Specification with Moore machines: the steam boiler control example

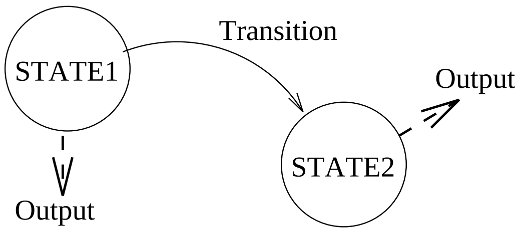
Victor Yodaiken
Department of Computer Science
New Mexico Institute of Technology
Socorro, NM 87801, USA
yodaiken@chelm.nmt.edu

Abstract

A formalization of the steam-boiler specification in terms of Moore machines.

1 Introduction

Abrial[Abr94b] provides an informal specification of a control program for a steam-boiler and asks for examples of "formal" specifications. In this paper, I will show how the steam-boiler program can be specified as a Moore machine [Moo64, HU79] — a state machine that associates an output value with each state.



In order to allow for a concise and sufficiently abstract specification, Moore machines will be defined by recursive equations. Define a distinguished variable σ that ranges over transition sequences and treat Moore machines as functions implicitly depending on this variable. Now write $\mathsf{M}()$ for the output of state machine M in the state determined by σ. Say $\mathsf{M}()$ is the *current state output* of M. Given this convention, it is natural to define the operation of state machines by defining the initial state output and an algorithm for computing the *next state* output from the next transition symbol and the current state output (possibly of several previously defined state machines). Since σ ranges over all finite sequences of state transitions, the statement $\mathsf{M}() > 0$ is true if and only if M outputs a non-zero value in every reachable state. The statement $\mathsf{M}() > 0 \rightarrow \mathsf{N}() > 0$ is true if and only if every sequence that drives M to a state where it outputs a non-zero value also drives N to a state where it outputs a non-zero value.

Write $\textsc{Start}\ \mathsf{M}()$ for the output of M when σ is the null sequence — thus $\textsc{Start}\ \mathsf{M}()$ is the initial state output of M. Write $\textsc{After}\, z{\cdot}\mathsf{M}()$ for the output of M in the state reached by appending sequence z to σ. Thus, $\textsc{After}\, z{\cdot}\mathsf{M}()$ is the output of M *after* z. Given an alphabet of inputs A, we can then define the operation of M completely by $\textsc{Start}\ \mathsf{M}() = k_0$ and $(\forall a \in A)\textsc{After}\langle a\rangle{\cdot}\mathsf{M}() = f(a, \mathsf{M}(), \mathsf{M}_1(), \ldots \mathsf{M}_n())$ where $\mathsf{M}_1, \ldots M_n$ have been previously defined. This technique makes it convenient to define and interconnect state machines that are too large and complex for traditional methods. The technique is made precise in the appendix and the mathematical basis in automata presentations and products can be found in [Yod95] as well as [Yod91a], and [Yod91b]. For our purposes here, however, this informal introduction should be sufficient.

The specification carried out here follows the informal specification closely, without simplifications (except in notation) or additional assumptions as these would defeat the purpose of the exercise. I have also taken pains to not specify any more of the implementation of the control program than is explicit in the informal specification. In particular, the compositional structure of the program is not specified here even though the method of modeling state machines by recursive equations offers a powerful means for composing specifications of concurrent modules. The informal specification does not constrain how the program is to be implemented, and so this specification will not either.

Readers who are accustomed to the "formal methods" literature need to be a little cautious about not assuming too much here. The method is based on ordinary, unimproved, deterministic state machines, yet no dire consequences for expressiveness or abstraction materialize. Furthermore, even though the method is not "formal" in the sense of formal logic, it is mathematically well defined and as rigorous as necessary.

1.1 Outline and notation

The remainder of this paper falls in 4 sections. Section 2 defines the input and output alphabets introduces the concept of state machines acting as *instruments* and then states the highest level specification. Section 3 defines instrument state machines that track the system dynamics and operating state (*mode*). Section 4 specifies the high level operation of the system. And Section 5 defines the remaining instrument state machines. There is an appendix that, briefly, reviews the mathematical basis of the technique used here. Figure 4 on page 6 provides a cross reference with the page number and definition number of each defined state machine.

Some notation. State machines in this paper are always indicated by a special typeface: M is a state machine, M is not. Many of the state machines used here will output maps rather than just scalar values. If the outputs of M are maps , then $\mathsf{M}()(x)$ will be the result of applying the current state output of M to x. To simplify notation, I will write $\mathsf{M}(x)$ instead of $\mathsf{M}()(x)$. For certain functions I will write $f.x$ in place of $f(x)$. This notation is reserved for functions that represent blocks of messages so that $b.x$ can be the contents of message x in block b. Finally, following [GKP89], the notation $[e]$ for a true-or-false expression e will have value 1 if e is true and 0 otherwise. For example $\sum_{i=1}^{n}[\mathsf{M}(i) > 0]$ is counts the number of integers $0 < i \leq n$ where $\mathsf{M}(i) > 0$.

2 The basic approach

2.1 Inputs and Outputs

The informal specification tells us that the control program follows a simple cycle: every 5 seconds a block of messages is received from the physical devices and before the next block is received the program must calculate and transmit a block of messages to control the devices.

Throwing detail to the winds, we can sketch the operation of the control program in figure 1. The transition alphabet of the control program state machine then may be constructed of

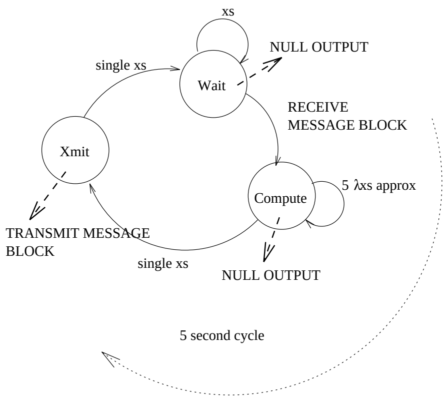


Figure 1: A sketch of control program operation

symbols corresponding to the action of receiving a message block and a symbol denoting the passage of a sufficiently fine moment of physical time. The output alphabet consists of symbols corresponding to transmission of output message blocks and a symbol denoting output states where no message block is transmitted.

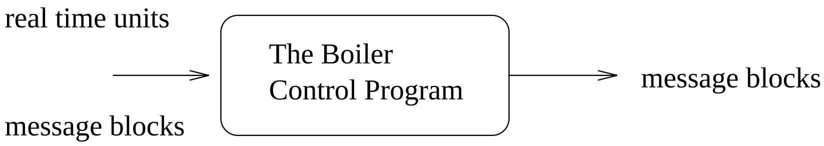


Figure 2: The inputs and outputs of the program

The input marking the passage of time will be called xs. We suppose that there are λ xs transitions per second of physical-time where λ is a parameter of the specification. It is possible to pick some concrete unit of time, ms would almost certainly be sufficient, but the informal specification does not require such detail so, on principle, we leave the unit to be specified when more design decisions have been made. Note that the time transitions do not represent the ticking of a clock device: physical time passes without needing help from a computer chip. One can think of the time transitions as marking the end of sampling periods in a discretization of system operation — e.g. the view provided by an imaginary and perfectly accurate logic analyzer.

According to the informal specification, both receiving and transmitting involve simultaneous transfer of a block of messages. We can model these "blocks" of messages as functions. A block function assigns a value to each message name. To make the specifications more readable we will write $a.x$ rather than $a(x)$ when a is a message block. For example, $a.\textsc{PumpState}_\mathrm{n}$ in place of $a(\textsc{PumpState}_\mathrm{n})$.

If the message is not present in the transmission, the name maps to $\mathbb{F}$ (false). If the message is present there are two possibilities. The one possibility is that the message is a simple command or acknowledgment so that its presence or absence is the only datum of interest. In this case, the block function will assign the value $\mathbb{T}$ (true) to the message name. For example, if a is an input block function then $a.\textsc{PumpControlRepaired}_{\mathrm{n}} = \mathbb{T}$ if and only if a includes a $\textsc{PumpControlRepaired}_{\mathrm{n}}$ message. The other possibility is that the message contains some data value. In this case the block function will assign the data value to the message name. For example, the input message $\textsc{PumpState}_{\mathrm{n}}$ carries the status of the pump device and has value either *on* or *off*. Thus, $a.\textsc{PumpState}_{\mathrm{n}} \in \{on, off, \mathbb{F}\}$. Of course, it is essential that $\mathbb{F}$ is not a valid message value for any message type, but this is easily arranged.

Definition: 1 InputMessageNames = {
$\textsc{Stop}, \textsc{SteamBoilerWaiting}$,
$\textsc{PhysicalUnitsReady}$,
$\textsc{Level}$, $\textsc{Steam}$,
$\textsc{PumpState}_{\mathrm{n}}$, $\textsc{PumpControlState}_{\mathrm{n}}$,
$\textsc{PumpRepaired}_{\mathrm{n}}$, $\textsc{PumpControlRepaired}_{\mathrm{n}}$,
$\textsc{LevelRepaired}$, $\textsc{SteamRepaired}$,
$\textsc{PumpControlFailureAck}_{\mathrm{n}}$,
$\textsc{LevelFailureAck}$, $\textsc{SteamOutcomeFailureAck}$,
$\textsc{PumpFailureAck}_{\mathrm{n}} : n \in PumpNames\}$.

OutputMesageNames=
{ $\textsc{Mode}, \textsc{LevelFailureDetection}$, $\textsc{ProgramReady}$, $\textsc{Valve}$,
$\textsc{OpenPump}_{\mathrm{n}}$, $\textsc{ClosePump}_{\mathrm{n}}$,
$\textsc{SteamFailureDetection}$, $\textsc{LevelFailureDetection}$,
$\textsc{PumpFailureDetection}_{\mathrm{n}}$, $\textsc{PumpControlFailureDetection}_{\mathrm{n}}$,
$\textsc{LevelRepairedAck}$, $\textsc{SteamRepairedAck}$,
$\textsc{PumpRepairedAck}_{\mathrm{n}}$,
$\textsc{PumpControlRepairedAck}_{\mathrm{n}} : n \in PumpNames\}$

Definition: 2 The input alphabet $A_{transition}$ consists of the symbol xs and the set of functions from $InputMessageNames$ to $DataValues \cup \{\mathbb{T}, \mathbb{F}\}$. The output alphabet A_{output} consists of the symbol $\mathbb{F}$ (for null output) and the set of functions from $OutputMessageNames$ to $\{\mathbb{T}, \mathbb{F}\}$.

2.2 Instrument state machines

The challenge is to fill in the details called for in the informal specification, without overly constraining the implementation. After all, the control program could be implemented as a circuit or as a collection of communicating processes under a time-shared operating system. In order to get the required abstraction, we will make extensive use of *instrument* state machines. In essence, an instrument state machine, $\mathsf{I\_N}$ for a state machine M encapsulates M and is able to monitor its inputs, outputs, and transition behavior.

As an example of an instrument, consider a state machine that counts how long it has been since a message block was received. The simple (infinite state) instrument $\mathsf{I\_SinceRecv}$ may be defined by $\textsc{Start}\ \mathsf{I\_SinceRecv}() = 0$ to define the initial output, and $\textsc{After}\langle a\rangle{\cdot}\mathsf{I\_SinceRecv}()$

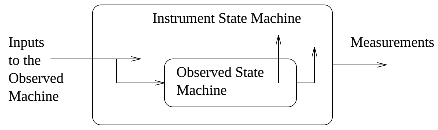


Figure 3: A state machine used as an instrument

$(1 + \mathsf{I\_SinceRecv}()) * [a = xs]$ so that if $a \neq xs$ the next output of $\mathsf{I\_SinceRecv}$ will be 0 and if $a = xs$ the next value will be 1 plus the current output.

Now consider an instrument that tells us how long it has been since a message block was transmitted by the program. Suppose that the state machine modeling the program is C (for **c**ontrol program). And suppose that the instrument $\mathsf{I\_Xout}$ has output b when C is transmitting message block b and has output $\mathbb{F}$ when C is not transmitting any message block. Then we can define $\mathsf{I\_SinceXmit}$ by $\textsc{Start}\ \mathsf{I\_SinceXmit}() = 0$ and define $\mathsf{I\_SinceXmit}()$ after a to be equal to 0 if $\mathsf{I\_Xout}() = \mathbb{F}$, to be unchanged if $\mathsf{I\_Xout}() \neq \mathbb{F}$ but $a \neq xs$, and to be otherwise incremented by 1.

We do not suppose that the instruments are components of the program. Instead, we use the instrument state machines to measure the operation of the program: if $\mathsf{I\_SinceXmit}$ sees the same input stream as the program and if it can monitor the outputs of the program plus other state machines also acting as instruments, then $\mathsf{I\_SinceXmit}$ will tell us something about how the program should behave.

Definition: 3

$$\textsc{Start}\ \mathsf{I\_SinceRecv}() = 0$$
$$\textsc{After}\langle a\rangle\cdot\mathsf{I\_SinceRecv}() = (1 + \mathsf{I\_SinceRecv}()) * [a = xs]$$

$$\textsc{Start}\ \mathsf{I\_SinceXmit}() = 0$$
$$\textsc{After}\langle a\rangle\cdot\mathsf{I\_SinceXmit}() = ([a = xs] + \mathsf{I\_SinceXmit}()) * [\mathsf{I\_Xout}() = \mathbb{F}]$$

Definition: 4 $\mathsf{I\_Xout}() \in A_{output}$
and
$\mathsf{I\_Xout}() \neq \mathbb{F} \rightarrow \textsc{After}\langle xs\rangle\cdot\mathsf{I\_Xout}() = \mathbb{F}$

We leave the details of $\mathsf{I\_Xout}$ for a latter stage in the design. For example, suppose that C were a circuit and $\mathsf{C}(p)$ was the instantaneous signal present on pin p. Then, $\mathsf{I\_Xout}$ might be defined as $(\mathsf{C}(d0), \ldots \mathsf{C}(dn))$ in the state where $\mathsf{C}(datastrobe)$ has just become true, and as $\mathbb{F}$ otherwise. Suppose, on the other hand, that C was implemented in some programming language so that $\mathsf{C}(v)$ is the current value of program or machine variable v. In this case, we might define $\mathsf{I\_Xout}()$ to be $\mathsf{C}(x)$ when $program(\mathsf{C}(programcounter)) =$ "$write(x)$" for some location x, and define $\mathsf{I\_Xout}() = \mathbb{F}$ otherwise. The specification we have been asked to formalize does not go to such a level of detail, so we will not either.
The index of instruments follows.

Instrument name	definition
• I_MaxAdj	definition 8 on page 8
• I_MaxCalc	definition 10 on page 9
• I_MinAdj	definition 9 on page 8
• I_MinCalc	definition 10 on page 9
• I_Monitor	definition 20 on page 17
• I_NoFail	definition 16 on page 11
• I_Op	definition 15 on page 11
• I_Permitted	Section definition 4.2 on page 13
• I_PumpStat	definition 19 on page 16
• I_PumpsNeeded	definition 17 on page 14
• I_PumpsNotNeeded	definition 18 on page 15
• I_Raw	definition 7 on page 8
• I_RepairStat	definition 22 on page 18
• I_SinceRecv	definition 3 on page 5
• I_SinceXmit	definition 3 on page 5
• I_TooManyFail	definition 14 on page 11
• I_TransmitError	definition 23 on page 19
• I_Xout	definition 4 on page 5

Figure 4: Index of instruments

2.3 Highest level specification

We can now state the high level specification. We will adopt a top-down strategy, filling in the details as we go. The highest level of the specification consists of a collection of assertions about what must be measured in order for a state machine C to properly manage the steam boiler hardware. There are 3 such assertions and so the top level specification is:

> **Specification.** A state machine C is a steam boiler control program if and only if equations 1 through 3 (on page 12) are satisfied by C when the transition alphabet is $A_{transition}$.

Constructing the specification then becomes a two part process. We must define instruments that will capture interesting behavior of C, and we must then use these instruments to constrain the permitted operation of the steam boiler control program.

3 System dynamics and operating state

3.1 System dynamics

This section formalizes the notes on system dynamics that are given in [Abr94a]. The program must determine the status of the physical devices using possibly partial information extracted from input message blocks. The *raw* values are those extracted from the input message blocks, the *calculated* values are those estimated for the next cycle, the *adjusted* values are the current cycle best estimate based on *raw* and *calculated* values.

Define DevNames to be the set of device names, one for each physical device that should be tracked.

Definition: 5 $DevNames = \{level, steam, monitor_n, pump_n : n \in PumpNames\}$

Define $v(a, d)$ to extract a raw numerical measure for each device from each input message block.

Definition: 6

$$v(a,d) = \begin{cases} a.\textsc{Level} & \text{if } d = level \\ a.\textsc{Steam} & \text{if } d = steam \\ P & \text{if } d = pump_n \text{ and } a.\textsc{PumpState}_{\mathrm{n}} = on \\ & \text{or } d = monitor_n \text{ and } a.\textsc{PumpControlState}_{\mathrm{n}} = on \\ 0 & \text{if } d = pump_n \text{ and } a.\textsc{PumpState}_{\mathrm{n}} = \mathit{off} \\ & \text{or } d = monitor_n \text{ and } a.\textsc{PumpControlState}_{\mathrm{n}} = \mathit{off} \\ \mathit{Fail} & \text{otherwise} \end{cases}$$

The instrument $\mathsf{I\_Raw}$ will retain either the value extracted by v, or *Fail* if that value is not within expected limits. $\mathsf{I\_Raw}$ depends on four other instruments: $\mathsf{I\_MinCalc}$ (the expected minimum value for each device in the next state), $\mathsf{I\_MaxCalc}$ (the expected maximum value for each device in the next state), $\mathsf{I\_PumpStat}$ which checks to see if pumps are responding to commands, and $\mathsf{I\_RepairStat}$ (the repair status for each device). On a xs input, $\mathsf{I\_Raw}$ should remain unchanged. A block message input a will change $\mathsf{I\_Raw}(d)$ to either $v(a, d)$ or *Fail*.

Definition: 7

$$
\begin{aligned}
&\textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) = \\
&\begin{cases}
\mathsf{I\_Raw}(d) & \text{if } a = xs \\
v(a,d) & \text{if } a \neq xs \\
& \text{and } d = pump_n \rightarrow \textsc{After}\langle a\rangle\cdot\mathsf{I\_PumpStat}(n) \neq \mathit{Fail} \\
& \text{and } d = monitor_n \rightarrow \textsc{After}\langle a\rangle\cdot\mathsf{I\_Monitor}(n) \neq \mathit{Fail} \\
& \text{and } \mathsf{I\_MinCalc}(d) \leq v(a,d) \leq \mathsf{I\_MaxCalc}(d) \\
& \text{and } \left(\begin{array}{l} \mathsf{I\_Raw}(d) \neq \mathit{Fail} \\ \text{or } \textsc{After}\langle a\rangle\cdot\mathsf{I\_RepairStat}(d) = \mathit{repaired} \end{array} \right. \\
\mathit{Fail} & \text{otherwise}
\end{cases}
\end{aligned}
$$

Now consider the adjusted values. Suppose we know $\mathsf{I\_MaxAdj}(d)$ in the current state for some $d \in \mathit{DeviceNames}$. Then what value should be assigned to $\textsc{After}\langle a\rangle\cdot\mathsf{I\_MaxAdj}(d)$? If $a = xs$ time has passed but no new message block has been received – so we ignore it and put $\textsc{After}\langle xs\rangle\cdot\mathsf{I\_MaxAdj}(d) = \mathsf{I\_MaxAdj}(d)$. If a is an input message block then there are two alternatives. Either a contains a believable message from the physical system, in which case $\textsc{After}\langle a\rangle\cdot\mathsf{I\_MaxAdj}(d) = \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d)$ or device d has failed and $\textsc{After}\langle a\rangle\cdot\mathsf{I\_MaxAdj}(d) = \mathsf{I\_MaxCalc}(d)$. Note that since $\mathsf{I\_MaxCalc}(d)$ is the calculated value for the *next* cycle so we use the current value, not $\textsc{After}\langle a\rangle\cdot\mathsf{I\_MaxCalc}(d)$.

Definition: 8

$$
\textsc{After}\langle a\rangle\cdot\mathsf{I\_MaxAdj}(d) = \begin{cases}
\mathsf{I\_MaxAdj}(d) & \text{if } a = xs \\
\textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) & \text{if } a \neq xs \\
& \text{and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) \neq \mathit{Fail} \\
\mathsf{I\_MaxCalc}(d) & \text{otherwise}
\end{cases}
$$

$\mathsf{I\_MinAdj}$ has identical structure.

Definition: 9

$$
\textsc{After}\langle a\rangle\cdot\mathsf{I\_MinAdj}(d) = \begin{cases}
\mathsf{I\_MinAdj}(d) & \text{if } a = xs \\
\textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) & \text{if } a \neq xs \\
& \text{and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) \neq \mathit{Fail} \\
\mathsf{I\_MinCalc}(d) & \text{otherwise}
\end{cases}
$$

For *steam* $\mathsf{I\_MinCalc}$ and $\mathsf{I\_MaxCalc}$ are straightforward. The informal specification tells us that U_1 and U_2 are, respectively, the maximum gradients of increase and decrease for the steam outcome.

Definition: 10

$$\begin{aligned}\mathsf{I\_MinCalc}(steam) &= \mathsf{I\_MinAdj}(steam) - U_2\Delta\ t\\ \mathsf{I\_MaxCalc}(steam) &= \mathsf{I\_MaxAdj}(steam) - U_1\Delta\ t\end{aligned}$$

What is $\Delta\ t$ in this formula? Since the quantities are given in seconds and the reported values cover 5 second periods, $\Delta\ t = 5$ seems reasonable.

The *level* is slightly more complicated but similar.

Definition: 11

$$\begin{aligned}\mathsf{I\_MinCalc}(level) = \mathsf{I\_MinAdj}(level) - \mathsf{I\_MaxAdj}(steam)\Delta\ t - 1/2U_1\Delta\ t^2 \\ + \sum_{n \in PumpNames} \mathsf{I\_MinAdj}(pump_n)\end{aligned}$$

$$\begin{aligned}\mathsf{I\_MaxCalc}(level) = \mathsf{I\_MaxAdj}(level) - \mathsf{I\_MinAdj}(steam)\Delta\ t - 1/2U_2\Delta\ t^2 \\ + \sum_{n \in PumpNames} \mathsf{I\_MaxAdj}(pump_n)\end{aligned}$$

After the program transmits a OpenPump_n command to pump n, the pump may require some time to overcome boiler pressure. Thus, $\mathsf{I\_PumpStat}(pump_n) = starting_1$ as soon as the program transmits OpenPump_n; if the next message block a has $a.\text{PumpState}_\text{n} = on$ then $\mathsf{I\_PumpStat}(pump_n)$ gets set to *on*, otherwise it gets set to $starting_2$; if the next message block a has $a.\text{PumpState}_\text{n} = on$ then $\mathsf{I\_PumpStat}(pump_n)$ gets set to *on*, otherwise it gets set to *Fail*. The pump is supposed to shut down without delay when a $\text{ClosePump}_\text{n}$ message is transmitted, so there is no need to monitor a similar process on stopping.

Definition: 12

$$\mathsf{I\_MinCalc}(pump_i) = \begin{cases} 0 & \text{if } \mathsf{I\_Raw}(pump_i) = \mathit{Fail} \\ & \text{or } \mathsf{I\_PumpStat}(i) \neq on \\ P & \text{otherwise} \end{cases}$$

$$\mathsf{I\_MaxCalc}(pump_i) = \begin{cases} P & \text{if } \mathsf{I\_Raw}(pump_i) = \mathit{Fail} \\ & \text{or } \mathsf{I\_PumpStat}(i) = on \\ 0 & \text{otherwise} \end{cases}$$

We postpone the definition of $\mathsf{I\_PumpStat}$ to page 16 definition 19.

Let's note here a curious element of the original specification. The pump controllers (the monitors) appear to have only one function — failing. According to the original specification, when the monitors disagree with the pumps themselves, we should assume the monitors have failed. This may be incorrect boiler engineering, but as humble computer scientists our task is merely to write the program, not to dispute such arcane matters. The instrument $\mathsf{I\_Monitor}$ is similar in function to $\mathsf{I\_PumpStat}$ and so we can define the calculated values for the monitors similarly as well.

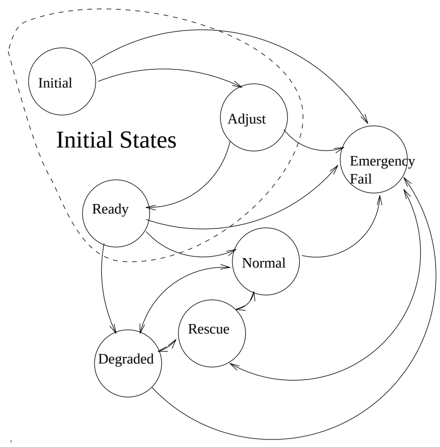


Figure 5: The operating states of the control program

Definition: 13

$$\mathsf{I\_MinCalc}(monitor_i) = \begin{cases} 0 & \text{if } \mathsf{I\_Raw}(monitor_i) = \mathit{Fail} \\ & \text{or } \mathsf{I\_Monitor}(i) \neq \mathit{on} \\ P & \text{otherwise} \end{cases}$$

$$\mathsf{I\_MaxCalc}(monitor_i) = \begin{cases} P & \text{if } \mathsf{I\_Raw}(monitor_i) = \mathit{Fail} \\ & \text{or } \mathsf{I\_Monitor}(i) = \mathit{on} \\ 0 & \text{otherwise} \end{cases}$$

3.2 I_Op

I_Op tracks the *operating state* of the program. The informal specification defines five *modes* of operation: *Initial*, *Emergency*, *Normal*, *Rescue*, and *Degraded*. We take each mode as a *operational state* and add two new operational states during the *Initial* mode. The program starts up waiting for the SteamBoilerWaiting message in state *Initial*. Once it gets a SteamBoilerWaiting message it moves into the *Adjust* state and evaluates the hardware. If the steam detection unit reports steam, the detection unit must be defective and the program enters the fail state. Otherwise, the program tries to adjust the level of water in the boiler – still looking for conditions that will cause it to fail. Once everything is ready, the program signals the physical units and waits for an ack in the *Ready* state before entering a working state – either normal or degraded. The state diagram corresponding to I_Op is given in figure 5 on page 10

The definition of I_Op below formalizes and provides details of the state diagram in figure 5. I_Op depends on a great many other instruments, one of which is worth defining and discussing immediately. The instrument I_TooManyFail should output $\mathbb{T}$ when too many of the physical devices have failed for continued operation. The informal specification seems recklessly optimistic here. The spec requires that the water level measurement unit and either the steam output detection unit or at least one pump control unit be failed before the control program can enter *Emergency* state. The specification here formalizes this requirement, but at this point one might either want to discuss the issue with the engineers or plan to be somewhere else during testing. A boiler that can be considered operational with no working pumps is too interesting.

Definition: 14

$$\mathsf{I\_TooManyFail}() = \begin{cases} \mathbb{T} & \text{if } \mathsf{I\_Raw}(\mathit{level}) = \mathit{Fail} \\ & \text{and } \left(\begin{array}{l} \mathsf{I\_Raw}(\mathit{steam}) = \mathit{Fail} \\ \text{or } (\exists n)\mathsf{I\_Raw}(\mathit{monitor}_n) = \mathit{Fail} \end{array} \right) \\ \mathbb{F} & \text{otherwise} \end{cases}$$

$\mathsf{I\_Op}$ is defined in two pieces: one to check for an emergency fail state and the other to calculate the next state if the system can keep operating.

Definition: 15

$$\begin{array}{l} \textsc{Start}\ \mathsf{I\_Op}() = \mathit{Initial} \\ \textsc{After}\langle a\rangle\cdot\mathsf{I\_Op}() \\ = \begin{cases} \mathit{Emergency} & \text{if } \mathsf{I\_Op}() = \mathit{Emergency} \\ & \text{or } \textsc{After}\langle a\rangle\cdot\mathsf{I\_TransmitError}() \\ & \text{or } \left(\begin{array}{l} \mathsf{I\_Op}() \in \{\mathit{Initial}, \mathit{Adjust}, \mathit{Ready}\} \\ \text{and } a.\textsc{Steam} \neq 0 \text{ or } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(\mathit{level}) = \mathit{Fail} \end{array} \right) \\ & \text{or } \left(\begin{array}{l} \mathsf{I\_Op}() \in \{\mathit{Normal}, \mathit{Degraded}, \mathit{Rescue}\} \\ \text{and } \left(\begin{array}{l} \textsc{After}\langle a\rangle\cdot\mathsf{I\_MaxCalc}(\mathit{level}) > M_2() \\ \text{or } \textsc{After}\langle a\rangle\cdot\mathsf{I\_MinCalc}(\mathit{level}) < M_1 \\ \text{or } \textsc{After}\langle a\rangle\cdot\mathsf{I\_TooManyFail}() \end{array} \right) \end{array} \right) \\ \textsc{After}\langle a\rangle\cdot\mathsf{I\_NoFail}(\mathsf{I\_Op}()) & \text{otherwise} \end{cases} \end{array}$$

So $\mathsf{I\_NoFail}$ outputs a map from operating states to operating states. $\mathsf{I\_NoFail}$ quite directly follows the description of the operating modes in the informal specification.

Definition: 16

$$
\textsc{After}\langle a\rangle\cdot\mathsf{I\_NoFail}(opst)
$$
$$
= \begin{cases}
\mathit{Adjust} & \text{if } opst = \mathit{Initial} \text{ and } a.\textsc{SteamBoilerWaiting} = \mathbb{T} \\
\mathit{Ready} & \text{if } opst = \mathit{Adjust} \text{ and } N_1 \leq \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(\mathit{level}) \leq N_2 \\
\mathit{Normal} & \text{if } \left(\begin{array}{l} opst = \mathit{Ready} \text{ and } a.\textsc{PhysicalUnitsReady} = \mathbb{T} \\ \text{and } (\forall d)\textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) \neq \mathit{Fail} \end{array} \right. \\
 & \text{or } opst \in \{\mathit{Degraded}, \mathit{Rescue}\} \text{ and } (\forall d)\textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) \neq \mathit{Fail} \\
\mathit{Degraded} & \text{if } \left(\begin{array}{l} opst = \mathit{Ready} \text{ and } a.\textsc{PhysicalUnitsReady} = \mathbb{T} \\ \text{and } (\exists d)\textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) = \mathit{Fail} \end{array} \right. \\
 & \text{or } \left(\begin{array}{l} opst \in \{\mathit{Normal}, \mathit{Rescue}\} \\ \text{and } (\exists d)\textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) = \mathit{Fail}\text{and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(\mathit{steam}) \neq \mathit{Fail} \\ \text{ and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(\mathit{level}) \neq \mathit{Fail} \end{array} \right. \\
\mathit{Rescue} & \text{if } opst \in \{\mathit{Normal}, \mathit{Degraded}\} \\
 & \text{and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(\mathit{level}) \neq \mathit{Fail} \\
 & \text{and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(\mathit{steam}) = \mathit{Fail} \\
opst & \text{otherwise}
\end{cases}
$$

4 The high level specification

4.1 The high level specification

The high level specification specifies the timing of output messages and requires that output messages be *permitted* as specified by the I_Permitted instrument. Note that input timing is specified through the I_TransmitError instrument which flags an unrecoverable error if the environment does not provide input message blocks at the required rate.

- Timing. *The compute state must never last more than* $5\lambda + \delta_1$ *time units except in the Initial state where the program is waiting for a* SteamBoilerWaiting *message from the physical devices.*

  (1) $$\mathsf{I\_SinceXmit}() \leq 5\lambda - \delta_1 \text{ or } \mathsf{I\_Op}() = \mathit{Initial}$$

- *The program always takes at least* $5\lambda - \delta_2$ *time units between transmitting messages.* Here we introduce the instrument I_Xout which has value equal to the message block being transmitted by C or value $\mathbb{F}$ if C is not transmitting a message block.

  (2) $$\textsc{After}\langle a\rangle\cdot\mathsf{I\_Xout}() \neq \mathbb{F} \rightarrow (5\lambda - \delta_2 \leq \mathsf{I\_SinceXmit}())$$

- *When the system program does transmit a message block, it does so according to the constraints of the* I_Permitted *instrument.* For any program message block b, I_Permitted(b) is true if and only if the program has permission to send a message b in its current state.

  (3) $$\mathsf{I\_Xout}() \neq \mathbb{F} \rightarrow \mathsf{I\_Permitted}(\mathsf{I\_Xout}())$$

4.2 Permitted outputs

Now we define the $\mathsf{I\_Permitted}$ instrument. Note that we do not define $\mathsf{I\_Permitted}$ completely, we only require that when $\mathsf{I\_Permitted}(b)$ is true b have certain properties. Equation 3 above ensures that $\mathsf{I\_Permitted}(b)$ is sometimes true.

- The program ready message can be sent in *Ready* operating state and only in that state.

$$(4) \qquad \mathsf{I\_Permitted}(b) \rightarrow (b.\textsc{ProgramReady} \leftrightarrow \mathsf{I\_Op}() = \mathit{Ready})$$

- If the operating state is one of the initial states then the output mode must be *Initial*, otherwise the mode must the same as the operating state.

$$(5) \qquad \mathsf{I\_Permitted}(b) \rightarrow \\ \mathsf{I\_Op}() \in \{\mathit{Initial}, \mathit{Adjust}, \mathit{Ready}\} \text{ and } b.\textsc{Mode} = \mathit{Initial} \\ \text{or } \mathsf{I\_Op}() \notin \{\mathit{Initial}, \mathit{Adjust}, \mathit{Ready}\} \text{ and } b.\textsc{Mode} = \mathsf{I\_Op}()$$

- The program can turn on the valve only in *Adjust* mode and must turn on the valve in *Adjust* mode if there is water in the boiler.

$$(6) \qquad \mathsf{I\_Permitted}(b) \rightarrow (b.\textsc{Valve} \leftrightarrow (\mathsf{I\_Raw}(\mathit{level}) > N_2 \text{ and } \mathsf{I\_Op}() = \mathit{Adjust})$$

- The program will adjust the water level by opening pumps only in accordance with the $\mathsf{I\_PumpsNeeded}$ instrument.

$$(7) \qquad \mathsf{I\_Permitted}(b) \rightarrow (\sum_{n \in PumpNames} [b.\textsc{OpenPump}_{\mathrm{n}}]) = \mathsf{I\_PumpsNeeded}()$$

- The program will adjust the water level by closing pumps similarly.

$$(8) \qquad \mathsf{I\_Permitted}(b) \rightarrow (\sum_{n \in PumpNames} [b.\textsc{ClosePump}_{\mathrm{n}}]) = \mathsf{I\_PumpsNotNeeded}()$$

- The program will send failure detection messages only in the *Normal*, *Degraded*, and *Rescue* states and must transmit these messages whenever a failure is detected and not yet acknowledged — it will repeatedly send detection messages until acknowledged. The program will send acknowledgments of repairs only in those same states and only when repairs have not been acknowledged. The $\mathsf{I\_RepairStat}$ instrument is used here, and $\mathsf{I\_RepairStat}(d)$ is the repair status of device d. The repair status is *justfailed* if the device has failed and no notice has yet been sent. The status is *unacked* if a notice has been sent, but no ack has been received. The status is *repaired* if the repair notice has been received but not yet acknowledged. Two functions are also used here: $\mathit{failnote}(d)$ is the name of the message that should be sent to notify the physical devices that a failure has been detected on device d, and $\mathit{repairack}(d)$ is the name of the message that should be sent to acknowledge a message from the physical devices signaling that a device has been repaired. These two functions are defined on page 18.

$$(9) \qquad \mathsf{I\_Permitted}(b) \rightarrow \left(\begin{array}{l} \left(\begin{array}{l} b.\mathit{failnote}(d) \leftrightarrow \\ \left(\begin{array}{l} \mathsf{I\_Op}() \in \{\mathit{Normal}, \mathit{Degraded}, \mathit{Rescue}\} \\ \text{and } \mathsf{I\_RepairStat}(d) \in \{\mathit{justfailed}, \mathit{unacked}\} \end{array} \right. \end{array} \right. \\ \text{and} \\ \left(\begin{array}{l} b.\mathit{repairack}(d) \leftrightarrow \\ \left(\begin{array}{l} \mathsf{I\_Op}() \in \{\mathit{Normal}, \mathit{Degraded}, \mathit{Rescue}\} \\ \text{and } \mathsf{I\_RepairStat}(d) = \mathit{repaired} \end{array} \right. \end{array} \right. \end{array} \right.$$

5 Pumps, monitors, and errors

In this section we define the remaining instruments. These instruments measure what pumps are needed, the pump and monitor status, the repair states of the physical devices and, finally, whether there is a transmission error from the physical devices.

5.1 Pumps needed and not

Finally, we turn to I_PumpsNeeded and I_PumpsNotNeeded. Following the invitation of the original specification, we will use a horribly simple algorithm. In accord with the informal definition, during the adjust operating state we will only turn on one pump if water is needed in the boiler. During the working operating states we will start all pumps whenever the calculated minimal level is less than N_1 and start $\lceil g/P \rceil - h$ pumps whenever the calculated level is greater than N_1 but less than or equal to N_2. Here g is the expected drop in level so $\lceil g/P \rceil$ is the number of pumps needed to counteract this drop and h is the number of pumps already started but not yet on. I_PumpsNotNeeded operates similarly. Note that there is a theoretical possibility here that $\mathsf{I\_PumpsNeeded}() > 0$ and $\mathsf{I\_PumpsNotNeeded}() > 0$. The boiler designer should be sure that $\mathsf{I\_MaxCalc}(\mathit{level}) > N_2$ implies $\mathsf{I\_MinCalc}(\mathit{level}) > N_1$ to avoid this possibility.

Definition: 17

$$\mathsf{I\_PumpsNeeded}() =$$
$$\begin{cases} 1 & \text{if } \mathsf{I\_Op}() = \mathit{Adjust} \\ & \text{ and } \mathsf{I\_MinAdj}(\mathit{level}) < N_1 \\ k & \text{if } \mathsf{I\_Op}() \in \{\mathit{Normal}, \mathit{Degraded}, \mathit{Rescue}\} \\ & \text{ and } \mathsf{I\_MinCalc}(\mathit{level}) < N_1 \\ & \text{ and } k = \sum_{n \in PumpNames}[\mathsf{I\_PumpStat}() = \mathit{off}] \\ min(\lceil g/P \rceil - h, k) & \text{if } \mathsf{I\_Op}() \in \{\mathit{Normal}, \mathit{Degraded}, \mathit{Rescue}\} \\ & \text{ and } N_1 \leq \mathsf{I\_MinCalc}(\mathit{level}) < N_2 \\ & \text{ and } g = (\mathsf{I\_MinAdj}(\mathit{level}) - \mathsf{I\_MinCalc}(\mathit{level})) > 0 \\ & \text{ and } h = \sum_{n \in PumpNames}[\mathsf{I\_PumpStat}(n) = \mathit{starting}_i, i < 3] \\ & \text{ and } k = \sum_{n \in PumpNames}[\mathsf{I\_PumpStat}(n) = \mathit{off}] \\ & \text{ and } h < \lceil g/P \rceil \\ 0 & \text{otherwise} \end{cases}$$

Definition: 18

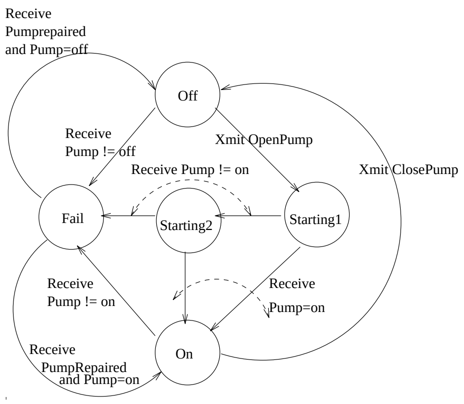
$$\mathsf{I\_PumpsNotNeeded}() =$$
$$\begin{cases} h & \text{if } \mathsf{I\_Op}() \in \{\mathit{Normal}, \mathit{Degraded}, \mathit{Rescue}\} \\ & \text{ and } \mathsf{I\_MaxCalc}(\mathit{level}) > N_2 \\ & \text{ and } h = \sum_{n \in PumpNames}[\mathsf{I\_PumpStat}(n) \in \{\mathit{on}, \mathit{starting}_i\}] \\ min(\lceil (-g)/P \rceil, h) & \text{if } \mathsf{I\_Op}() \in \{\mathit{Normal}, \mathit{Degraded}, \mathit{Rescue}\} \\ & \text{ and } N_1 \leq \mathsf{I\_MaxCalc}(\mathit{level}) > N_1 \\ & \text{ and } g = (\mathsf{I\_MaxAdj}(\mathit{level}) - \mathsf{I\_MaxCalc}(\mathit{level})) < 0 \\ & \text{ and } h = \sum_{n \in PumpNames}[\mathsf{I\_PumpStat}(n) \in \{\mathit{on}, \mathit{starting}_i\}] \\ 0 & \text{otherwise} \end{cases}$$

5.2 Pumps and pump control units

The I_PumpStat instrument monitors both commands from the program to the pumps and responses from the pumps. The pump control units, as described in the informal specification, are simple devices that check to see if water is flowing from the pumps into the boiler or not. To start with, let's assume that the status determined by I_PumpStat is always right, so that all that is required of the pump control units is that they synchronize status with the pump status.

The operation of I_PumpStat is sketched in figure 5.2.

Initially, assume the pumps are all closed. The informal specification tells us that any message block a that arrives must have $a.\textsc{PumpState}_{\mathrm{n}} \in \{on, off\}$ or else there is an unrecoverable failure — a transmission failure. Then we set $\mathsf{I\_PumpStat}(n) = a.\textsc{PumpState}_{\mathrm{n}}$ if $a.\textsc{PumpState}_{\mathrm{n}} = \mathsf{I\_PumpStat}(n)$ (the status is unchanged), or if $\mathsf{I\_PumpStat}(n) = starting_i$ and $a.\textsc{PumpState}_{\mathrm{n}} = on$ (the pump responds properly to a command sent in the last transmit



state), or if the pump has just recovered from an error. We set $\mathsf{I\_PumpStat}(n)$ to *starting* or *off* respectively if $a = xs$ and in the current state $\mathsf{I\_Xout}().\textsc{OpenPump}_\mathrm{n}$ or $\mathsf{I\_Xout}().\textsc{ClosePump}_\mathrm{n}$ (the pump stops instantaneously according to the informal specification). The pump is required to be operating within 5 seconds, so the *second* message received back from the pump control unit should confirm an *on* status. If there is no command being transmitted an xs transition leaves $\mathsf{I\_PumpStat}(n)$ unchanged. Any other input indicates a pump failure.

Definition: 19

$$
\begin{array}{l}
\textsc{Start}\ \mathsf{I\_PumpStat}(n) = \mathit{off} \\
\textsc{After}\langle a\rangle \cdot \mathsf{I\_PumpStat}(n) \\
\quad = \begin{cases}
a.\textsc{PumpState}_{\mathrm{n}} & \text{if } a \neq xs \text{ and } a.\textsc{PumpState}_{\mathrm{n}} \neq \mathbb{F} \\
 & \text{and } \left(\begin{array}{l} \mathsf{I\_PumpStat}(n) = a.\textsc{PumpState}_{\mathrm{n}} \\ \text{or } \left(\begin{array}{l} \mathsf{I\_PumpStat}(n) = \mathit{starting}_i \\ \text{and } a.\textsc{PumpState}_{=} \mathit{on} \end{array} \right) \\ \text{or } \left(\begin{array}{l} a.\textsc{PumpRepaired}_{(} n) = \mathbb{T} \\ \ \text{and } \mathsf{I\_PumpStat}(n) = \mathit{Fail} \end{array} \right) \end{array} \right) \\
\mathsf{I\_PumpStat}(n) & \text{if } a = xs \\
 & \text{and } \mathsf{I\_Xout}().\textsc{OpenPump}_{\mathrm{n}} = \mathbb{F} \\
 & \text{and } \mathsf{I\_Xout}().\textsc{ClosePump}_{\mathrm{n}} = \mathbb{F} \\
\mathit{starting}_1 & \text{if } a = xs \\
 & \text{and } \mathsf{I\_Xout}().\textsc{OpenPump}_{\mathrm{n}} = \mathbb{T} \\
 & \text{and } \mathsf{I\_Xout}().\textsc{ClosePump}_{\mathrm{n}} = \mathbb{F} \\
\mathit{starting}_2 & \text{if } a \neq xs \\
 & \text{and } \mathsf{I\_PumpStat}() = \mathit{starting}_1 \\
 & \text{and } a.\textsc{PumpState}_{\mathrm{n}} = \mathit{off} \\
\mathit{off} & \text{if } a = xs \\
 & \text{and } \mathsf{I\_Xout}().\textsc{ClosePump}_{\mathrm{n}} = \mathbb{T} \\
 & \text{and}\mathsf{I\_Xout}().\textsc{OpenPump}_{\mathrm{n}} = \mathbb{F} \\
\mathit{Fail} & \text{otherwise}
\end{cases}
\end{array}
$$

Now we can define $\mathsf{I\_Monitor}$.

Definition: 20

$$
\begin{array}{l}
\textsc{Start}\ \mathsf{I\_Monitor}(n) = \mathit{off} \\
\textsc{After}\langle a\rangle \cdot \mathsf{I\_Monitor}(n) \\
\quad = \begin{cases}
a.\textsc{PumpControlState}_{\mathrm{n}} & \text{if } a \neq xs \\
 & \text{and } a.\textsc{PumpControlState}_{\mathrm{n}} \neq \mathbb{F} \\
 & \text{and } \textsc{After}\langle a\rangle \cdot \mathsf{I\_PumpStat}(n) = a.\textsc{PumpControlState}_{\mathrm{n}} \\
 & \text{and } \left(\begin{array}{l} \mathsf{I\_Monitor}() \neq \mathit{Fail} \\ \text{or} \\ \left(\begin{array}{l} \mathsf{I\_Monitor}(n) = \mathit{Fail} \\ \text{and } a.\textsc{PumpControlRepaired}_{\mathrm{n}} = \mathbb{T} \end{array} \right) \end{array} \right) \\
\mathsf{I\_Monitor}(n) & \text{if } a = xs \\
\mathit{Fail} & \text{otherwise}
\end{cases}
\end{array}
$$

5.3 Repair status

What is required here is a collection of auxiliary functions to map from the device names to the appropriate message names. The auxiliary functions tell us which message is needed to notify of a failure or a repair or to acknowledge a failure or repair for any device d. The actual instrument $\mathsf{I\_RepairStat}$ is then trivial/

Definition: 21

$$failnote(d) \stackrel{\text{def}}{=} \begin{cases} \textsc{PumpFailureDetection}_\mathrm{n} & \text{if } d = pump_n \\ \textsc{PumpControlFailureDetection}_\mathrm{n} & \text{if } d = monitor_n \\ \textsc{LevelFailureDetection} & \text{if } d = level \\ \textsc{SteamFailureDetection} & \text{if } d = steam \end{cases}$$

$$failack(d) \stackrel{\text{def}}{=} \begin{cases} \textsc{PumpFailureAck}_\mathrm{n} & \text{if } d = pump_n \\ \textsc{PumpControlFailureAck}_\mathrm{n} & \text{if } d = monitor_n \\ \textsc{LevelFailureAck} & \text{if } d = level \\ \textsc{SteamOutcomeFailureAck} & \text{if } d = steam \end{cases}$$

$$repairnote(d) \stackrel{\text{def}}{=} \begin{cases} \textsc{PumpRepaired}_\mathrm{n} & \text{if } d = pump_n \\ \textsc{PumpControlRepaired}_\mathrm{n} & \text{if } d = monitor_n \\ \textsc{LevelRepaired} & \text{if } d = level \\ \textsc{SteamRepaired} & \text{if } d = steam \end{cases}$$

$$repairack(d) \stackrel{\text{def}}{=} \begin{cases} \textsc{PumpRepairedAck}_\mathrm{n} & \text{if } d = pump_n \\ \textsc{PumpControlRepairedAck}_\mathrm{n} & \text{if } d = monitor_n \\ \textsc{LevelRepairedAck} & \text{if } d = level \\ \textsc{SteamRepairedAck} & \text{if } d = steam \end{cases}$$

One thing to note about $\mathsf{I\_RepairStat}$ is that it does not have to check for errors. Inappropriate messages from C are impossible, due to specification equation 9 on page 14. And inappropriate messages from the physical devices cause $\mathsf{I\_TransmitError}()$ to become true putting the system in an unrecoverable state.

Definition: 22

$$\textsc{Start}\ \mathsf{I\_RepairStat}(d) = \mathit{Working}$$
$$\textsc{After}\langle a\rangle\cdot\mathsf{I\_RepairStat}(d)$$
$$= \begin{cases} \mathit{justfailed} & \text{if } \mathsf{I\_Raw}(d) \neq \mathit{Fail} \text{ and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) = \mathit{Fail} \\ \mathit{unacked} & \text{if } \mathsf{I\_RepairStat}(d) = \mathit{justfailed} \\ & \ \text{and } a = \mathit{xs} \text{ and } \mathsf{I\_Xout}() \neq \mathbb{F} \\ & \ \text{and } \mathsf{I\_Xout}().\mathit{failnote}(d) \\ \mathit{Fail} & \text{if } \mathsf{I\_RepairStat}(d) = \mathit{unacked} \\ \text{and } a \neq \mathit{xs} \text{ and } a.\mathit{failack}(d) & \\ \mathit{repaired} & \text{if } \mathsf{I\_RepairStat}(d) = \mathit{Fail} \text{ and } a \neq \mathit{xs} \\ & \text{and } \textsc{After}\langle a\rangle\cdot\mathsf{I\_Raw}(d) \neq \mathit{Fail} \\ & \text{and } a.\mathit{repairnote}(d) = \mathbb{T} \\ \mathit{Working} & \text{if } \mathsf{I\_RepairStat}(d) = \mathit{repaired} \text{ and } a = \mathit{xs} \\ & \text{and } \mathsf{I\_Xout}().\mathit{repairack}(d) \\ \mathsf{I\_RepairStat}(d) & \text{otherwise.} \end{cases}$$

5.4 Transmission errors

The informal specification tells us that a transmission error is encountered whenever a spurious message is found in a message block or a required message is not found. This error is not recoverable.

Definition: 23

$$\textsc{Start}\ \mathsf{I\_TransmitError}() = \mathbb{F}$$
$$\textsc{After}\langle a\rangle\cdot\mathsf{I\_TransmitError}()$$
$$= \begin{cases} \mathbb{T} & \text{if } a = xs \text{ and } \mathsf{I\_SinceRecv}() = 5\lambda \\ & \text{or } a \neq xs \\ & \text{and} \left(\begin{array}{l} \mathsf{I\_SinceRecv}() < 5\lambda - \delta_1 \\ \text{or } (\exists n) a.\textsc{PumpState}_{\mathrm{n}} = \mathbb{F} \\ \text{or } (\exists n) a.\textsc{PumpControlState}_{\mathrm{n}} = \mathbb{F} \\ \text{or } a.\textsc{Level} = \mathbb{F} \\ \text{or } a.\textsc{Steam} = \mathbb{F} \\ \text{or} \left(\begin{array}{l} a.\textsc{SteamBoilerWaiting} \\ \text{and } \mathsf{I\_Op}() \neq \textit{Initial} \end{array} \right) \\ \text{or} \left(\begin{array}{l} a.\textsc{PhysicalUnitsReady} \\ \text{and } \mathsf{I\_Op}() \neq \textit{Ready} \end{array} \right) \\ \text{or} \left(\begin{array}{l} a.\textit{failack}(d) \\ \text{and } \mathsf{I\_RepairStat}(d) \neq \textit{justfailed} \end{array} \right) \\ \text{or} \left(\begin{array}{l} a.\textit{repairnote}(d) \\ \text{and } \mathsf{I\_RepairStat}(d) \neq \textit{unacked} \end{array} \right) \end{array} \right) \\ \mathsf{I\_TransmitError}() & \text{otherwise} \end{cases}$$

References

[Abr94a] Jean-Raymond Abrial. Additional information for the steam-boiler control specification problem. In *Web Page. Dagstuhl Meeting: Methods for Semantics and Specification.* `http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html/additional-info.ps`, August 1994.

[Abr94b] Jean-Raymond Abrial. Steam-boiler control specification problem. In *Web Page. Dagstuhl Meeting: Methods for Semantics and Specification.* `http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html/steam-boiler-problem.ps`, August 1994.

[Arb68] Michael A. Arbib. *Algebraic theory of machines, languages, and semi-groups.* Academic Press, 1968.

[GKP89] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley, 1989.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Welsey, Reading MA, 1979.

[Moo64] E.F. Moore, editor. *Sequential Machines: Selected Papers.* Addison-Welsey, Reading MA, 1964.

[Yod91a] Victor Yodaiken. The algebraic feedback product of automata. In *Papers from the DIMACS Workshop on Computer Aided Verification*, AMS-DIMACS Series. American Mathematical Society, 1991.

[Yod91b] Victor Yodaiken. Modal functions for concise representation of finite automata. *Information Processing Letters*, Nov 20 1991.

[Yod95] Victor Yodaiken. Representations of automata. Technical Report TR-99, Department of Computer Science, New Mexico School of Mines, `http://www.cs.nmt.edu/~yodaiken`, 1995.

A State machines

The basic idea here is to use formal objects that *present* Moore machines conveniently. Let U be the universal alphabet and let U^* be the free monoid generated by U with $\circ$ as the monoid operation. Let ϵ be identity in U^*. Let ζ be the zero map on U^* with $\zeta w = \epsilon$ for all w, and let ι be the identity on U^*.

Define the class $\mathbb{P}$ of presentations of Moore machines on U to be the smallest set containing every pair that can be generated by finite application of the following three rules.

- $(f, \iota) \in \mathbb{P}$ for every $f : U^* \to X$.
- If $(f, \zeta) \in \mathbb{P}$ for every $f : U^* \to X$.
- If $(f, \theta) \in \mathbb{P}$ and $z \in U^*$, then $(f, \theta^z) \in \mathbb{P}$ where $\theta^z w = \theta(w \circ z)$ for all w.

Let σ be a distinguished variable over U^*. For any $\mathsf{M} = (f, \theta) \in \mathbb{P}$ define $\mathsf{M}() = f(\theta\sigma)$, define $\textsc{Start}\ \mathsf{M} = (f, \zeta)$ and define $\textsc{After}\, z{\cdot}\mathsf{M} = (f, \theta^z)$. It is trivial to show that $\mathbb{P}$ is closed under the $\textsc{Start}$ and $\textsc{After}\cdot\cdot$ operators. If is also not hard to show that $\mathbb{P}$ is closed under p.r. definitions of the form

$$\begin{aligned}&\textsc{Start}\ \mathsf{M}() = k_0\\ &\textsc{After}\, a{\cdot}\mathsf{M}() = h(a, \mathsf{M}(), \mathsf{M}_1(), \dots \mathsf{M}_n())\end{aligned}$$

The mapping from $\mathbb{P}$ to standard tuple presentations of Moore machines is via the Nyquist congruence[Arb68]. Thus, each $\mathsf{M} \in \mathbb{P}$ corresponds to a constructed Moore machine tuple. Of course, we have not restricted f in pairs (f, θ) so M does not necessarily correspond to a finite or even to a computable Moore machine tuple.

In [Yod95] I show how these formal objects can be composed to form trees with simple pairs at the leaves. The trees are shown to correspond to feedback product factorizations of Moore machines, and the feedback product offers a direct model of interconnection of systems and parallelism.