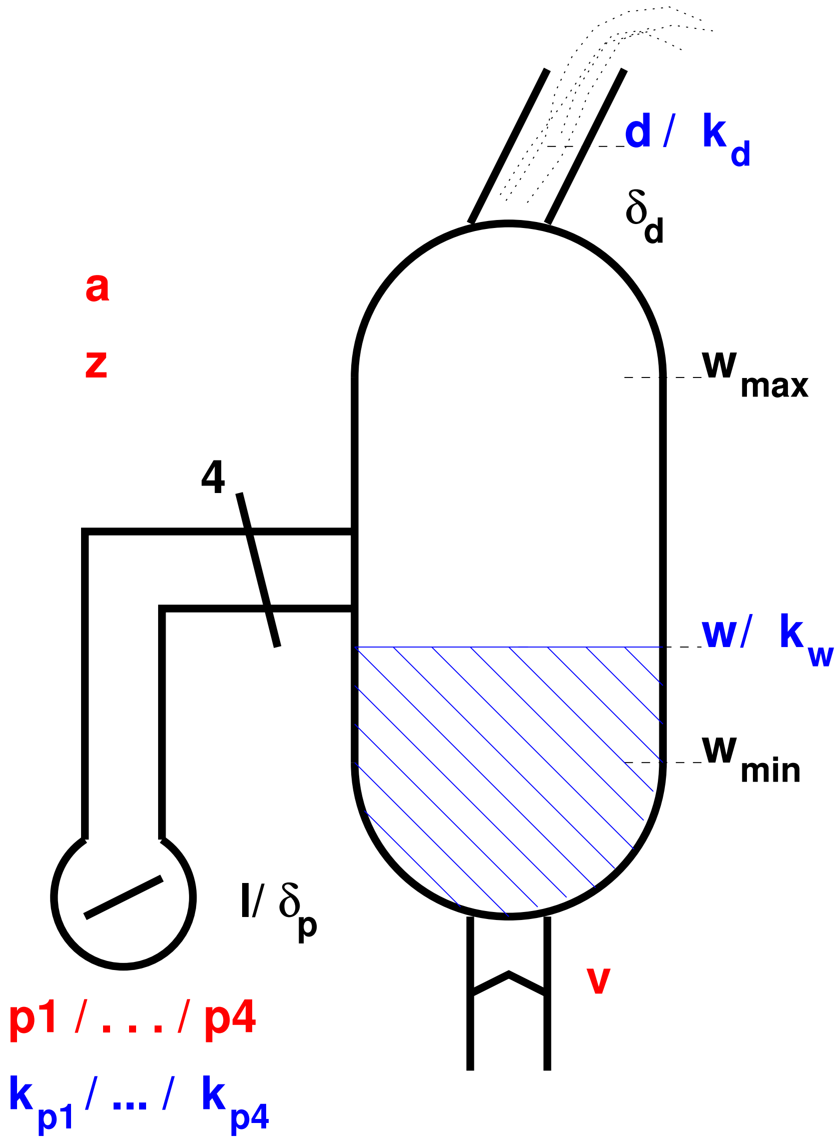
Formal Specification of Software

Steam Boiler Control
An Example in ASM Formalisation

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

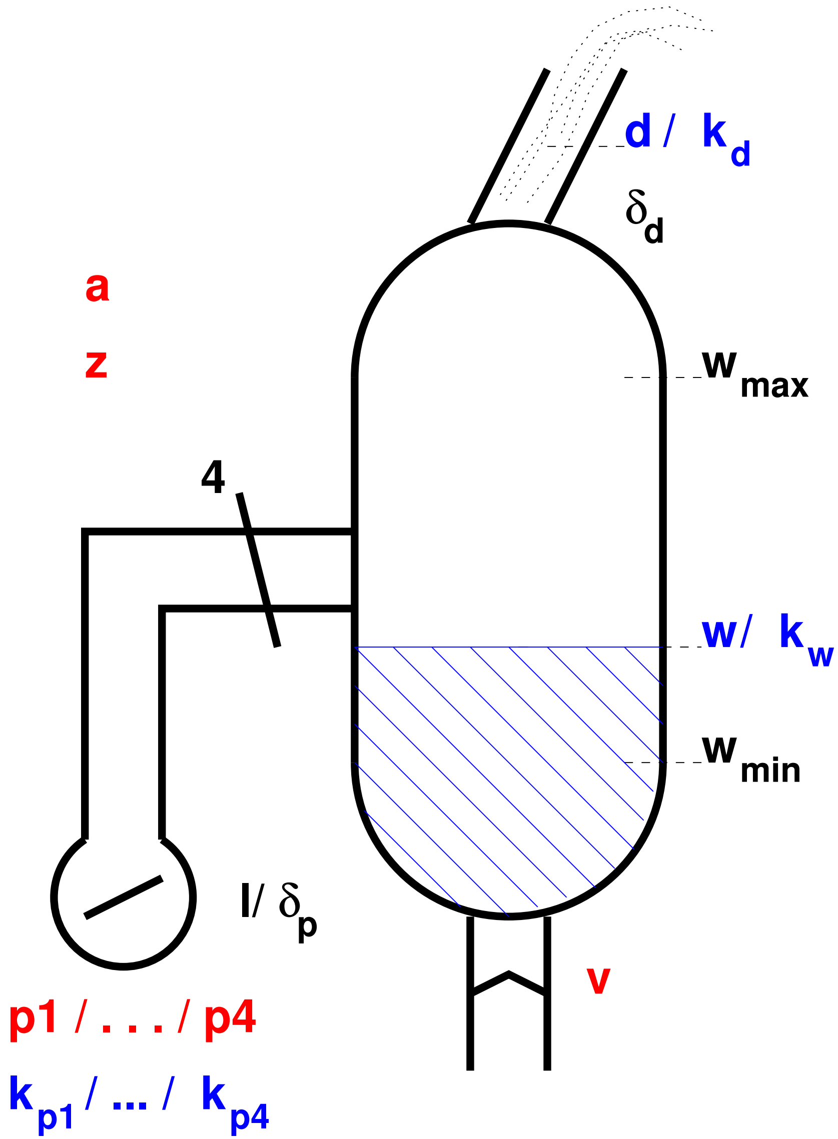
Steam Boiler Control: Scenario



System Components

- steam boiler
- water level measuring device
- four pumps
- four pump controlers
- steam quantity measuring device
- valve for emptying the boiler

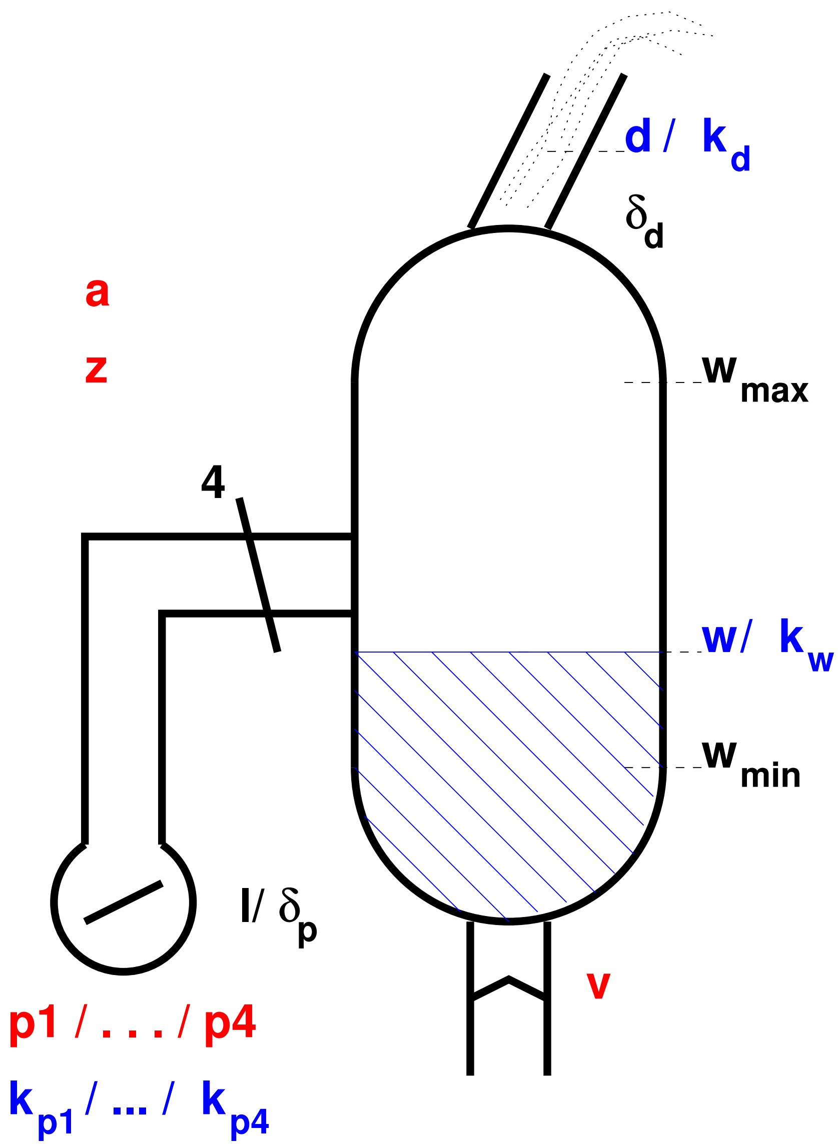
Steam Boiler Control: Scenario



Physical constants

w_{min}	**minimal water level**
w_{max}	**maximal water level**
l	**water amount per pump**
d_{max}	**maximal quantity of steam exiting the boiler**
δ_p	**error in the value of l**
δ_d	**error in steam measurement**

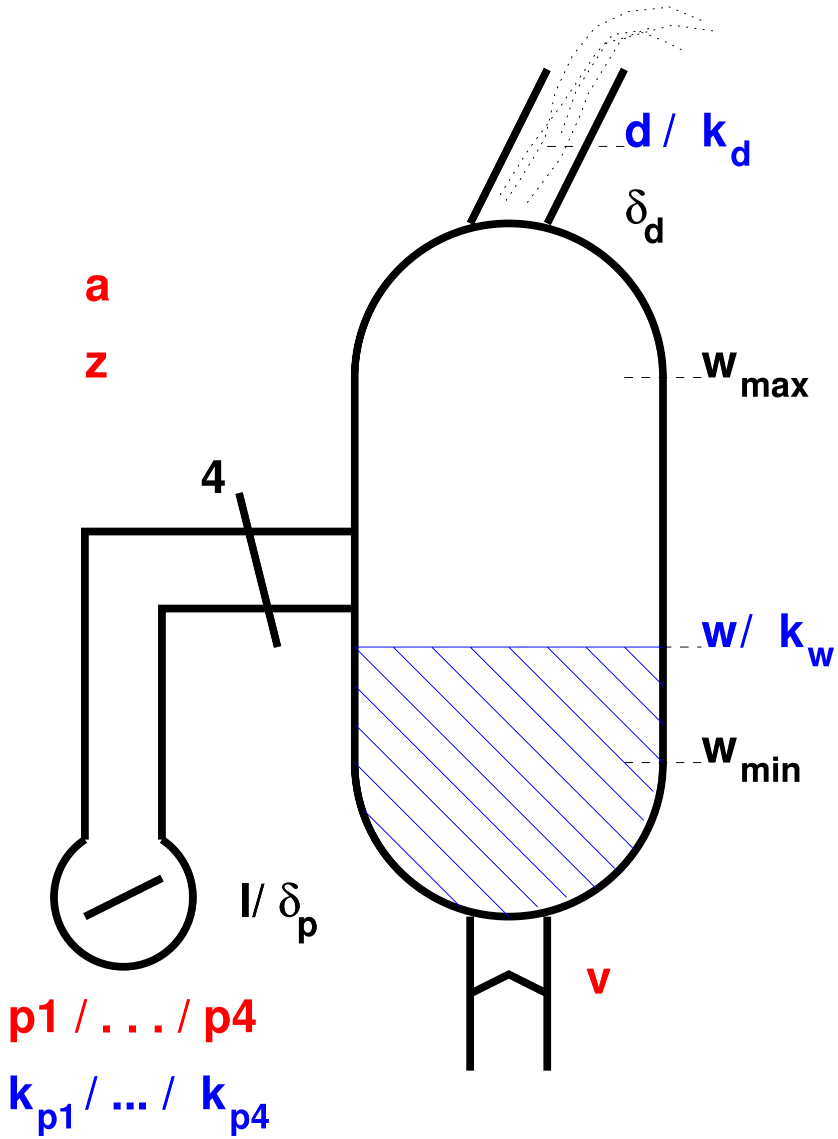
Steam Boiler Control: Scenario



Measured values

w	water level
d	amount of steam exiting the boiler
$k_p(i)$	pump i works/broken
k_w	water level measuring device works/broken
k_d	steam amount measuring device works/broken

Steam Boiler Control: Scenario



Control values

$p(i)$	**pump i on/off**
v	**valve open/closed**
a	**boiler on/off**
z	**state**
	init/norm/broken/stop

Steam Boiler with ASMs

Restrictions

- Real-time aspects not modelled
- Communication between devices not modelled

Steam Boiler with ASMs

Restrictions

- **Real-time aspects not modelled**
- **Communication between devices not modelled**

Measured values

Modelled as functions that are changed externally

Control values

Modelled as functions that are read externally

Steam Boiler with ASMs: Two Versions

First version

The possibility that devices are broken is not modelled

States: $init, normal, stop$

Second version

The possibility that devices are broken is included in the model

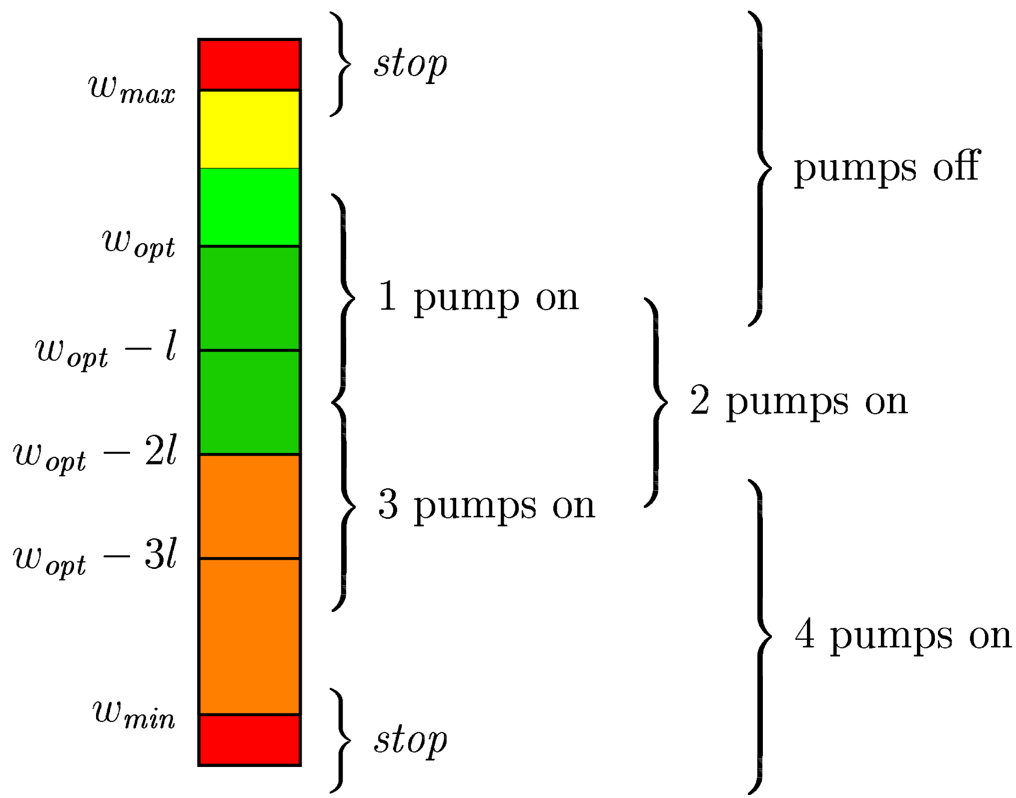
Additional state: $broken$

First Version: Strategy for Filling

Additional constant w_{opt}

Optimal water level

Strategy



First Version: Vocabulary

Universes

$$\begin{aligned}
state &= \{init, norm, stop\} \\
openClosed &= \{open, closed\} \\
water &= \mathbb{N} \\
pumps &= \{1, 2, 3, 4\} \\
onOff &= \{on, off\}
\end{aligned}$$

First Version: Vocabulary

Universes

$$
\begin{array}{lcl}
state & = & \{init, norm, stop\} \\
openClosed & = & \{open, closed\} \\
water & = & \mathbb{N} \\
pumps & = & \{1, 2, 3, 4\} \\
onOff & = & \{on, off\}
\end{array}
$$

Note

These are unary boolean functions; they define a type/class

First Version: Vocabulary

Dynamic functions

$p:$	pumps $\rightarrow$	onOff	controling the pumps
$v:$	$\rightarrow$	openClosed	controling the steam valve
$a:$	$\rightarrow$	onOff	controling the boiler
$z:$	$\rightarrow$	state	boiler state

External functions

$w:$	$\rightarrow$	water	water level
$d:$	$\rightarrow$	water	steam exiting boiler

Static functions

$+, -, *:$	$\mathbb{N} \times \mathbb{N} \rightarrow$	$\mathbb{N}$	arithmetic
$<, \leq:$	$\mathbb{N} \times \mathbb{N} \rightarrow$	Boole	ordering
$w_{max}, w_{min}, w_{opt}, l, d_{max}$	$\rightarrow$	$\mathbb{N}$	physical constants

Initial State

$$a = \mathit{off}$$

$$z = \mathit{init}$$

Rule *Initialisation*

```
if ¬(z = init) then
  skip
else
  if 0 < d then
    z := stop
  else if w < w_min + d_max then
    par
      v    :=  closed
      p(i) :=  on    (i = 1..4)
    endpar
  else if w_max < w then
    par
      v    :=  open
      p(i) :=  off   (i = 1..4)
    endpar
  else
    par
      z    :=  norm
      v    :=  closed
      a    :=  on
      p(i) :=  off   (i = 1..4)
    endpar
  endif endif endif
endif
```

Rule *Normal*

```
if ¬(z = norm) then
   skip
else
   if w_max < w ∨ w < w_min then
      par
         a := off
         z := stop
      endpar
   else
      par
         if w ≤ w_opt           then p(1) := on else p(1) := off endif
         if w ≤ w_opt − l       then p(2) := on else p(2) := off endif
         if w ≤ w_opt − (2 ∗ l) then p(3) := on else p(3) := off endif
         if w ≤ w_opt − (3 ∗ l) then p(4) := on else p(4) := off endif
      endpar
   endif
endif
```

Rule *Control*

par
Initialisation
Normal
endpar

Second Version: Vocabulary

Universes

$$
\begin{aligned}
\mathit{state} &= \{\mathit{init}, \mathit{norm}, \mathit{broken}, \mathit{stop}\} \\
\mathit{openClosed} &= \{\mathit{open}, \mathit{closed}\} \\
\mathit{water} &= \mathbb{N} \\
\mathit{pumps} &= \{1, 2, 3, 4\} \\
\mathit{onOff} &= \{\mathit{on}, \mathit{off}\} \\
\mathit{worksBroken} &= \{\mathit{works}, \mathit{broken}\}
\end{aligned}
$$

Second Version: Vocabulary

Dynamic functions

p :	pumps $\longrightarrow$	onOff	controling the pumps
v :	$\longrightarrow$	openClosed	controling steam valve
a :	$\longrightarrow$	onOff	controling the boiler
z :	$\longrightarrow$	state	boiler state
s_{min}, s_{max} :	$\longrightarrow$	water	estimated water level
n_p :	$\longrightarrow$	pumps	number of active pumps

External functions

w :	$\longrightarrow$	water	water level
d :	$\longrightarrow$	water	steam exiting boiler
k_p :	pumps $\longrightarrow$	worksBroken	pump works/broken
k_w :	$\longrightarrow$	worksBroken	water level device
k_d :	$\longrightarrow$	worksBroken	steam amount device

Second Version: Vocabulary

Static functions

$+, -, *, \min :$	$\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$	**arithmetic**
$<, \leq :$	$\mathbb{N} \times \mathbb{N} \rightarrow$ **Boole**	**ordering**
$w_{max}, w_{min}, l :$	$\rightarrow \mathbb{N}$	**physical constants**
$d_{max}, \delta_p, \delta_d :$	$\rightarrow \mathbb{N}$	**physical constants**
$optPumps :$	**water** $\times$ **water** $\rightarrow$ **pumps**	**optimal pump number**
$numWorking :$	$\mathbb{N} \times$ **worksBroken**$^4 \rightarrow \mathbb{N}$	**number of working pumps**
$controlPumps :$	**pumps**$^2 \times$ **worksBroken**$^4 \rightarrow$ **onOff**	**control for each pump**

Second Version: Vocabulary

Static function $optPumps$ (encodes the strategy)

$optPumps(w_1, w_2)$ = **optimal number of pumps for water level between w_1 and w_2**

Second Version: Vocabulary

Static function $optPumps$ **(encodes the strategy)**

$optPumps(w_1, w_2)$ = **optimal number of pumps for water level between** w_1 **and** w_2

Static function $numWorking$

$$numWorking(i, k_1, k_2, k_3, k_4) \;=\; \#\{j \mid j \leq i \wedge k_j = works\}$$

Second Version: Vocabulary

Static function $optPumps$ **(encodes the strategy)**

$optPumps(w_1, w_2)$ = **optimal number of pumps for water level between w_1 and w_2**

Static function $numWorking$

$$numWorking(i, k_1, k_2, k_3, k_4) = \#\{j \mid j \le i \wedge k_j = works\}$$

Static function $controlPumps$

$$controlPumps(i, n_{opt}, k_1, k_2, k_3, k_4) = \begin{cases} on & \textbf{if } numWorking(i-1, k_1, k_2, k_3, k_4) < n_{opt} \\ off & \textbf{otherwise} \end{cases}$$

Rule *Initialisation*

if $\neg(z = init)$ **then**
  skip
else
  if $0 < d \vee k_w = broken$
    $\vee k_d = broken$ **then**
   $z := stop$
  else if $w < w_{min} + d_{max}$ **then**
   par
    $v := closed$
    $p(\mathbf{i}) := on \quad (\mathbf{i} = 1..4)$
   endpar
  else if $w_{max} < w$ **then**
   par
    $v := open$
    $p(\mathbf{i}) := off \quad (\mathbf{i} = 1..4)$
   endpar
  else
   par
    $z := norm$
    $v := closed$
    $s_{min} := w$
    $s_{max} := w$
    $n_p := 0$
    $p(\mathbf{i}) := off \quad (\mathbf{i} = 1..4)$
   endpar
  endif endif endif
endif

Rule *NormBroken*

if $\neg(z = norm \vee z = broken)$ **then**
  skip
else
  if $k_w = works$ **then**
    let $min = w,\ max = w,\ z_{val} = norm$ **in** ***ControlPumps*** **endlet**
  else if $k_d = works$ **then**
    let $min = s_{min} - d + n_p \cdot l - \delta_d - n_p \cdot \delta_p$,
      $max = s_{max} - d + n_p \cdot l + \delta_d + n_p \cdot \delta_p,$
      $z_{val} = broken$
    in ***ControlPumps*** **endlet**
  else
    par
      $z \ := \ stop$
      $a \ := \ off$
    endpar
  endif endif
endif

Rule *ControlPumps*

if $min < w_{min} \vee w_{max} < max$ **then**
  par
    $z := stop$
    $a := \mathit{off}$
  endpar
else
  let $n_{opt} = optPumps(min, max)$ **in**
    par
      $p(\mathbf{i}) := controlPumps(i, n_{opt}, k_p(1), \ldots, k_p(4))$ $(\mathbf{i} = 1..4)$
      $n_p := \min(n_{opt}, numWorking(4, k_p(1), \ldots, k_p(4)))$
      $s_{min} := min$
      $s_{max} := max$
      $z := z_{val}$
    endpar
  endlet
endif

Rule *Control*

```
par
   Initialisation
   NormBroken
endpar
```

Alternative Solution: Vocabulary

Universes

$$
\begin{aligned}
state &= \{init, norm, broken, stop\} \\
openClosed &= \{open, closed\} \\
water &= \mathbb{N} \\
pumps &= \{1, 2, 3, 4\} \\
onOff &= \{on, off\} \\
worksBroken &= \{works, broken\} \\
waitCompute &= \{wait, compute\}
\end{aligned}
$$

Alternative Solution: Vocabulary

Additional dynamic functions

$i : \; \rightarrow$	**pumps**	**current pump**
$f : \; \rightarrow$	**waitCompute**	**next cycle**

Meaning of function f

$f = compute$**:** **Control the pumps**

$f = wait$**:** **Measurement**

Alternative: Rule *Initialisation*

if $\neg(z = init)$ **then**
  skip
else
  if $0 < d \vee k_w = broken$
      $\vee k_d = broken$ **then**
    $z := stop$
  else if $w < w_{min} + d_{max}$ **then**
    par
      $v := closed$
      $p(\mathbf{i}) := on \quad (\mathbf{i} = 1..4)$
      $f := wait$
    endpar
  else if $w_{max} < w$ **then**
    par
      $v := open$
      $p(\mathbf{i}) := off \quad (\mathbf{i} = 1..4)$
      $f := wait$
    endpar
  else
    par
      $z := norm$
      $f := wait$
      $v := closed$
      $s_{min} := w$
      $s_{max} := w$
      $n_p := 0$
      $p(\mathbf{i}) := off \quad (\mathbf{i} = 1..4)$
    endpar
endif endif endif endif

Alternative: Rule *NormBroken* (1)

if $\neg((z = norm \vee z = broken) \wedge f = wait)$ **then**
  skip
else
  if $k_w = works$ **then**
    par

$$
\begin{array}{lll}
s_{min} & := & w \\
s_{max} & := & w \\
z & := & norm \\
f & := & compute \\
i & := & 1 \\
n_p & := & 0
\end{array}
$$

    endpar

Alternative: Rule *NormBroken* (2)

else if $k_d = works$ **then**

par

$$
\begin{array}{lcl}
s_{min} & := & s_{min} - d + n_p \cdot l - \delta_d - n_p \cdot \delta_p \\
s_{max} & := & s_{max} - d + n_p \cdot l + \delta_d + n_p \cdot \delta_p \\
z & := & broken \\
f & := & compute \\
i & := & 1 \\
n_p & := & 0
\end{array}
$$

endpar

else

par

$$
\begin{array}{lcl}
z & := & stop \\
a & := & off
\end{array}
$$

endpar

endif endif

endif

Alternative: Rule *ControlPumps* (1)

if $\neg((z = norm \lor z = broken) \land f = compute)$ **then**
   skip
else
   if $s_{min} < w_{min} \lor w_{max} < s_{max}$ **then**
      par
$$z := stop$$
$$a := off$$
      endpar

Alternative: Rule *ControlPumps* (2)

```
else
  par
    if n_p < optPumps(s_min, s_max) ∧ k_p(i) = works then
      par
        p(i)  :=  on
        n_p   :=  n_p + 1
      endpar
    else
      p(i) := off
    endif
    if i < 4 then
      i := i + 1
    else
      f := wait
    endif
  endpar
endif
```

Alternative: Rule *Control*

```
par
    Initialisation
    NormBroken
    ControlPumps
endpar
```