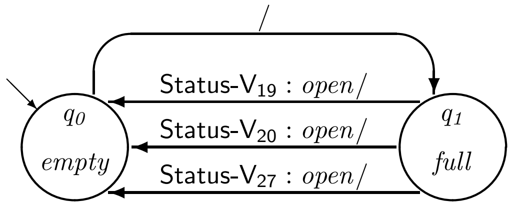

Figure 18: Transition diagram of $Tank_6$.

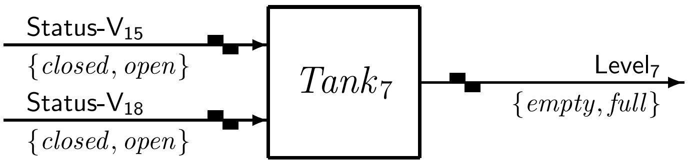

Figure 19: Block diagram of $Tank_7$.

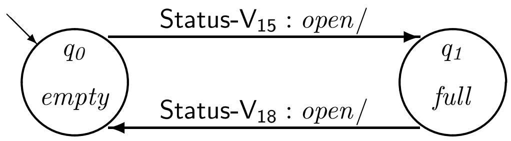

Figure 20: Transition diagram of $Tank_7$.

2.1.4 The Heater

The heater is modeled by the DCES *Heater*. Heating can be started by sending the Cmd-H : *start* event and stopped by sending the Cmd-H : *stop* event. The current state of the heater can be observed at the Status-H condition output.

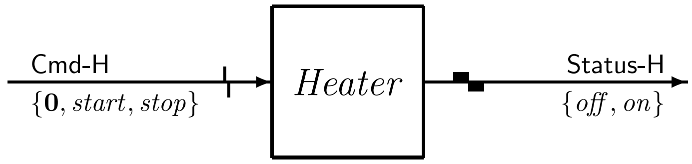

Figure 21: Block diagram of *Heater*.

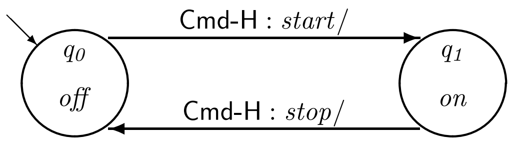

Figure 22: Transition diagram of *Heater*.

2.2 The Control Programs

The control programs are given in [Kow98] as Sequential Function Charts (SFCs). We implemented these SFCs as DCESs, trying to stay as close to their original structure as we can in our abstract framework. Each numbered step i of the SFC becomes a control state q_i in the DCES. The commands executed in step i are represented in the events generated when entering q_i. Conditions which enable the change from step i to step j are the enabling conditions of the DCES transition from q_i to q_j.

The following parts of the SFCs are not modeled, since we do not model the respective physical devices: User interaction with a display and some confirmation input, concentration sensor inputs, the cooling liquid flow sensor, and temperature sensors.

All DCES control programs are started by an external event $\mathsf{Cmd\text{-}Prg_N}$: *start*, where N is the name of the program. When this event occurs, the control state changes from q_1 to q_2, and the control program interacts with the physical devices until some final control state is reached. In contrast to the SFCs, the DCESs only do one cycle, i.e., they do not return to the initial control state when they have reached some final control state.

2.2.1 Program Prg_{B2}

This program fills tank 2 with fresh water via valve 7 until the maximum level is reached.

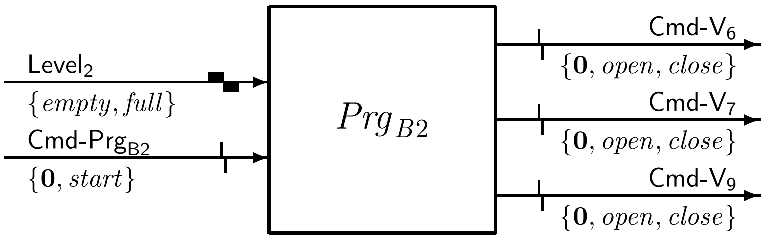

Figure 23: Block diagram of Prg_{B2}.

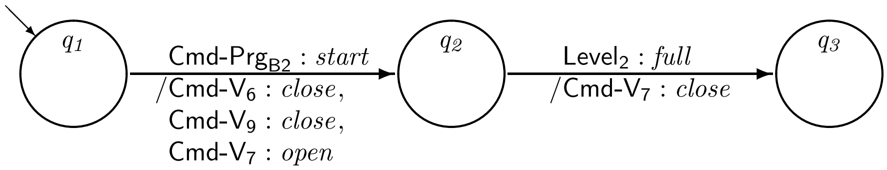

Figure 24: Transition diagram of Prg_{B2}.

2.2.2 Program Prg_{B3}

This program produces the concentrated brine in tank 3 by mixing highly concentrated brine from tank 1 via valve 8 and water from tank 2 via valve 9. Finally, the solution is drained into tank 4 via valve 11. Note that the mixing unit and the concentration sensor is not modeled.

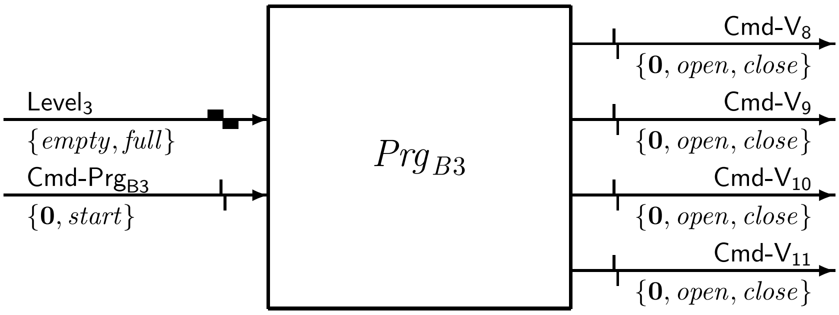

Figure 25: Block diagram of Prg_{B3}.

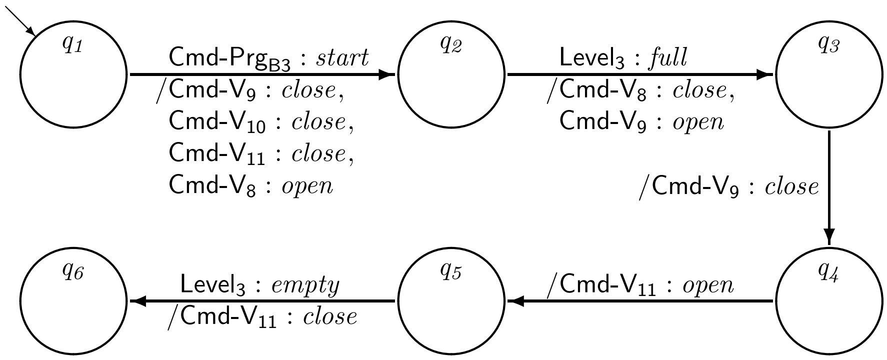

Figure 26: Transition diagram of Prg_{B3}.

2.2.3 Program Prg_{B3K}

This program controls the manual production of the concentrated brine in tank 3. First tank 3 is filled with water from tank 2 via valve 9 until a desired level is reached. Then salt is inserted manually (not modeled) while the mixing unit (not modeled) is working until a certain concentration is reached (concentration sensor not modeled), and then again water is added via valve 9 until the desired concentration is reached. The last step is needed because the user will normally add some more salt after he is signaled to stop adding salt. Finally the mixing unit is stopped and valve 9 is closed.

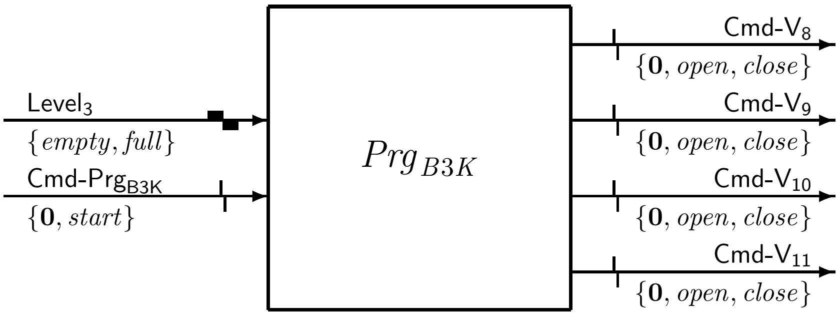

Figure 27: Block diagram of Prg_{B3K}.

2.2.4 Program Prg_{B3U}

This program controls pumping the manually produced concentrated brine in tank 3 into tank 1 via valves 10, 23, 22, 1, and 3, using pump 1.

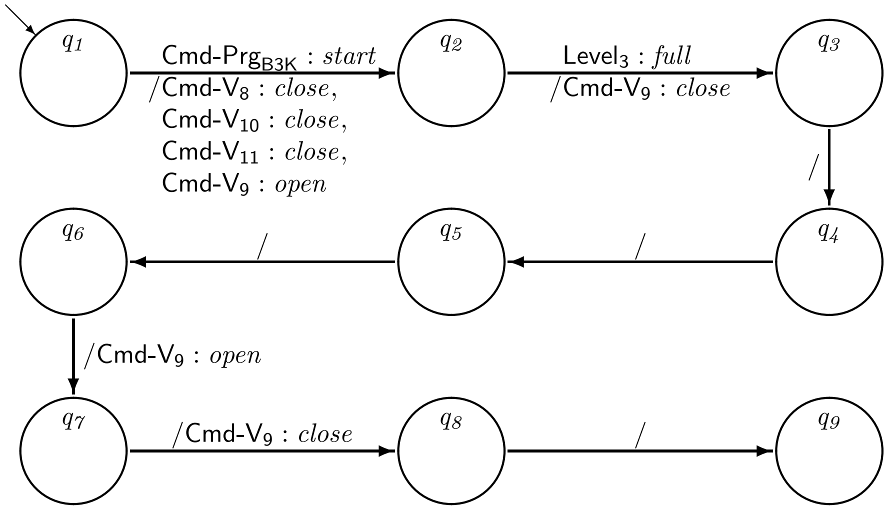

Figure 28: Transition diagram of Prg_{B3K}.

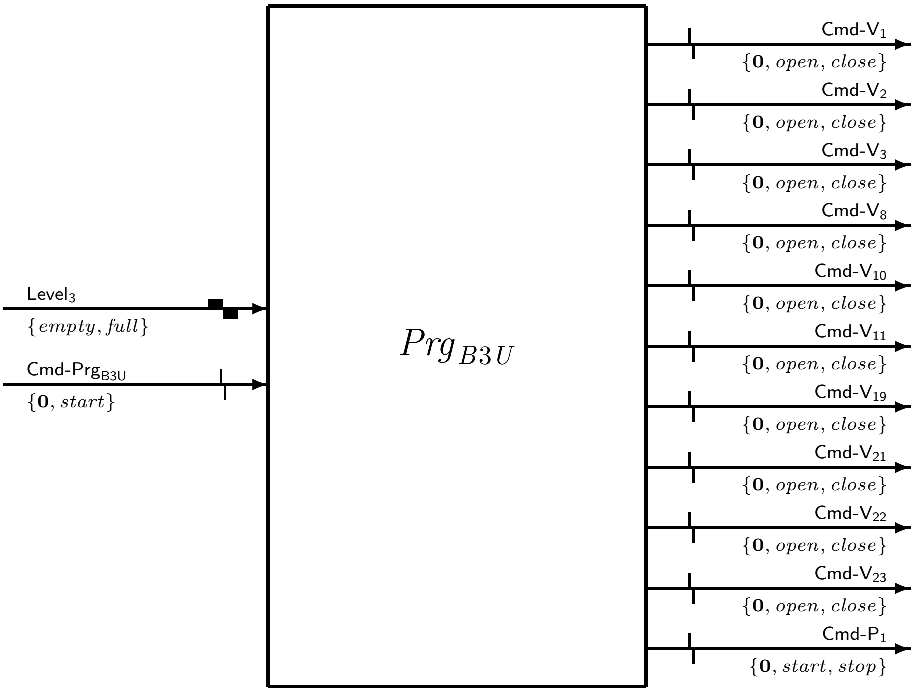

Figure 29: Block diagram of Prg_{B3U}.

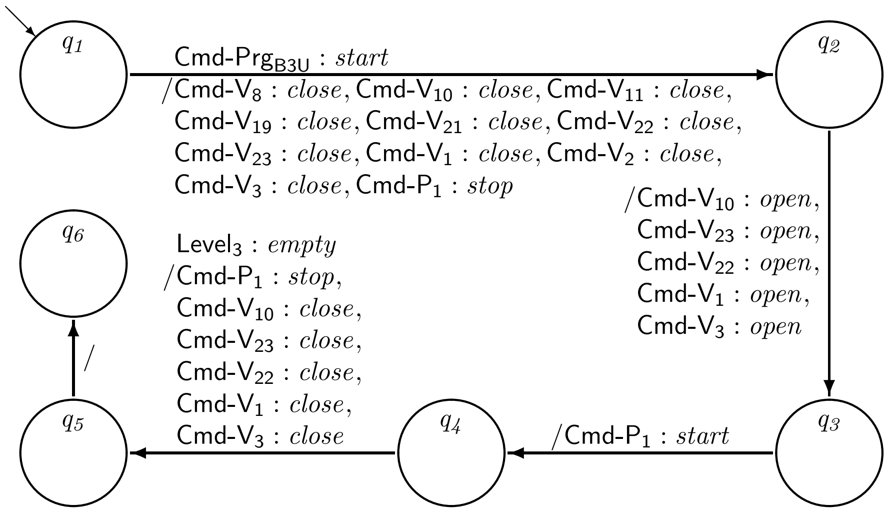

Figure 30: Transition diagram of Prg_{B3U}.

2.2.5 Program Prg_{B5}

This program controls the evaporation of the concentrated brine in tank 5 until the desired concentration is reached. First some concentrated brine from tank 4 is filled into tank 5 via valve 12 until tank 5 is full. Then the cooling in the condenser is started by opening valve 13. The heater is switched on, and the evaporation starts. If the desired concentration is reached (not modeled), the heater is switched off, and the tank 5 is drained into tank 7 via valve 12.

If during the evaporation process the water level in tank 5 goes below 8 cm (modeled as Level$_5$: *half*), the heater is switched of, and we enter state q_6, denoting a failure.

2.2.6 Program Prg_{B6}

This program controls the cooling of the distilled water in tank 6. The cooling water valve 29 is opened until the temperature in tank 6 is below 25 °C (temperature sensor not modeled). If tank 6 is empty at the start of the cooling phase, we enter state q_3, denoting a failure.

2.2.7 Program Prg_{B6A}

This program drains the contents of tank 6 via valve 27.

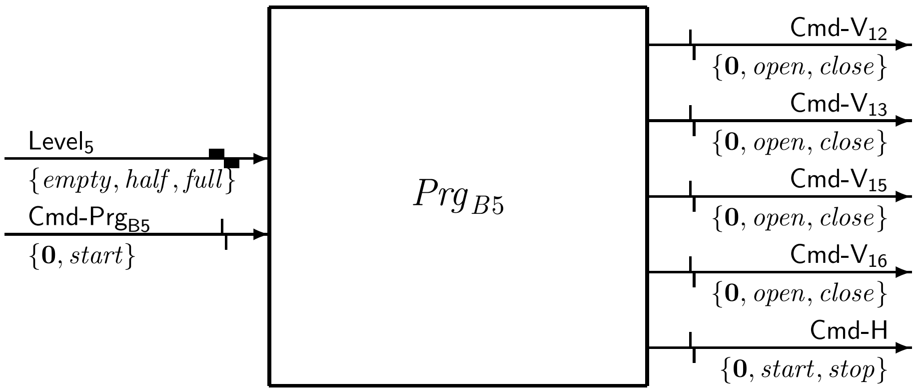

Figure 31: Block diagram of Prg_{B5}.

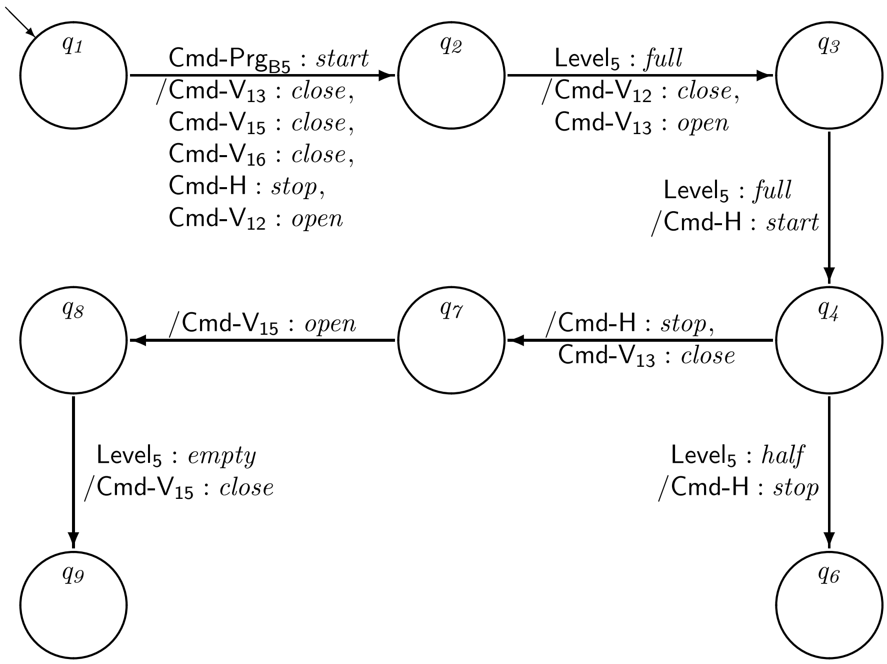

Figure 32: Transition diagram of Prg_{B5}.

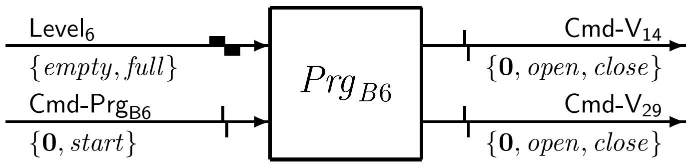

Figure 33: Block diagram of Prg_{B6}.

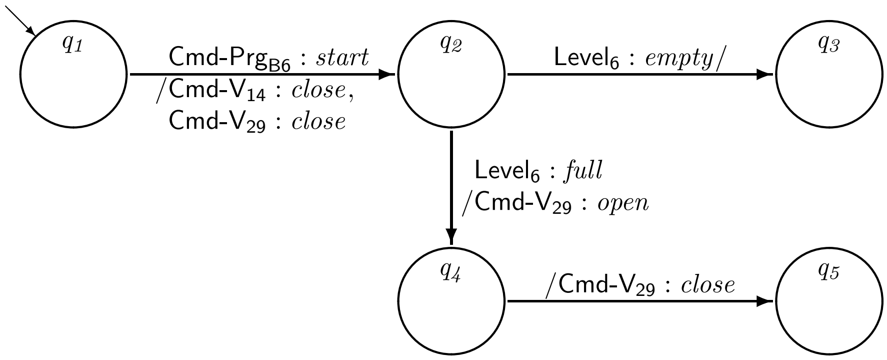

Figure 34: Transition diagram of Prg_{B6}.

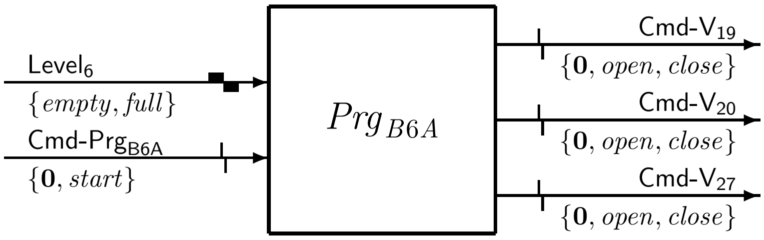

Figure 35: Block diagram of Prg_{B6A}.

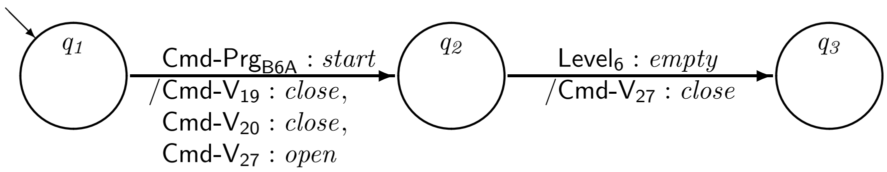

Figure 36: Transition diagram of Prg_{B6A}.

2.2.8 Program Prg_{B6S}

This program controls pumping the cooled distilled water from tank 6 into tank 2 via valves 20, 21, 23, 22, 1, 2, 4, and 6, using pump 1.

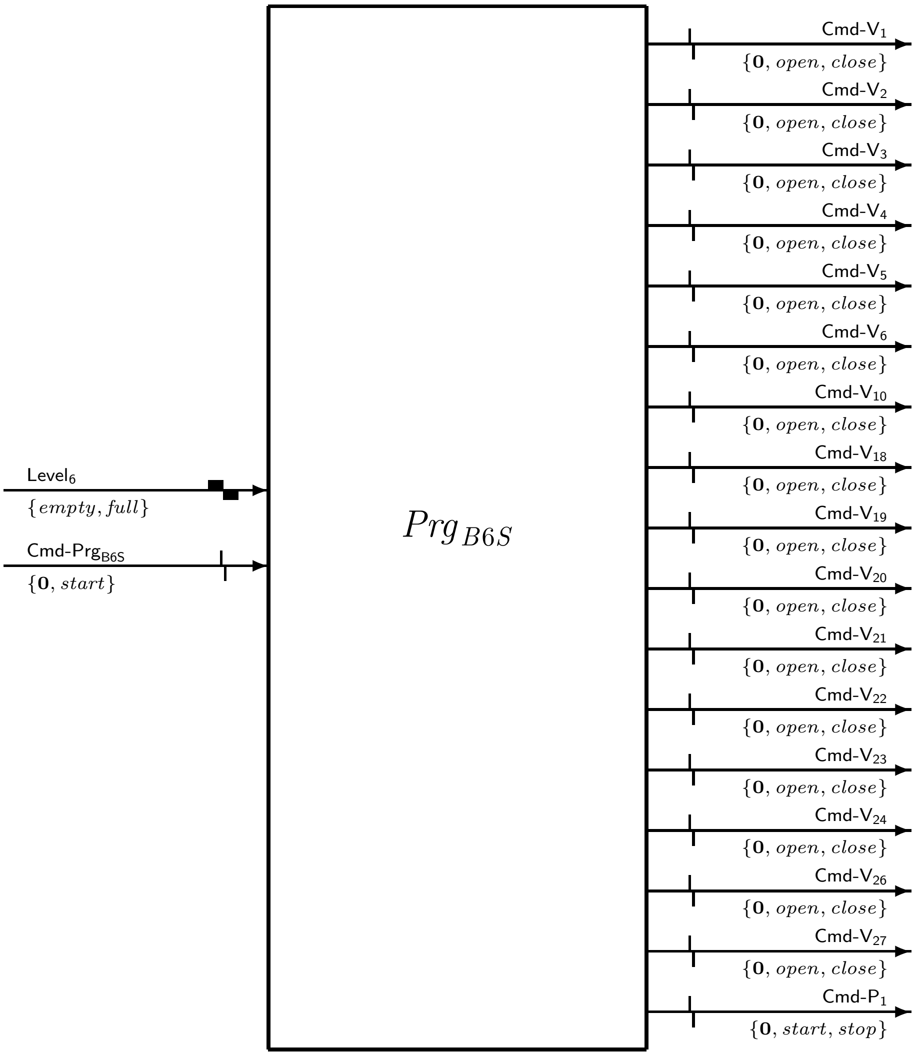

Figure 37: Block diagram of Prg_{B6S}.

2.2.9 Program Prg_{B6U}

This program controls pumping the cooled distilled water from tank 6 into tank 2 via valves 20, 24, 25, 5, and 6, using pump 2. It can be used instead of program Prg_{B6S} if pump 1 is not working.

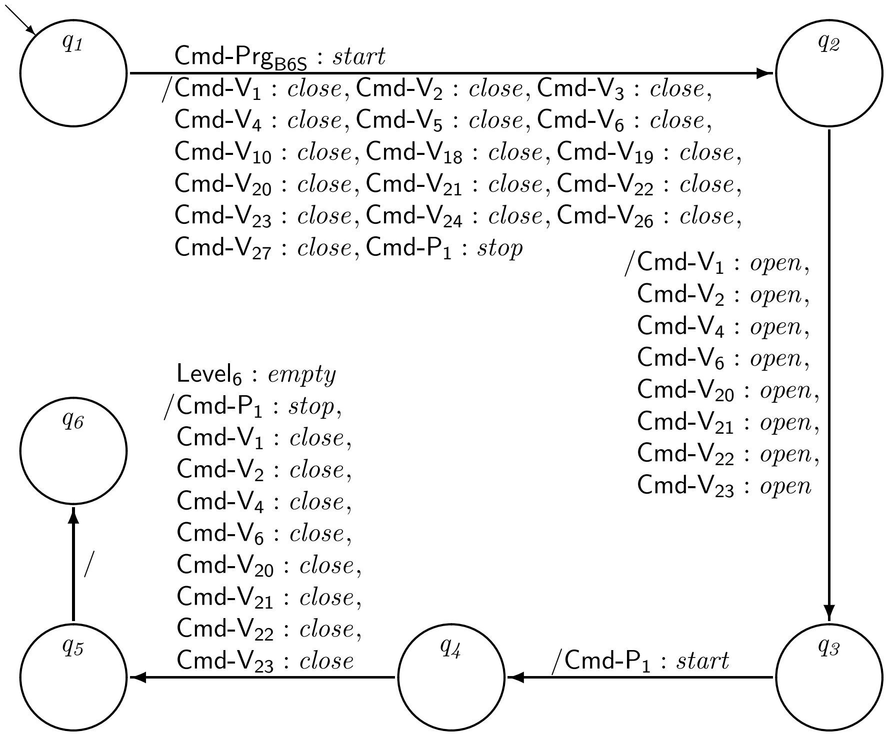

Figure 38: Transition diagram of Prg_{B6S}.

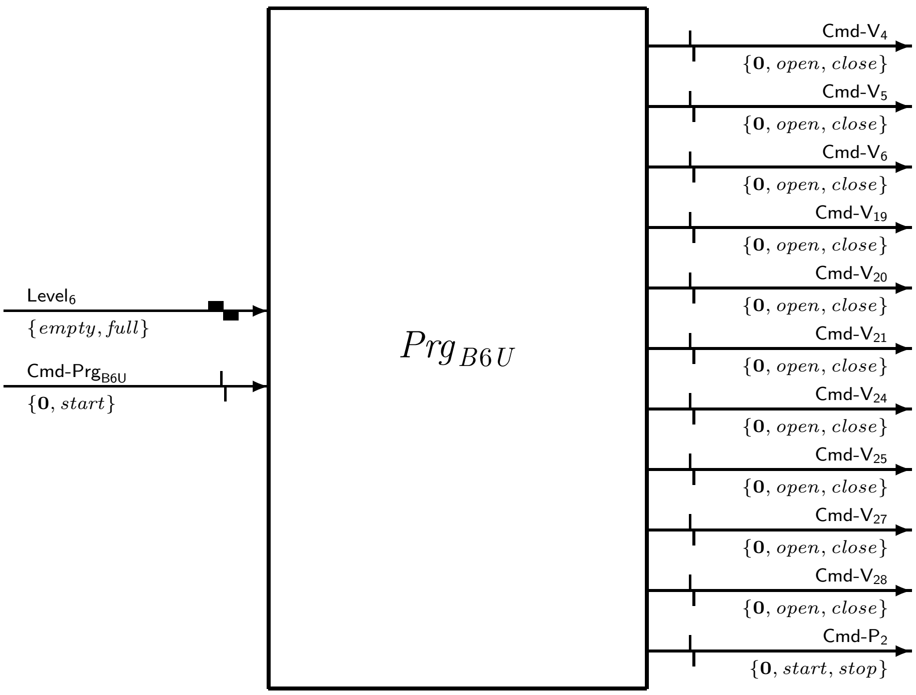

Figure 39: Block diagram of Prg_{B6U}.

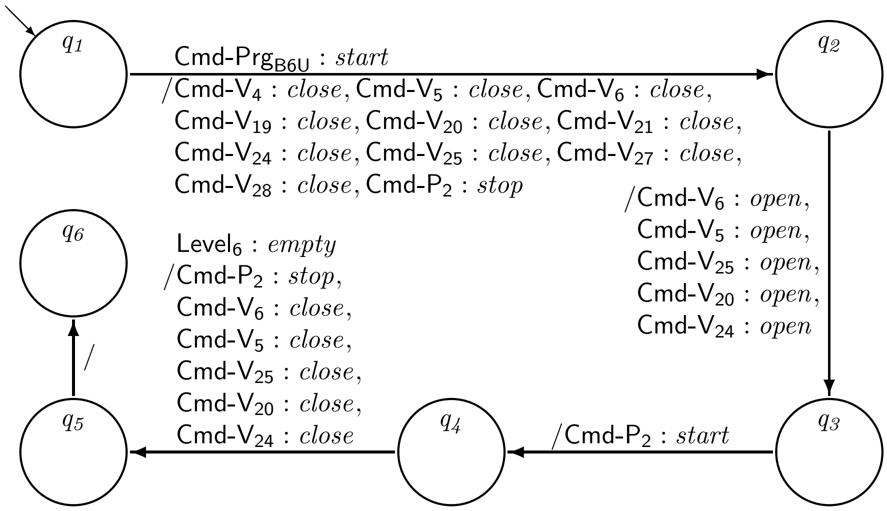

Figure 40: Transition diagram of Prg_{B6U}.

2.2.10 Program Prg_{B7}

This program controls the cooling of the concentrated brine in tank 7. The cooling water valve 17 is opened until the temperature in tank 7 is below 25 °C (temperature sensor not modeled). If tank 7 is empty at the start of the cooling phase, we enter state q_3, denoting a failure.

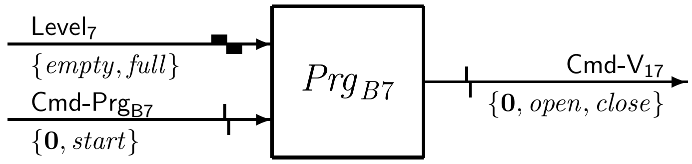

Figure 41: Block diagram of Prg_{B7}.

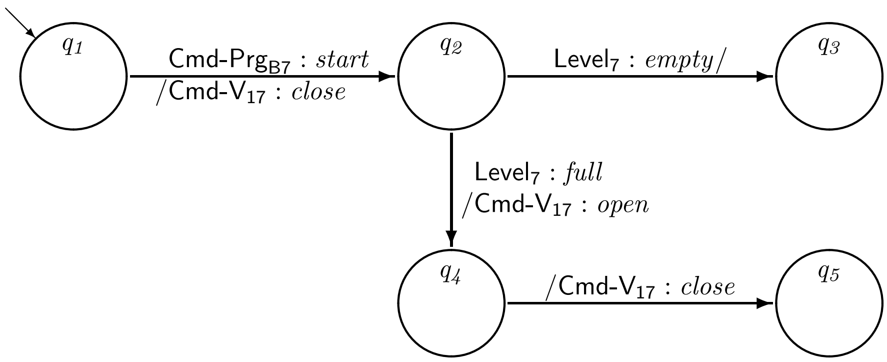

Figure 42: Transition diagram of Prg_{B7}.

2.2.11 Program Prg_{B7U}

This program controls pumping the cooled concentrated brine from tank 7 into tank 1 via valves 18, 23, 22, 1, and 3, using pump 1.

2.2.12 Program Prg_{SP1}

This program is used for rinsing the outer ring pipe with water from tank 2 using pump 1. First tank 2 is filled with fresh water via valve 7. If tank 2 is full, valve 7 is closed, valve 9 opened, and tank 3 is filled. If the level in tank 3 has reached some desired level, valve 10 is opened and pump 1 is started. Now water is flowing through the valves 10, 23, 22, 1, 2, 4, 5, and 28. If tank 3 is empty, the pump is stopped, and all valves are closed.

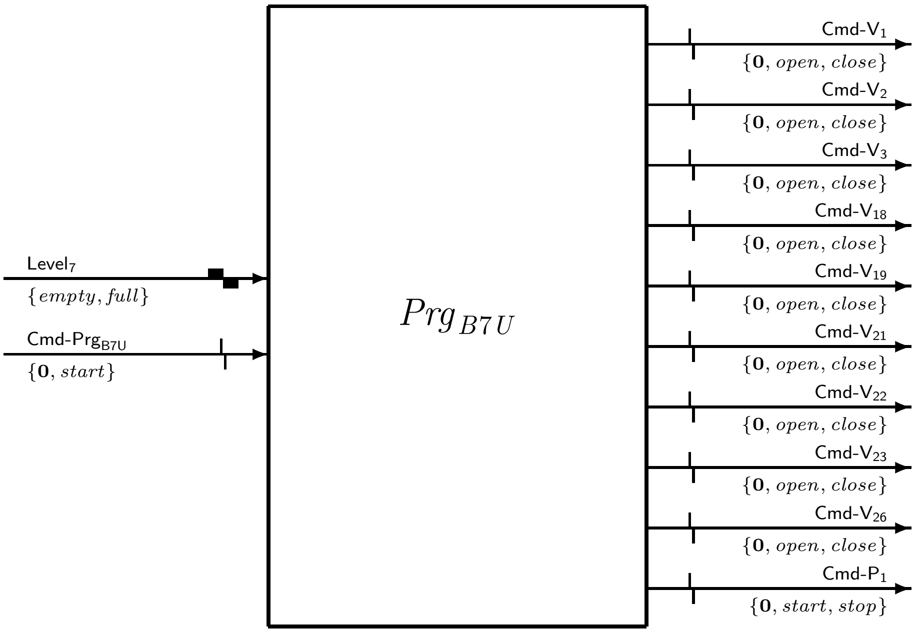

Figure 43: Block diagram of Prg_{B7U}.

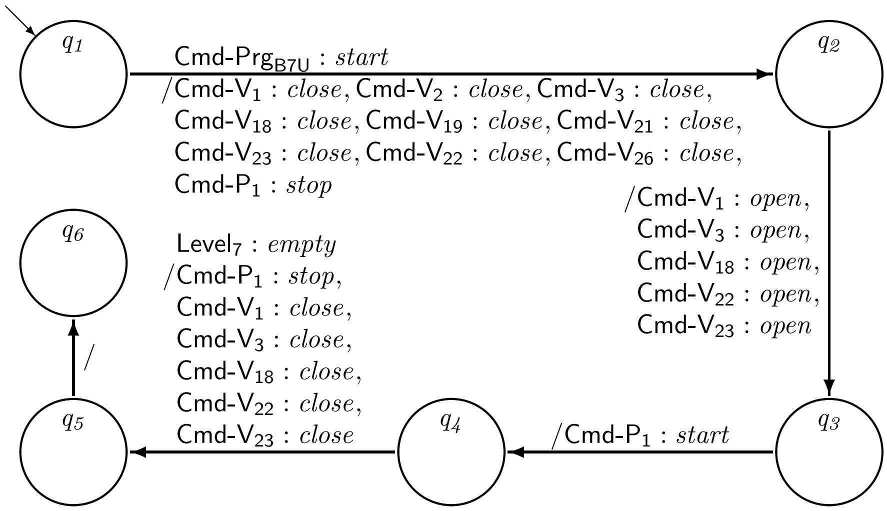

Figure 44: Transition diagram of Prg_{B7U}.

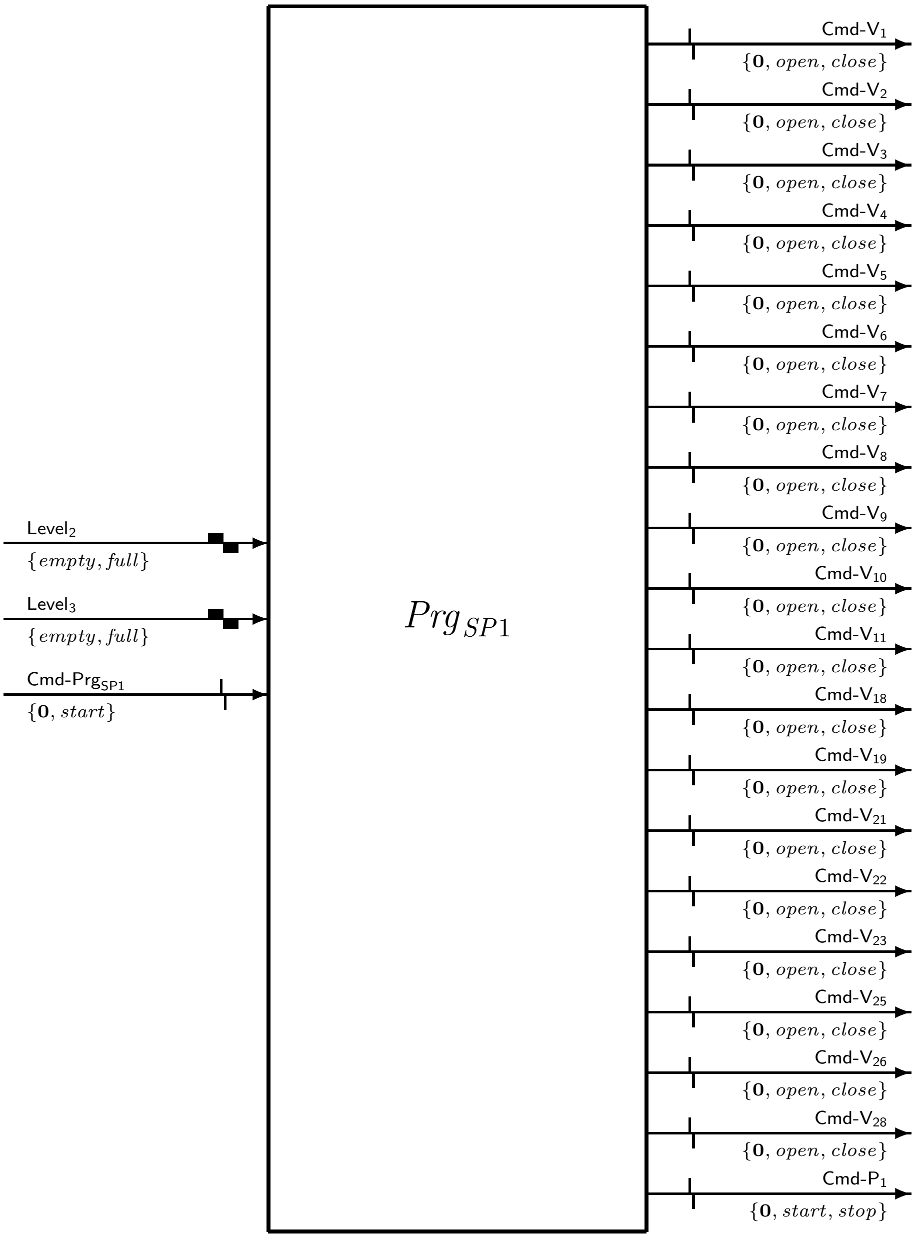

Figure 45: Block diagram of Prg_{SP1}.

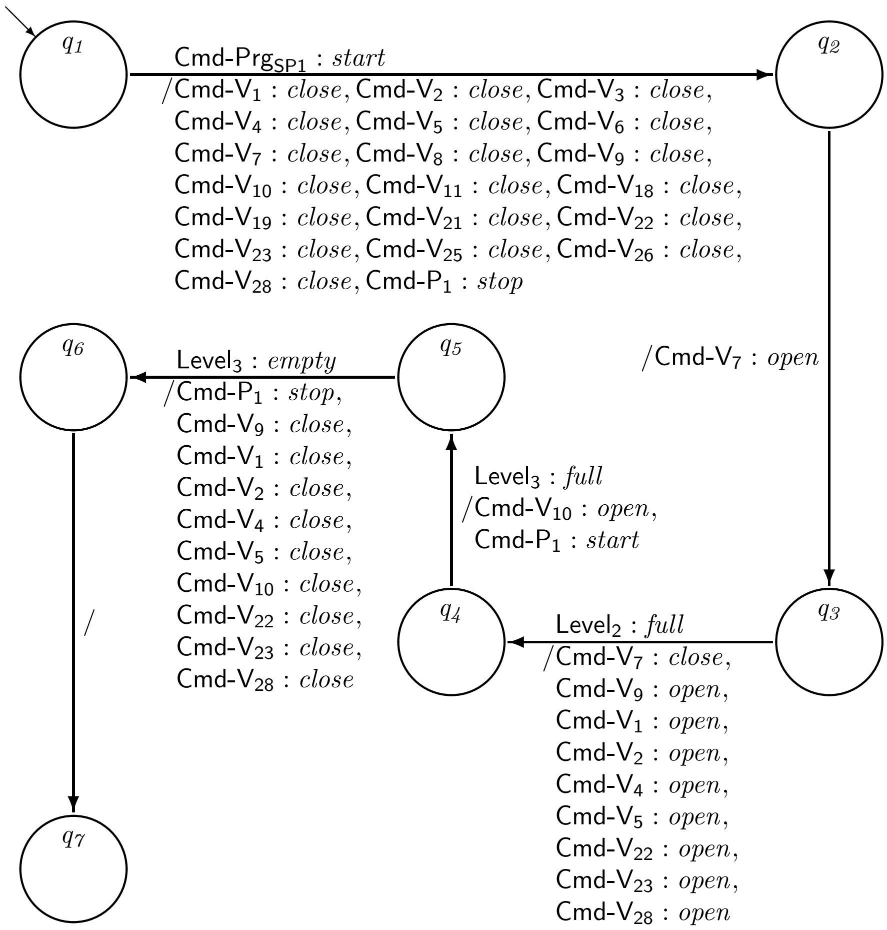

Figure 46: Transition diagram of Prg_{SP1}.

2.2.13 Program Prg_{SP2}

This program is used for rinsing the outer ring pipe with water from tank 6. using pump 1. First tank 6 is filled with fresh water via valve 14. If tank 6 is full, valve 14 is closed, and the valves 19, 23, 22, 1, 2, 4, 5, and 28 are opened. Then pump 1 is started, and water is flowing through these valves. If tank 6 is empty, the pump is stopped, and all valves are closed.

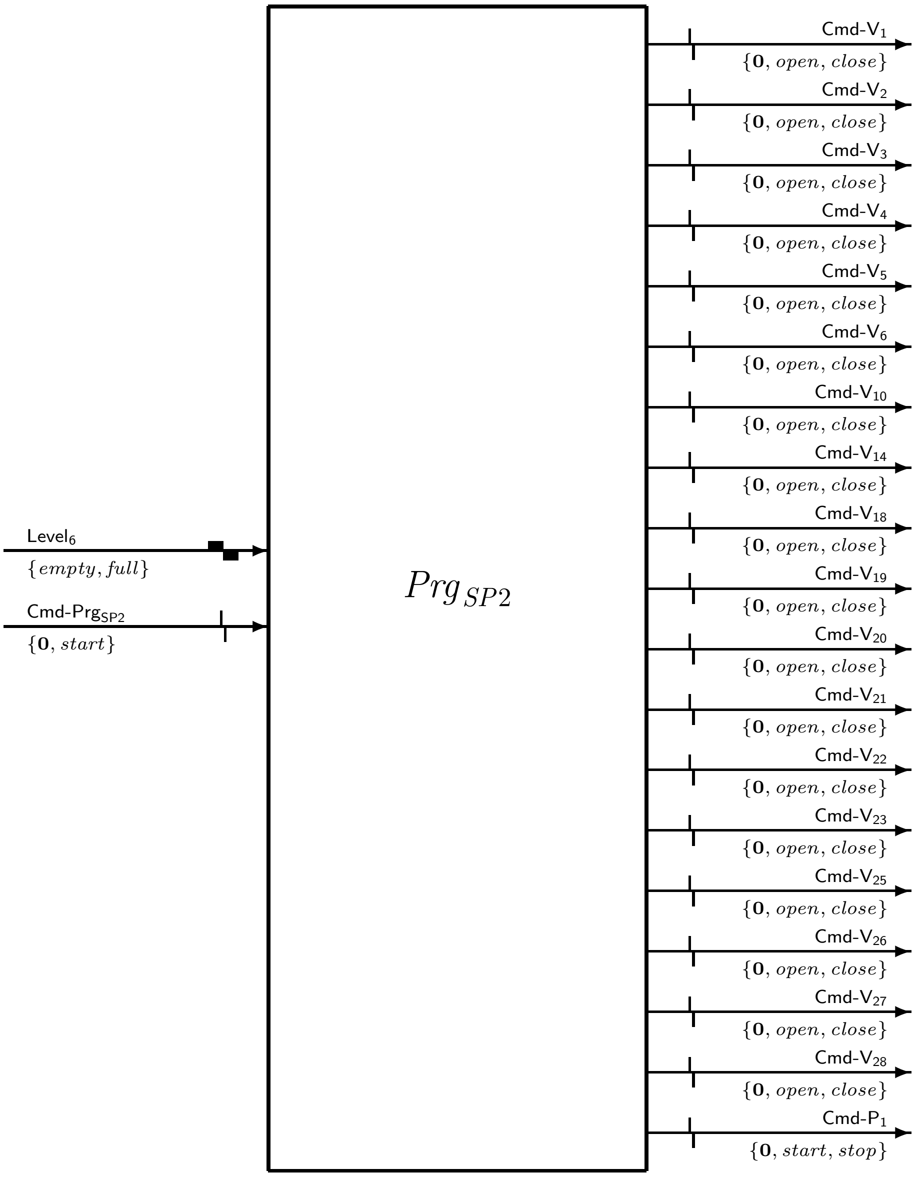

Figure 47: Block diagram of Prg_{SP2}.

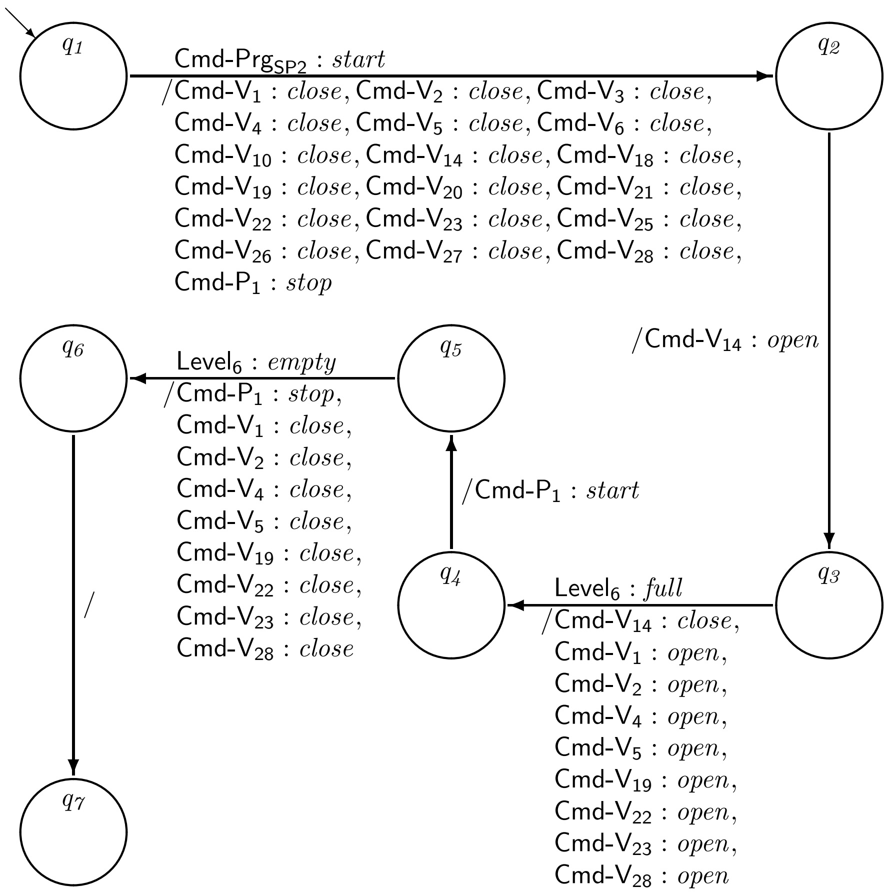

Figure 48: Transition diagram of Prg_{SP2}.

3 The Properties

We want to check some safety properties of the control programs. The following ones have to be checked for each of the control programs:

- Whenever a control program terminates, all valves are closed.
- Whenever a control program terminates, the pumps are switched off.
- Whenever a control program terminates, the heater is switched off.

If these three properties are valid, we are sure that the initial plant state "all valves closed, pumps off, heater off" also holds after the termination of a control program. If we furthermore assume that two programs never run simultaneously, we know that during normal operation of the plant the initial state at the start of a program is always "all valves closed, pumps off, heater off". This allows us to do compositional reasoning, i.e., when checking one program's properties, we do not have to consider the actions other programs might have done before or do simultaneously.

However, the states of the tanks are *not* preserved by the programs. We have to take this into account when verifying the other properties.

To prevent the heating unit in tank 5 to be damaged by overheating we have the following property:

- Whenever the heater in tank 5 is turned on, the water level of tank 5 is high enough.

We demand that during the heating process steam can leave tank 5 into the condenser only:

- Whenever the heater in tank 5 is working, none of the valves 12, 15, and 16 are open.

Furthermore, the cooling device in the condenser must be provided with cold water during the heating process to prevent a dangerous pressure inside the condenser:

- Whenever the heater in tank 5 is working, valve 13 is open.

The regular pressure of the cooling water supply is not sufficient if all 3 cooling units operate at the same time. The following property prevents this undesired situation:

- At most two cooling units are active simultaneously.

Although the pumps are equipped with a pressure limit switches, we demand:

- Pumps are not pumping against closed valves.

We do not want an uncontrolled flow of water through a tank:

- Each tank's input and output valves are not open simultaneously.

Before we check the properties listed above, we need a framework for which we have a model checking tool.

4 The Transformation to SMV

We transform the DCESs into the input language of the symbolic model checker SMV [McM93, McM92]. The transformation is described in detail in Chapter 7 of [Luk99]. Since the transformation has to be done repeatedly (once for each set of DCESs we need to compose for a property), and since a transformation by hand can be the source of many errors, we use a compiler prototype to perform this step.

The compiler takes the LaTeX [Lam94] source of the DCESs' graphical description (even the source code of this document can be used) and a list of system names (e.g., Prg_{B6A}, $Valve_{19}$, $Valve_{20}$, $Valve_{27}$) as input, and produces the respective SMV code of the parallel interconnection of these systems as output.

Since SMV specifications are written in CTL [CE81] we reformulate the properties in CTL. Now SMV can check the systems against the desired properties.

5 The Verification

For each of the desired properties and each control program involved in the validity of this property we perform the following steps to verify it:

1. Use the LaTeX-to-SMV compiler to generate the SMV code for the DCESs we need to model check the property.

2. Write the property as a CTL formula and insert it into the SMV code using a `SPEC` declaration.

3. Use SMV to verify the property.

We illustrate this in detail with the property "Whenever the heater in tank 5 is turned on, the water level of tank 5 is high enough":

The DCESs we have to inspect are Prg_{B5} (this is the only control program sending commands to the heater) and $Tank_5$ (because we must take a look at its condition output Level_5). Thus, we use the LaTeX-to-SMV compiler to generate the parallel composition of Prg_{B5} and $Tank_5$ as an SMV file.

We formulate the property as a CTL formula:

$$\forall\Box(\mathsf{Cmd\text{-}H} = start \Rightarrow \mathsf{Level}_5 \neq empty)$$

and insert it into the SMV file using the following `SPEC` declaration (note that variable names must be preceded by the module that is writing the variable):

```
SPEC
  AG (PrgB5.CmdH=start -> !Tank5.Level5=empty)
```

The resulting SMV input file is shown in Figure 49. For the sake of brevity we left out the `TRANS` declarations.

Now we use SMV to verify the property. The output of SMV is:

```
-- specification AG (PrgB5.CmdH = start ->
                     !Tank5.Level5 ...  is true
```

Thus, control program Prg_{B5} satisfies the property.

We could have put more than just Prg_{B5} and $Tank_6$ into the parallel composition to verify the property, like the valves 12, 15, 16, or the heater. This is not necessary, since these additional systems are not needed. The outputs of these systems are defined as global variables (`StatusV12`, `StatusV15`, `StatusV16`, and `StatusH` in the first `VAR` declaration in Figure 49), and SMV chooses arbitrary values for these variables in each computation step. In other words, we have "chaotic" behavior of the physical devices not contained in the parallel composition. So any behavior of the parallel composition of Prg_{B5}, $Tank_6$, and some more systems is also a behavior of the parallel composition of just these two systems.

We are only investigating properties that describe all possible behaviors, i.e., CTL formulae of the form $\forall\Box(\varphi)$. Thus, we can always try to compose a small set of systems, and if the verification fails due to a behavior that is not a behavior of the complete system, we can add more systems to the parallel composition until we succeed.

Now we start to verify if the control programs fulfill our list of properties. In the following we will list for each property which systems have to be composed, the respective CTL formula (in SMV syntax), and the verification result obtained by SMV. We also note why a certain verification fails and what the consequences are.

```
MODULE main

VAR  StatusV12: {closed,open};  StatusV15: {closed,open};
     StatusV16: {closed,open};  StatusH: {off,on};
     CmdPrgB5: {0,start};

VAR  conditionsymbols: {closed,open,off,on,empty,half,full};

VAR  Tank5: Tank5_module(StatusV12,StatusV15,StatusV16,StatusH);
     PrgB5: PrgB5_module(Tank5.Level5,CmdPrgB5);

INIT  CmdPrgB5=0

SPEC  AG (PrgB5.CmdH=start -> !Tank5.Level5=empty)


MODULE Tank5_module(StatusV12,StatusV15,StatusV16,StatusH)

VAR  state : {q_0,q_1,q_2};

INIT  state in {q_0}

DEFINE  Level5 := case
                    state=q_0: empty;
                    state=q_1: half;
                    state=q_2: full;
                  esac;

TRANS  [...]


MODULE PrgB5_module(Level5,CmdPrgB5)

VAR  state : {q_1,q_2,q_3,q_4,q_6,q_7,q_8,q_9};

VAR  CmdV12: {0,open,close};  CmdV13: {0,open,close};
     CmdV15: {0,open,close};  CmdV16: {0,open,close};
     CmdH: {0,start,stop};

INIT  state in {q_1}

INIT  CmdV12=0 & CmdV13=0 & CmdV15=0 & CmdV16=0 & CmdH=0

TRANS  [...]
```

Figure 49: SMV input file for the parallel composition of Prg_{B5} and $Tank_5$.

Whenever a control program terminates, all valves are closed. All control programs open at least one valve, so we compose each control program with the valves it sends commands to. We check if the status variable of each valve has the value `closed` when the control program is at a terminating state. Since we assume that all valves are closed when a program is started and two programs do not run simultaneously, we do not need to consider valves that are not switched by the program. Programs Prg_{B5}, Prg_{B6}, Prg_{B7} have two terminating states, and the others have one terminating state.

- Prg_{B2}, $Valve_6$, $Valve_7$, $Valve_9$

  ```
  SPEC
    AG (PrgB2.state=q_3 ->
        Valve6.StatusV6=closed & Valve7.StatusV7=closed &
        Valve9.StatusV9=closed)
  ```

  Verification result: `true`

- Prg_{B3}, $Valve_8$, $Valve_9$, $Valve_{10}$, $Valve_{11}$

  ```
  SPEC
    AG (PrgB3.state=q_6 ->
        Valve8.StatusV8=closed & Valve9.StatusV9=closed &
        Valve10.StatusV10=closed & Valve11.StatusV11=closed)
  ```

  Verification result: `true`

- Prg_{B3K}, $Valve_8$, $Valve_9$, $Valve_{10}$, $Valve_{11}$

  ```
  SPEC
    AG (PrgB3K.state=q_9 ->
        Valve8.StatusV8=closed & Valve9.StatusV9=closed &
        Valve10.StatusV10=closed & Valve11.StatusV11=closed)
  ```

  Verification result: `true`

- Prg_{B3U}, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_8$, $Valve_{10}$, $Valve_{11}$, $Valve_{19}$, $Valve_{21}$, $Valve_{22}$, $Valve_{23}$

  ```
  SPEC
    AG (PrgB3U.state=q_6 ->
        Valve1.StatusV1=closed & Valve2.StatusV2=closed &
        Valve3.StatusV3=closed & Valve8.StatusV8=closed &
        Valve10.StatusV10=closed) & Valve11.StatusV11=closed) &
        Valve19.StatusV19=closed) & Valve21.StatusV21=closed) &
        Valve22.StatusV22=closed) & Valve23.StatusV23=closed)
  ```

  Verification result: `true`

- Prg_{B5}, $Valve_{12}$, $Valve_{13}$, $Valve_{15}$, $Valve_{16}$

  ```
  SPEC
    AG (PrgB5.state=q_9 ->
        Valve12.StatusV12=closed & Valve13.StatusV13=closed &
        Valve15.StatusV15=closed & Valve16.StatusV16=closed)
  ```

  Verification result: `true`

  ```
  SPEC
    AG (PrgB5.state=q_6 ->
        Valve12.StatusV12=closed & Valve13.StatusV13=closed &
        Valve15.StatusV15=closed & Valve16.StatusV16=closed)
  ```

  Verification result: `false`

  If control program Prg_{B5} enters state q_6 this indicates that during the evaporation process the water level in tank 5 has fallen too low. Therefore the heater is switched off, and the program stops. The verification result however shows that valve 13, which was opened earlier when entering state q_3, is still open. This is indicated by the following trace generated by SMV (we omit some less interesting output):

  ```
  state 1.1:
  Valve12.StatusV12 = closed
  Valve13.StatusV13 = closed
  Valve15.StatusV15 = closed
  Valve16.StatusV16 = closed
  PrgB5.state = q_1

  state 1.2:
  CmdPrgB5 = start
  Valve12.StatusV12 = open
  PrgB5.state = q_2

  state 1.3:
  Valve12.StatusV12 = closed
  Valve13.StatusV13 = open
  PrgB5.state = q_3

  state 1.4:
  Level5 = half
  PrgB5.state = q_4

  state 1.5:
  PrgB5.state = q_6
  ```

Although this is a violation of our property, it is not that important, since we can assume that after this error state q_6 is reached, there is some intervention by the user such that other programs will not be started before the plant is reseted to its initial state, and this reset includes the closing of all valves.

Furthermore, a permanent open state of valve 13 does not endanger the plant, since valve 13 only controls the cooling water supply for the condenser. If, e.g., valve 12 would remain open permanently, an overflow of tank 5 might occur.

- Prg_{B6}, $Valve_{14}$, $Valve_{29}$

```
SPEC
  AG (PrgB6.state in {q_3,q_5} ->
      Valve14.StatusV14=closed & Valve29.StatusV29=closed)
```

Verification result: `true`

- Prg_{B6A}, $Valve_{19}$, $Valve_{20}$, $Valve_{27}$

```
SPEC
  AG (PrgB6A.state=q_3 ->
      Valve19.StatusV19=closed & Valve20.StatusV20=closed &
      Valve27.StatusV27=closed)
```

Verification result: `true`

- Prg_{B6S}, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{10}$, $Valve_{18}$, $Valve_{19}$, $Valve_{20}$, $Valve_{21}$, $Valve_{22}$, $Valve_{23}$, $Valve_{24}$, $Valve_{26}$, $Valve_{27}$

```
SPEC
  AG (PrgB6S.state=q_6 ->
      Valve1.StatusV1=closed & Valve2.StatusV2=closed &
      Valve3.StatusV3=closed & Valve4.StatusV4=closed &
      Valve5.StatusV5=closed & Valve6.StatusV6=closed &
      Valve10.StatusV10=closed & Valve18.StatusV18=closed &
      Valve19.StatusV19=closed & Valve20.StatusV20=closed &
      Valve21.StatusV21=closed & Valve22.StatusV22=closed &
      Valve23.StatusV23=closed & Valve24.StatusV24=closed &
      Valve26.StatusV26=closed & Valve27.StatusV27=closed)
```

Verification result: `true`

- Prg_{B6U}, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{19}$, $Valve_{20}$, $Valve_{21}$, $Valve_{24}$, $Valve_{25}$, $Valve_{27}$, $Valve_{28}$

```
SPEC
  AG (PrgB6U.state=q_6 ->
      Valve4.StatusV4=closed & Valve5.StatusV5=closed &
      Valve6.StatusV6=closed & Valve19.StatusV19=closed &
      Valve20.StatusV20=closed & Valve21.StatusV21=closed &
      Valve24.StatusV24=closed & Valve25.StatusV25=closed &
      Valve28.StatusV28=closed & Valve27.StatusV27=closed)
```

Verification result: `true`

- Prg_{B7}, $Valve_{17}$

```
SPEC
  AG (PrgB7.state in {q_3,q_5} -> Valve17.StatusV17=closed)
```

Verification result: `true`

- Prg_{B7U}, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_{18}$, $Valve_{19}$, $Valve_{21}$, $Valve_{22}$, $Valve_{23}$, $Valve_{26}$

```
SPEC
  AG (PrgB7U.state=q_6 ->
      Valve1.StatusV1=closed & Valve2.StatusV2=closed &
      Valve3.StatusV3=closed & Valve18.StatusV18=closed &
      Valve19.StatusV19=closed & Valve21.StatusV21=closed &
      Valve22.StatusV22=closed & Valve23.StatusV23=closed &
      Valve26.StatusV26=closed)
```

Verification result: `true`

- Prg_{SP1}, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_7$, $Valve_8$, $Valve_9$, $Valve_{10}$, $Valve_{11}$, $Valve_{18}$, $Valve_{19}$, $Valve_{21}$, $Valve_{22}$, $Valve_{23}$, $Valve_{25}$, $Valve_{26}$, $Valve_{28}$

```
SPEC
  AG (PrgSP1.state=q_7 ->
      Valve1.StatusV1=closed & Valve2.StatusV2=closed &
      Valve3.StatusV3=closed & Valve4.StatusV4=closed &
      Valve5.StatusV5=closed & Valve6.StatusV6=closed &
      Valve7.StatusV7=closed & Valve8.StatusV8=closed &
      Valve9.StatusV9=closed & Valve10.StatusV10=closed &
      Valve11.StatusV11=closed & Valve18.StatusV18=closed &
      Valve19.StatusV19=closed & Valve21.StatusV21=closed &
      Valve22.StatusV22=closed & Valve23.StatusV23=closed &
      Valve25.StatusV25=closed & Valve26.StatusV26=closed &
      Valve28.StatusV28=closed)
```

Verification result: `true`

This result was difficult to obtain due to the state explosion problem; the verification took about 23 hours on a Sun UltraSPARC system running at 167 MHz and consumed about 340 Megabytes of main memory.

In this special case avoiding this problem is easy. We can split up the conjunction in the formula, thus yielding the task of checking the formula

```
AG (PrgSP1.state=q_7 -> Valvei.StatusVi=closed)
```

on the parallel composition of Prg_{SP1} and $Valve_i$ for each $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 18, 19, 21, 22, 23, 25, 26, 28\}$. Each of these 19 checks succeeds within a fraction of a second.

- Prg_{SP2}, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{10}$, $Valve_{14}$, $Valve_{18}$, $Valve_{19}$, $Valve_{20}$, $Valve_{21}$, $Valve_{22}$, $Valve_{23}$, $Valve_{25}$, $Valve_{26}$, $Valve_{27}$, $Valve_{28}$

```
SPEC
  AG (PrgSP2.state=q_7 ->
      Valve1.StatusV1=closed & Valve2.StatusV2=closed &
      Valve3.StatusV3=closed & Valve4.StatusV4=closed &
      Valve5.StatusV5=closed & Valve6.StatusV6=closed &
      Valve10.StatusV10=closed & Valve14.StatusV14=closed &
      Valve18.StatusV18=closed & Valve19.StatusV19=closed &
      Valve20.StatusV20=closed & Valve21.StatusV21=closed &
      Valve22.StatusV22=closed & Valve23.StatusV23=closed &
      Valve25.StatusV25=closed & Valve26.StatusV26=closed &
      Valve27.StatusV27=closed & Valve28.StatusV28=closed)
```

  Verification result: `true`

Whenever a control program terminates, the pumps are switched off. There are five control programs which send commands to pump 1: Prg_{B3U}, Prg_{B6S}, Prg_{B7U}, Prg_{SP1}, and Prg_{SP2}. We compose each of these programs with $Pump_1$ and check if the status variable of pump 1 has the value `off` when the control program is at its terminating state.

- Prg_{B3U}, $Pump_1$

```
SPEC
  AG (PrgB3U.state=q_6 -> Pump1.StatusP1=off)
```

  Verification result: `true`

- Prg_{B6S}, $Pump_1$

```
SPEC
  AG (PrgB6S.state=q_6 -> Pump1.StatusP1=off)
```

  Verification result: `true`

- Prg_{B7U}, $Pump_1$

```
SPEC
  AG (PrgB7U.state=q_6 -> Pump1.StatusP1=off)
```

  Verification result: `true`

- Prg_{SP1}, $Pump_1$

```
SPEC
  AG (PrgSP1.state=q_7 -> Pump1.StatusP1=off)
```

  Verification result: `true`

- Prg_{SP2}, $Pump_1$

```
SPEC
  AG (PrgSP2.state=q_7 -> Pump1.StatusP1=off)
```

  Verification result: `true`

Analogously, we check this for program Prg_{B6U}, which is the only program using pump 2.

- Prg_{B6U}, $Pump_2$

```
SPEC
  AG (PrgB6U.state=q_6 -> Pump2.StatusP2=off)
```

  Verification result: `true`

Whenever a control program terminates, the heater is switched off. Only control program Prg_{B5} uses the heater. We compose Prg_{B5} and *Heater* and check if the heater's status variable is `off` when the program is in one of its final states.

- Prg_{B5}, *Heater*

```
SPEC
  AG (PrgB5.state in {q_6,q_9} -> Heater.StatusH=off)
```

  Verification result: `true`

Now we know that whenever a control program terminates, all valves are closed (with one exception listed above), and the pumps and the heater are switched off, provided this was also the case when the control program was started. We further assume that two programs are not running simultaneously. This allows us to verify the rest of our properties.

Whenever the heater in tank 5 is turned on, the water level of tank 5 is high enough. Only control program Prg_{B5} uses the heater. We compose Prg_{B5} and $Tank_5$ and check if the tank's water level does not have the `empty` value when the Cmd-H : *start* event occurs. We do not need to include the heater in the composition, since we are only interested in the starting command generated by the control program.

- Prg_{B5}, $Tank_5$

  ```
  SPEC
    AG (PrgB5.CmdH=start -> !Tank5.Level5=empty)
  ```

  Verification result: `true`

Whenever the heater in tank 5 is working, none of the valves 12, 15, and 16 are open. Again, only control program Prg_{B5} uses the heater. We compose Prg_{B5}, $Heater$, $Valve_{12}$, $Valve_{15}$, and $Valve_{16}$, and check the status variables of the valves if the heater status is `on`.

- Prg_{B5}, $Heater$, $Valve_{12}$, $Valve_{15}$, $Valve_{16}$

  ```
  SPEC
    AG (Heater.StatusH=on ->
        Valve12.StatusV12=closed &
        Valve15.StatusV15=closed &
        Valve16.StatusV16=closed)
  ```

  Verification result: `true`

Whenever the heater in tank 5 is working, valve 13 is open. We compose Prg_{B5}, $Heater$, and $Valve_{13}$, and check the status variable of valve 13 if the heater status is `on`.

- Prg_{B5}, $Heater$, $Valve_{13}$

  ```
  SPEC
    AG (Heater.StatusH=on -> Valve13.StatusV13=open)
  ```

  Verification result: `true`

At most two cooling units are active simultaneously. Each of the three cooling units is active when the valve through which the cooling water flows is open. These are the valves 13, 17, and 29. Since there is no control program which sends commands to all three valves and since two programs are not running simultaneously, this property is trivially fulfilled.

Pumps are not pumping against closed valves. We show that whenever a pump is working, the water flowing through the pump can leave the system via valve 26 or valve 28, or is pumped into tank 1 or tank 2.

For pump 1, this leads to four possible paths:

1. From pump 1 through valves 22 and 26.
2. From pump 1 through valves 22, 1, and 3 into tank 1.
3. From pump 1 through valves 22, 1, 2, 4, and 6 into tank 2.
4. From pump 1 through valves 22, 1, 2, 4, 5, and 28.

This yields the SMV specification show below. Pump 1 is used by the control programs Prg_{B3U}, Prg_{B6S}, Prg_{B7U}, Prg_{SP1}, and Prg_{SP2}.

- Prg_{B3U}, $Pump_1$, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{22}$, $Valve_{26}$, $Valve_{28}$

  ```
  SPEC
    AG (Pump1.StatusP1=on -> Valve22.StatusV22=open &
        (Valve26.StatusV26=open | Valve1.StatusV1=open &
         (Valve3.StatusV3=open |
          Valve2.StatusV2=open & Valve4.StatusV4=open &
           (Valve6.StatusV6=open |
            Valve5.StatusV5=open & Valve28.StatusV28=open))))
  ```

  Verification result: `true`

- Prg_{B6S}, $Pump_1$, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{22}$, $Valve_{26}$, $Valve_{28}$

  `SPEC`
  (same as above)

  Verification result: `true`

- Prg_{B7U}, $Pump_1$, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{22}$, $Valve_{26}$, $Valve_{28}$

  `SPEC`
  (same as above)

Verification result: `true`

- Prg_{SP1}, $Pump_1$, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{22}$, $Valve_{26}$, $Valve_{28}$

  SPEC
  (same as above)

  Verification result: `true`

- Prg_{SP2}, $Pump_1$, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{22}$, $Valve_{26}$, $Valve_{28}$

  SPEC
  (same as above)

  Verification result: `true`

For pump 2, this also leads to four possible paths:

1. From pump 2 through valves 25 and 28.
2. From pump 2 through valves 25, 5, and 6 into tank 2.
3. From pump 2 through valves 25, 5, 4, 2, and 3 into tank 1.
4. From pump 2 through valves 25, 5, 4, 2, 1, and 26.

This yields the SMV specification show below. Pump 2 is only used by control program Prg_{B6U}.

- Prg_{B6U}, $Pump_2$, $Valve_1$, $Valve_2$, $Valve_3$, $Valve_4$, $Valve_5$, $Valve_6$, $Valve_{25}$, $Valve_{26}$, $Valve_{28}$

```
SPEC
  AG (Pump2.StatusP2=on -> Valve25.StatusV25=open &
      (Valve28.StatusV28=open | Valve5.StatusV5=open &
       (Valve6.StatusV6=open |
        Valve4.StatusV4=open & Valve2.StatusV2=open &
         (Valve3.StatusV3=open |
          Valve1.StatusV1=open & Valve26.StatusV26=open))))
```

  Verification result: `true`

Each tank's input and output valves are not open simultaneously. This property ensures that we have a controlled flow of liquids in the plant, i.e., no program lets water flow directly through a tank without storing it first in that tank. For all tanks, we check this property on each program that is controlling at least one of the tank's input valves and one of the tank's output valves.

Tank 1

- Prg_{B3U}, $Valve_3$, $Valve_8$

  ```
  SPEC
    AG !(Valve3.StatusV3=open & Valve8.StatusV8=open)
  ```

  Verification result: `true`

- Prg_{SP1}, $Valve_3$, $Valve_8$

  ```
  SPEC
    AG !(Valve3.StatusV3=open & Valve8.StatusV8=open)
  ```

  Verification result: `true`

Tank 2

- Prg_{B2}, $Valve_6$, $Valve_7$, $Valve_9$

  ```
  SPEC
    AG !((Valve6.StatusV6=open | Valve7.StatusV7=open) &
         Valve9.StatusV9=open)
  ```

  Verification result: `true`

- Prg_{SP1}, $Valve_6$, $Valve_7$, $Valve_9$

  ```
  SPEC
    AG !((Valve6.StatusV6=open | Valve7.StatusV7=open) &
         Valve9.StatusV9=open)
  ```

  Verification result: `true`

Tank 3

- Prg_{B3}, $Valve_8$, $Valve_9$, $Valve_{10}$, $Valve_{11}$

  ```
  SPEC
    AG !((Valve8.StatusV8=open | Valve9.StatusV9=open) &
         (Valve10.StatusV10=open | Valve11.StatusV11=open))
  ```

  Verification result: `true`

- Prg_{B3K}, $Valve_8$, $Valve_9$, $Valve_{10}$, $Valve_{11}$

  ```
  SPEC
    AG !((Valve8.StatusV8=open | Valve9.StatusV9=open) &
         (Valve10.StatusV10=open | Valve11.StatusV11=open))
  ```

  Verification result: `true`

- Prg_{B3U}, $Valve_8$, $Valve_{10}$, $Valve_{11}$

  ```
  SPEC
    AG !(Valve8.StatusV8=open &
         (Valve10.StatusV10=open | Valve11.StatusV11=open))
  ```

  Verification result: `true`

- Prg_{SP1}, $Valve_8$, $Valve_9$, $Valve_{10}$, $Valve_{11}$

  ```
  SPEC
    AG !((Valve8.StatusV8=open | Valve9.StatusV9=open) &
         (Valve10.StatusV10=open | Valve11.StatusV11=open))
  ```

  Verification result: `false`

  The trace generated by SMV shows that valve 9 is opened in the step from state q_3 to state q_4, and valve 10 is opened in the following step to state q_5. This violates our property, since water is now flowing through tank 3. Since program Prg_{SP1} is used for maintenance purposes only and is not a part of the normal production cycle, this is not considered to be an error.

Tank 4 There is no control program that uses valve 11 *and* valve 12, so nothing has to be checked, and the property is trivially fulfilled.

Tank 5

- Prg_{B5}, $Valve_{12}$, $Valve_{15}$, $Valve_{16}$

  ```
  SPEC
    AG !(Valve12.StatusV12=open &
         (Valve15.StatusV15=open | Valve16.StatusV16=open))
  ```

  Verification result: `true`

Tank 6

- Prg_{SP2}, $Valve_{14}$, $Valve_{19}$, $Valve_{20}$, $Valve_{27}$

  ```
  SPEC
    AG !(Valve14.StatusV14=open &
         (Valve19.StatusV19=open | Valve20.StatusV20=open |
          Valve27.StatusV27=open))
  ```

  Verification result: `true`

Tank 7 There is no control program that uses valve 15 *and* valve 18, so nothing has to be checked, and the property is trivially fulfilled.

6 Conclusions

A discrete, untimed model of the batch plant and its control programs has been used to verify some safety properties. These properties help to ensure that the control programs do not cause damages to the plant's devices.

We used discrete condition/event systems (DCESs) to model the valves, the tanks, the pumps, the heater, and the 13 control programs. For each property, we used a compiler prototype to transform the DCESs required to check the property into SMV input, and provided SMV with the property in CTL syntax.

The formal verification reveals that all properties are fulfilled, with two exceptions. In these cases SMV shows the trace of a computation prefix leading to the state where the error occurs. This trace provides a convenient way of finding the error in the control program that causes the violation of the property. However, the two violations detected will not cause any damages to the plant; they can merely be regarded as small design inaccuracies.

Further work lies in automated techniques to determine which DCESs have to be composed for a given formula, in methods for reducing time and space consumption (like the splitting of a formula for several verification steps), and in using other levels of abstraction for the models.

Acknowledgments

Thanks to Jan Lukoschus who programmed large parts of the LaTeX-to-SMV compiler prototype.

References

[CE81] E.M. Clarke and E.A. Emerson. Synthesis of synchronisation skeletons for branching time temporal logic. In D. Kozen, editor, *Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[CES83] E.M. Clarke, E.A. Emerson, and E. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *10th ACM symposium of Programming Languages*. ACM Press, 1983.

[Kow98] S. Kowalewski. Description of VHS case study 1 "Experimental Batch Plant". Draft. University of Dortmund, Germany, July 1998.

[Lam94] L. Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, second edition, 1994.

[Luk99] B. Lukoschus. Composition and verification of condition/event systems. Technical Report TR-ST-99-1, Lehrstuhl für Software-Technologie, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, May 1999.

[McM92] K.L. McMillan. *The SMV system*. Carnegie-Mellon University, February 1992. Draft version describing SMV revision 2.2.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1982.

[SK91] R.S. Sreenivas and B.H. Krogh. On condition/event systems with discrete state realizations. In *Discrete Event Dynamic Systems: Theory and Applications 1*, pages 209–236. Kluwer Academic Publishers, Boston, USA, 1991.