Steam Boiler:
Extended Focus Specification and its Verification in Isabelle/HOL.

Maria Spichkova

January 19, 2007

Abstract

The main idea of this case study was taken from [1]. This paper represents an extension of the Focus specification [1] of the steam boiler, its translation in Isabelle/HOL [2] and the corresponding formal Isabelle/HOL proofs for the translated specifications, which show that the specified steam boiler architecture fulfills the specified steam boiler requirements.

Contents

1 Introduction

This paper represents the verification of the steam boiler specification. The original version of the formalization of the steam boiler as the Focus specifications is represented in [1]. These Focus specifications were first of all extended according the methodology "Focus on Isabelle/HOL", after that the (extended) specifications of all components of the system were translated schematically to Isabelle/HOL and the refinement relation between the requirement and the architecture specification of the system was proven. Thus, the extended Focus specifications of steam boiler are equal modulo syntax to presented here Isabelle/HOL predicates that represent they semantics.

In addition, the correctness of the input/output relations was proven for all components of the system. This case study shows also how we can deal with local variables (states) and in which way we can represent mutual recursive functions to avoid problems in proofs.

1.1 Focus

In Focus any specification characterizes the relation between the *communication histories* for the external *input* and *output channels*. The formal meaning of a specification is exactly this external *input/output relation*. The Focus specifications can be structured into a number of formulas each characterizing a different kind of property, the most prominent classes of them are *safety* and *liveness properties*.

The central concept in Focus are *streams*, that represent communication histories of *directed channels*.

In Focus streams are represented as functions mapping natural numbers to messages, where a message for the case of timed stream can be either a data message or *time tick*.

In the Focus specification of steam boiler are used the following Focus operators together with standard logical operators:

- ✓ An empty stream is represented by $\langle\rangle$.
- ✓ $\langle x\rangle$ denotes the stream (list) consisting of exactly one element, x.
- ✓ The nth message of a stream s can be denoted in Focus either by $s(n)$ or by $s.n$. We prefer the second kind of notation.
- ✓ $\#s$ yields the length of the stream to which is applied.
- ✓ $s_1 \frown s_2$ – concatenation of the streams s_1 and s_2 produces a stream that starts with the messages of the stream s_1 followed by the messages of the stream s_2.
- ✓ The append function $m \,\&\, s$ results concatenation of the message m to the head of stream s: $m \,\&\, s \overset{\mathsf{def}}{=} \langle m\rangle \frown s$
- ✓ $\mathsf{ft}.s$ and $\mathsf{rt}.s$ yield the first element of the stream and the stream without the first element respectively.
- ✓ $\overline{s}$ denotes in Focus the untimed stream obtained by removing all ticks in s.
- ✓ $s{\downarrow}_t$ truncates a timed *infinite* stream s at time t.
- ✓ $\mathsf{ts}(s)$ holds for a timed stream s iff s is time-synchronous in the sense that exactly one message is transmitted in each time unit.
- ✓ $\mathsf{ti}(s,n)$ denotes the sequence of messages that are present on the channel s at the time interval between nth and $(n+1)$th ticks.

For detailed description of Focus see [1].

We also introduce a new variant of the Focus tables – tiTable – to have more clear representation of component assurances when one argues about a component in time interval manner. The main difference of tiTable from the classical Focus table is that argumentation about streams in a classical Focus table treads *message-by-message*, where in a tiTable the argumentation about streams treads on time intervals – the possible combinations of message sequences on component channels at the time interval t between ticks $t-1$ and t are treated.

1.2 Isabelle

Isabelle [2] is a specification and verification system implemented in the functional programming language ML. Isabelle/HOL is the specialization of Isabelle for Higher Order Logic. To specify a system with Isabelle means creating *theories*. A theory is a named collection of types, functions (constants), and theorems (lemmas). Similar to the module concept from Focus, we can understand a theory in Isabelle as a module.

The base types in Isabelle/HOL are `bool`, the type of truth values and `nat`, the type of natural numbers. The base type constructors are `list`, the type of lists, and `set`, the type of sets. Function types are denoted by $\Rightarrow$. The operator $\Rightarrow$ is right-associative. The type variables are denoted by `'a`, `'b` etc.

Terms in Isabelle/HOL are formed as in functional programming by applying functions to arguments. Terms may also contain λ-abstractions.

For detailed description of Isabelle/HOL see [2] and [3].

2 Translation schema

In this section we present the translation schema only for the Focus operators and expressions, that are used in the case study.

Focus operator, f	Isabelle/HOL representation, $[\![f]\!]_{Isab}$
$\langle\rangle$	`[]` (an empty list)
$\#s$	`length` $[\![s]\!]_{Isab}$
$s.(n+1)$	`nth` $[\![s]\!]_{Isab}$ $[\![n]\!]_{Isab}$, $[\![s]\!]_{Isab}$ `!` $[\![n]\!]_{Isab}$
$x \frown y$	$[\![x]\!]_{Isab}$ @ $[\![y]\!]_{Isab}$
$x \in s$ ($s \in M^*$ for some M)	$[\![x]\!]_{Isab}$ `mem` $[\![s]\!]_{Isab}$
ft.s	`hd` $[\![s]\!]_{Isab}$

Table 1: Isabelle/HOL representation of the FOCUS operators on finite untimed streams

FOCUS operator, f	Isabelle/HOL representation, $[\![f]\!]_{Isab}$
$\#s$	∞
$s.(n+1)$	$[\![s]\!]_{Isab}\ [\![n]\!]_{Isab}$
$x \frown y$	$[\![x]\!]_{Isab}$, x is infinite
$x \frown y$	`fin_inf_append` $[\![x]\!]_{Isab}\ [\![y]\!]_{Isab}$ $(x \in M^{*},\ y \in M^{\infty})$

Table 2: Isabelle/HOL representation of the FOCUS operators on infinite untimed streams

FOCUS operator, f	Isabelle/HOL representation, $[\![f]\!]_{Isab}$
$\langle\rangle$	[] (an empty list)
$\#s$	`fin_length` $[\![s]\!]_{Isab}$
$s.n$	`fin_nth` $[\![s]\!]_{Isab}\ [\![n]\!]_{Isab}$
$x \frown y$	$[\![x]\!]_{Isab}$ @ $[\![y]\!]_{Isab}$
$\overline{s}$	`fin_make_untimed` $[\![s]\!]_{Isab}$
$\mathsf{ti}(s,t)$	`ti` $[\![s]\!]_{Isab}\ [\![t]\!]_{Isab}$

Table 3: Isabelle/HOL representation of the FOCUS operators on finite timed streams

FOCUS operator, f	Isabelle/HOL representation, $[\![f]\!]_{Isab}$
$s.n$	`inf_nth` $[\![s]\!]_{Isab}\ [\![n]\!]_{Isab}$
$x \frown y$	$[\![x]\!]_{Isab}$, x is infinite
$x \frown y$	`fin_inf_append` $[\![x]\!]_{Isab}\ [\![y]\!]_{Isab}$ $(x \in M^{\underline{*}},\ y \in M^{\underline{\infty}})$
$s \downarrow_t\ (*)$	`inf_truncate` $[\![s]\!]_{Isab}\ [\![t]\!]_{Isab}$ $(t \in \mathbb{N})$
$\mathsf{ts}(s)$	`ts` $[\![s]\!]_{Isab}$
$\overline{s}$	`make_untimed (InfT` $[\![s]\!]_{Isab}$`)`
$\mathsf{ti}(s,t)$	$[\![s]\!]_{Isab}\ [\![t]\!]_{Isab}$

Table 4: Isabelle/HOL representation of the FOCUS operators on infinite timed streams

FOCUS expression E	Isabelle/HOL representation, $[\![E]\!]_{Isab}$
true	*True*
false	*False*
$A = B$	$[\![A]\!]_{Isab} = [\![B]\!]_{Isab}$
$A \neq B$	$[\![A]\!]_{Isab} \neq [\![B]\!]_{Isab}$
$A < B,\ B > A$	$[\![A]\!]_{Isab} < [\![B]\!]_{Isab}$
$A \leq B,\ B \geq A$	$[\![A]\!]_{Isab} \leq [\![B]\!]_{Isab}$
$A < B < C,\ C > B > A$	$[\![A]\!]_{Isab} < [\![B]\!]_{Isab} \wedge [\![B]\!]_{Isab} < [\![C]\!]_{Isab}$
$A \leq B \leq C,\ C \geq B \geq A$	$[\![A]\!]_{Isab} \leq [\![B]\!]_{Isab} \wedge [\![B]\!]_{Isab} \leq [\![C]\!]_{Isab}$
$\neg Q$	$\neg [\![Q]\!]_{Isab}$
$Q_1 \vee Q_2$	$[\![Q_1]\!]_{Isab} \vee [\![Q_1]\!]_{Isab}$
$Q_1 \wedge Q_2$	$[\![Q_1]\!]_{Isab} \wedge [\![Q_2]\!]_{Isab}$
$Q_1 \Rightarrow Q_2$	$[\![Q_1]\!]_{Isab} \longrightarrow [\![Q_2]\!]_{Isab}$
$Q_1 \Leftrightarrow Q_2$	$[\![Q_2]\!]_{Isab} = [\![Q_2]\!]_{Isab}$
x	*x* (x is a variable)
$\exists\, x : Q$	$\exists\, x.\ [\![Q]\!]_{Isab}$
$\forall\, x : Q$	$\forall\, x.\ [\![Q]\!]_{Isab}$
if F then F_1 else F_2 fi	(*if* $[\![F]\!]_{Isab}$ *then* $[\![F_1]\!]_{Isab}$ *else* $[\![F_2]\!]_{Isab}$)
let A in B	*let* $[\![A]\!]_{Isab}$ *in* ($[\![B]\!]_{Isab}$)
$F(x_1, \dots, x_n)$	(*F* $[\![x_1]\!]_{Isab} \dots [\![x_n]\!]_{Isab}$)

Table 5: Isabelle/HOL representation of the FOCUS expressions

The where statement is in Focus of the form

F where $v_1, \dots, v_n$ so that G

where $v_1, \dots, v_n$ are logical variables and F, G are formulas (the formula G defines the values of the variables $v_1, \dots, v_n$). The semantics of this statement is defined in Focus by the formula

$\exists\, v_1, \dots, v_n :\ F \wedge G$

The original Focus definition allows to have as v_i $(1 \leq i \leq n)$ also a function name – the function is then specified by the formula G. To have more clear syntax, we recommend to restrict the where statement the cases where $v_1, \dots, v_n$ do not represent functions. Having this assumption about where statement, we can say that the where statement is dual to the let statement

let G in F

For this case we have

$[\![F$ where $v_1, \dots, v_n$ so that $G]\!]_{Isab} =$ let $[\![G]\!]_{Isab}$ in $([\![F]\!]_{Isab})$

3 Steam Boiler Specification in Focus

The steam boiler has a water tank, which contains a number of gallons of water, and a pump, which adds 10 gallons of water per time unit to its water tank, if the pump is on. At most 10 gallons of water are consumed per time unit by the steam production, if the pump is off. The steam boiler has a sensor that measures the water level.

In this section we discuss first of all the data types used in the case study. Then the FOCUS specifications of the components and their translation into Isabelle/HOL using the schema from the methodology "FOCUS on Isabelle/HOL" will be presented. The proof of the behavioral refinement relation between the requirements specification and the architecture specification will be discussed, as well as the resulting Isabelle/HOL theories `SteamBoiler.thy` and `SteamBoiler_proof.thy` are presented in Section 6.

3.1 Data Types

The original steam boiler specification [1] in FOCUS uses two additional datatypes:

$$\begin{aligned} \mathsf{type}\ \mathit{Gallons} &= \{r \in \mathbb{R} \mid 0 \le r \le 1000\} \\ \mathsf{type}\ \mathit{Switch} &= \{-1, +1\} \end{aligned}$$

We replace in the whole steam boiler specification [1] the type *Gallons* by the type $\mathbb{N}$, because representation the water level by a rational number as well as the restriction of upper bound 1000 does not play an important role in this specification.

Introduction of the datatype *Switch* in the specification [1] aims to combine a number with arithmetical operations ("+" and "−") over it, to have shorter and clearer representation. But the use thies type leads the problems with type checking. We will replace the type *Switch* by the type $\mathbb{B}$it (1 for"+1", 0 for "−1"). Thus, instead of

$$o.j + (x.j) * r,$$

where $x.j$ was of type *Switch* we will use

$$\mathsf{if}\ r = 0\ \mathsf{then}\ o.j - r\ \mathsf{else}\ o.j + r\ \mathsf{fi}\ .$$

Because we use here not only time-synchronous specifications, but also timed ones, we need to represent all of them in the *timed* frame and make corresponding changes according the methodology "FOCUS on Isabelle/HOL" to avoid the type checking mistakes.[1]

To have explicit difference between $0, 1 : \mathbb{N}$ and $0, 1 : \mathbb{B}\text{it}$ we define in Isabelle/HOL the type $\mathbb{B}$it as follows:

datatype *bit = Zero | One*

3.2 Requirement Specification

The specification *ControlSystem* describes the interface and represents the requirements to the component: in each time unit the system outputs it current water level in gallons and this level should always be between 200 and 800 gallons. The original FOCUS specification of the control system component [1] is the following one:

[1]Timed and time-synchronous streams have different syntax and cannot be combined together directly.

ControlSystem — time-synchronous

out $o : Gallons$

$\forall j \in \mathbb{N}_+ : 200 \leq o.j \leq 800$

We make a number of changes vs. original specification (according our methodology), and get as result the following Focus specification of the component *ControlSystem*:

ControlSystem — timed

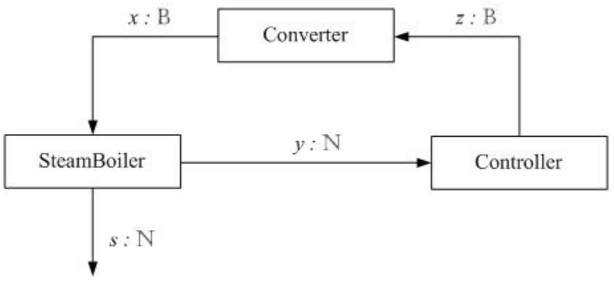
out $s : \mathbb{N}$

$\mathsf{ts}(s)$
$\forall j \in \mathbb{N} : 200 \leq \mathsf{ft.ti}(s, j) \leq 800$

3.3 Architecture Specification

The specification *ControlSystemArch* describes one possible architecture of the steam boiler system. The system consists of three components: a steam boiler, a converter, and a controller.

ControlSystemArch — glass-box



The corresponding Focus representation of this specification as plane text (constraint style):

ControlSystemArch — timed

out $s : \mathbb{N}$

loc $x,\ z : \mathbb{B}\mathrm{it};\ \ y : \mathbb{N}$

$(s,\ y) := SteamBoiler(x)$
$(z) := Controller(y)$
$(x) := Converter(z)$

3.4 Steam Boiler Component

The specification *SteamBoiler* describes steam boiler component. The steam boiler works in time-synchronous manner: the current water level is controlled every time unit. The boiler has two output channels with equal streams ($y = s$) and it fixes the initial water level to be 500 gallons. For every point of time the following must be true: if the pump is off, the boiler consumes a most 10 gallons of water, otherwise (the pump is on) at most 10 gallons of water will be added to its water tank. The original Focus specification of this component [1] is the following one:

SteamBoiler — time-synchronous

in $x : \mathit{Switch}$

out $y, o : \mathit{Gallons}$

$y = s \;\wedge\; s.1 = 500$
$\forall\, j \in \mathbb{N}_{+} : \exists\, r \in \mathit{Gallons} : 0 < r \leq 10 \;\wedge$
$\quad s.(j+1) = o.j + (x.j) * r$

We make a number of changes vs. original specification (according our methodology), and get as result

SteamBoiler — timed

in $x : \mathbb{B}\mathrm{it}$

out $y, s : \mathbb{N}$

asm $\mathsf{ts}(x)$

gar $\mathsf{ts}(y)$
$\mathsf{ts}(s)$
$y = s \;\wedge\; \mathsf{ft.ti}(s, 0) = 500$
$\forall\, j \in \mathbb{N} : \exists\, r \in \mathbb{N} : 0 < r \leq 10 \;\wedge$
$\quad \mathsf{ft.ti}(s, j+1) = \mathsf{if}\;\; \mathsf{ft.ti}(x, j) = 0 \;\;\mathsf{then}\;\; \mathsf{ft.ti}(s, j) - r \;\;\mathsf{else}\;\; \mathsf{ft.ti}(s, j) + r \;\;\mathsf{fi}$

3.5 Converter Component

The specification *Converter* describes the converter component. It converts the asynchronous output produced by the controller to time-synchronous input for the steam boiler. Initially the pump is off, and at every later point of time (from the receiving the first instruction from the controller) the output will be the latest input from the controller ($\overline{z\downarrow_t}.(\#\overline{z\downarrow_t})$). The original Focus specification of this component [1] is the following one:

Converter — timed

in $z : \mathit{Switch}$

out $x : \mathit{Switch}$

$\mathsf{ts}(x) \;\wedge\; \forall\, t \in \mathbb{N} : \overline{x}.(t+1) = \mathsf{if}\;\; \overline{z\downarrow_t} = \langle\rangle \;\;\mathsf{then}\;\; -1 \;\;\mathsf{else}\;\; \overline{z\downarrow_t}.(\#\overline{z\downarrow_t}) \;\;\mathsf{fi}$

Applying the rules of the methodology we get the following Focus specification:

Converter — timed

in $z : \mathbb{B}\mathrm{it}$

out $x : \mathbb{B}\mathrm{it}$

$\mathsf{ts}(x) \;\wedge\; \forall\, t \in \mathbb{N} : \mathsf{ft.ti}(x, t) = \mathsf{if}\;\; \overline{z\downarrow_t} = \langle\rangle \;\;\mathsf{then}\;\; 0 \;\;\mathsf{else}\;\; \overline{z\downarrow_t}.(\#\overline{z\downarrow_t}) \;\;\mathsf{fi}$

3.6 Controller Component

The specification *Controller* describes the controller component. Contrary to the steam boiler the controller behaves in a purely asynchronous manner to keep the number of control signals small, it means it might not be desirable to switch the pump on and off more often than necessary. The controller is responsible for switching the steam boiler pump on and off.

If the pump is off ($\mathit{off}(\langle r \rangle \frown y)$): if the current water level is above 300 gallons the pump stays be off ($\langle \surd \rangle \frown \text{off(y)}$), otherwise the pump is started ($\langle +1 \rangle \frown \langle \surd \rangle \frown \mathit{on}(y)$) and will run until the water level reaches 700 gallons.

If the pump is on ($\mathit{on}(\langle r \rangle \frown y)$): if the current water level is below 700 gallons the pump stays be on ($\langle \surd \rangle \frown \text{on(y)}$), otherwise the pump is turned off ($\langle -1 \rangle \frown \langle \surd \rangle \frown \mathit{off}(y)$) and will be off until the water level reaches 300 gallons.

The original Focus specification of this component [1] is the following one:

Controller — timed

in $y : \mathit{Gallons}$

out $z : \mathit{Switch}$

$z = \langle -1 \rangle \frown \langle \surd \rangle \frown \mathit{off}(\overline{y})$

where off so that $\forall\, r \in \mathit{Gallons};\ \ y \in \mathit{Gallons}^{\,\omega}$:

$\mathit{off}(\langle r \rangle \frown y) =$ if $r > 300$ then $\langle \surd \rangle \frown \mathit{off}(y)$ else $\langle +1 \rangle \frown \langle \surd \rangle \frown \mathit{on}(y)$ fi

$\mathit{on}(\langle r \rangle \frown y) =$ if $r < 700$ then $\langle \surd \rangle \frown \mathit{on}(y)$ else $\langle -1 \rangle \frown \langle \surd \rangle \frown \mathit{off}(y)$ fi

The specification of the controller is be a *timed* one, but making a proof in Isabelle/HOL that the architecture specification *ControlSystemArch* is a behavioral refinement of the requirement specification *ControlSystem*, we found out that to argue about its properties we need an assumption about the input stream y – that the stream y is a time-synchronous one: $\mathsf{ts}(y)$. In the steam boiler system specified by *ControlSystemArch* this assumption for the component *Controller* will be always true because the stream y is an output stream of the time-synchronous component *SteamBoiler*, but if we look at the component[2] *Controller* without taking into account its later combination with the component *SteamBoiler*, then we may have some problems. More precisely, the specification *Controller* says, that when some input message comes, the increment by 1 of the global digital clock ($\surd$ in the output stream) happens. This means following:

✓ If in some time interval there is no message in the input stream y, no increment of the clock happens.

✓ If in some time interval a number of messages comes, the clock will be incremented ones *for each* message.

Both cases lead to the wrong behavior of the component *Controller*, but for the steam boiler controller system it never happens, because the stream y is an output stream of the time-synchronous component *SteamBoiler*.

To argue about properties of controller and to have a possibility to reuse the component later, we extend the specification *Controller* by assumption predicate $\mathsf{ts}(y)$ to the specification *ControllerExt* (the data types *Gallons* and *Switch* are also replaced by $\mathbb{N}$ and $\mathbb{B}$it respectively).

[2]This is true also for the case of service semantics.

```
Controller                                                    timed
in    $y : \mathbb{N}$
out   $z : \mathbb{B}\mathrm{it}$
-------------------------------------------------------------------
asm   $\mathsf{ts}(y)$
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
gar
$z = \langle 0 \rangle \frown \langle \surd \rangle \frown \mathit{off}(\overline{y})$
where $\mathit{off}$ so that $\forall\, r \in \mathbb{N};\ \ y \in \mathbb{N}^{\omega}$ :
   $\mathit{off}(\langle r \rangle \frown y) =$ if $r > 300$ then $\langle \surd \rangle \frown \mathit{off}(y)$ else $\langle 1 \rangle \frown \langle \surd \rangle \frown \mathit{on}(y)$ fi
   $\mathit{on}(\langle r \rangle \frown y)\ =$ if $r < 700$ then $\langle \surd \rangle \frown \mathit{on}(y)$ else $\langle 0 \rangle \frown \langle \surd \rangle \frown \mathit{off}(y)$ fi
```

Remark: Please note, that the specification *ControllerExt* is not a behavioral refinement of component *Controller*.

The specification above uses mutually recursive functions *on* and *off* to specify the local state of the component, more precisely, the state of the steam boiler pump. Because the translation of such functions to Isabelle/HOL leads to problems, we need to reformulate the specification *ControllerExt* into a semantically equal specification that uses local states instead of mutually recursive functions. Therefore, we introduce a new local variable $l \in \mathbb{B}\mathrm{it}$ (0 corresponds to “off”, 1 corresponds to “on”) and set it initially to 0. Thus, we get Focus specification *Controller1* that is semantically equal to the Focus specification *ControllerExt*.

Accordig to our methodology, we can rewrite the formula that represents semantics of the tiTable *ControllerT1* to the formula

```
if $l = 0$
then if $300 < r$
        then $\mathsf{ti}(z, t+1) = \langle\rangle \wedge l' = 0$
        else $\mathsf{ti}(z, t+1) = \langle 1 \rangle \wedge l' = 1$ fi
else  if $r < 700$
        then $\mathsf{ti}(z, t+1) = \langle\rangle \wedge l' = 1$
        else $\mathsf{ti}(z, t+1) = \langle 0 \rangle \wedge l' = 0$ fi
fi
where $r = \mathsf{ft.ti}(y, t)$
```

The proof that the formula above describes the behavior of the mutually recursive functions *on* and *off* from the specifications *Controller* and *ControllerExt* is straightforward.

Controller1 timed

in $y : \mathbb{N}$

out $z : \mathbb{B}\mathrm{it}$

local $l \in \mathbb{B}\mathrm{it}$
univ $r \in \mathbb{N}$

init $l = 0$

asm
ts(y)

gar
ti$(z, 0) = \langle 0 \rangle$

tiTable *ControllerT*1: $\forall\, t \in \mathbb{N}$

	y	z'	l'	Assumption
1	$\langle r \rangle$	$\langle\rangle$	0	$300 < r \ \wedge \ l = 0$
2	$\langle r \rangle$	$\langle 1 \rangle$	1	$r \leq 300 \ \wedge \ l = 0$
3	$\langle r \rangle$	$\langle\rangle$	1	$r < 700 \ \wedge \ l = 1$
4	$\langle r \rangle$	$\langle 0 \rangle$	0	$700 \leq r \ \wedge \ l = 1$

4 Correctness of the Relations between Sets of Channels

Correctness of the relations between the sets of input and output channels[3], will be shown in Isabelle/HOL separately: first of all for a specification group[4] *SGroup* the theory ***SGroup_types.thy***, which contains all user-defined datatypes (including the names of single components and channel identifiers) used in the the specification group. The Isabelle/HOL theory ***inout.thy*** is defined as a general one (part of the deep embeding) and can be used without any changes (except the name of the theory ***SGroup*** that can be seen as paramether). We define here the signatures of the functions subcomponents, identifiers of input, output and local channels (or sheafs of channels), as well as all predicates that specify the correctnes properties of the relations between the sets of channels. The proof schema is standard, is part of the methodology "Focus on Isabelle/HOL" and can be used automatically. If the proof does not hold by this schema, the specification of corresponding set is incorrect and must be changed.

In the specification group *SteamBoiler* will be used

✓ components *ControlSystem* (specification of requirements), *ControlSystemArch* (specification of system architecture), *SteamBoiler*, *Controller*, and *Controller*;

✓ channels s, x, y and z.

[3]For example, the sets of input, output and local channles must be pairwise disjoint, every local stream l of the composite system S must be both an input stream of some its subcomponent S_i, and at the same time an output stream of some its subcomponent S_j $(i \neq j)$, etc.

[4]We will use the expression *specification group* to denote the set of specifications of the system on all refinement steps (both requirements and architecture specifications), and specifications of all subcomponents (or, more precisely, all subcomponents trees) on all refinement steps.

Thus, the steam boiler system is relative small, and all its components can be specified as a joined Isabelle/HOL theory (see Section **??**). We present the following Isabelle/HOL theories for this system:

- ✓ *SteamBoiler_types.thy* – the data type definitions,
- ✓ *SteamBoiler_inout.thy* – the specification of component interfaces (relations between the sets of input and output channels) and the corresponding correctness proofs,
- ✓ *SteamBoiler.thy* – the Isabelle/HOL specifications of the system components,
- ✓ *SteamBoiler_proof* – the proof of refinement relation between the requirements and the architecture specifications.

The Isabelle/HOL specification of these types and (component and channel) identifiers is presented below by the theory *SteamBoiler_types.thy*. This theory does not contain any other type definitions, because the specification group *SteamBoiler* does not use any non-standard datatypes.

Remark: Please note, that the theory *inout.thy* (see Section **??**) must be copied into the same folder as the theory *stream.thy*, *SteamBoiler_types.thy*, *SteamBoiler_inout.thy* etc. The "SpecificationGroupName" must be replaced by the name of specification group – by *SteamBoiler*.

```
theory SteamBoiler_types = Main + stream:

datatype specID =
           sControlSystem
         | sControlSystemArch
         | sSteamBoiler
         | sController
         | sConverter

datatype chanID =  ch_s   | ch_x   | ch_y   | ch_z
end
```

The theory *SteamBoiler_inout.thy* is based only on the theory *inout.thy*. First of all we specify the subcomponents relations for all components of the system by the function *subcomponents*. For all elementary components (specifications) the list of subcomponents must be empty. Then we specify the list of input, output and local channels for all components by the functions *ins*, *out* and *loc* respectively. After that we prove that the predicates *correctInOutLoc* and *correctComposition* hold for all components, and also that the predicates *correctCompositionIn*, *correctCompositionOut* and *correctCompositionLoc* holds for composite components (specifications).

```
theory SteamBoiler_inout = Main + inout:

primrec
  "subcomponents sControlSystem = []"
  "subcomponents sControlSystemArch =
     [sSteamBoiler, sController, sConverter]"
  "subcomponents sSteamBoiler = []"
  "subcomponents sController = []"
  "subcomponents sConverter = [] "

primrec
  "ins sControlSystem = []"
  "ins sControlSystemArch = []"
  "ins sSteamBoiler = [ch_x]"
  "ins sController = [ch_y]"
  "ins sConverter = [ch_z]"

primrec
  "loc sControlSystem = []"
  "loc sControlSystemArch = [ch_x, ch_y, ch_z]"
  "loc sSteamBoiler = []"
  "loc sController = []"
  "loc sConverter = []"

primrec
  "out sControlSystem = [ch_s]"
  "out sControlSystemArch = [ch_s]"
  "out sSteamBoiler = [ch_y, ch_s]"
  "out sController = [ch_z]"
  "out sConverter = [ch_x]"

Proofs for components

lemma spec_ControlSystem1:
"correctInOutLoc sControlSystem"
  by (simp add: correctInOutLoc_def disjoint_def)

lemma spec_ControlSystem2:
"correctComposition sControlSystem"
  by (simp add: correctComposition_def)
lemma spec_ControlSystemArch1:
"correctInOutLoc sControlSystemArch"
  by (simp add: correctInOutLoc_def disjoint_def)

lemma spec_ControlSystemArch2:
"correctComposition sControlSystemArch"
  by (simp add: correctComposition_def disjoint_def)

lemma spec_ControlSystemArch3:
"correctCompositionIn sControlSystemArch"
  by (simp add: correctCompositionIn_def disjoint_def)
lemma spec_ControlSystemarch4:
"correctCompositionOut sControlSystemArch"
  by (simp add: correctCompositionOut_def disjoint_def)

lemma spec_ControlSystemArch5:
"correctCompositionLoc sControlSystemArch"
  by (simp add: correctCompositionLoc_def, auto)
```

```
lemma spec_SteamBoiler1:
"correctInOutLoc sSteamBoiler"
  by (simp add: correctInOutLoc_def disjoint_def)

lemma spec_SteamBoiler2:
"correctComposition sSteamBoiler"
  by (simp add: correctComposition_def)

lemma spec_Controller1:
"correctInOutLoc sController"
  by (simp add: correctInOutLoc_def disjoint_def)

lemma spec_Controller2:
"correctComposition sController"
  by (simp add: correctComposition_def)

lemma spec_Converter1:
"correctInOutLoc sConverter"
  by (simp add: correctInOutLoc_def disjoint_def)

lemma spec_Converter2:
"correctComposition sConverter"
  by (simp add: correctComposition_def)
end
```

5 Steam Boiler Specification in Isabelle/HOL

Translating the Focus specifications of all components from the specification group *SteamBoiler* schematically (according the methodology), we get the Isabelle/HOL theory below.

```
theory SteamBoiler = Main + stream + Bit:

constdefs ControlSystem :: "nat istream ⇒ bool"
 "ControlSystem s ≡
  (ts s) ∧
   (∀ (j::nat). (200::nat) ≤ hd (s j) ∧ hd (s j) ≤ (800:: nat))"

constdefs
  SteamBoiler :: "bit istream ⇒ nat istream ⇒ nat istream ⇒ bool"
 "SteamBoiler x s y ≡
  ts x
  ⟶
  ((ts y) ∧ (ts s) ∧ (y = s) ∧
   (hd (s 0) = (500::nat)) ∧
   (∀ (j::nat). (∃ (r::nat).
               (0::nat) < r ∧ r ≤ (10::nat) ∧
               hd (s (Suc j)) =
                  (if hd (x j) = Zero
                   then (hd (s j)) - r
                   else (hd (s j)) + r)) ))"

constdefs
  Converter :: "bit istream ⇒ bit istream ⇒ bool"
 "Converter z x
  ≡
```

```
(ts x)
∧
(∀ (t::nat).
  hd (x t) =
      (if  (fin_make_untimed (inf_truncate z t) = [])
       then
           Zero
       else
           (fin_make_untimed (inf_truncate z t)) !
               ((length (fin_make_untimed (inf_truncate z t))) - (1::nat))
    ))"
```

constdefs
```
  Controller_L ::
    "nat istream ⇒ bit iustream ⇒ bit iustream ⇒ bit istream ⇒ bool"
 "Controller_L y lIn lOut z
  ≡
  (z 0 = [Zero])
  ∧
  (∀ (t::nat).
  ( if (lIn t) = Zero
    then ( if 300 < hd (y t)
           then  (z t) = []    ∧ (lOut t) = Zero
           else  (z t) = [One] ∧ (lOut t) = One
          )
    else ( if  hd (y t) < 700
           then  (z t) = []     ∧ (lOut t) = One
           else  (z t) = [Zero] ∧ (lOut t) = Zero ) ))"
```

constdefs
```
  Controller :: "nat istream ⇒ bit istream ⇒ bool"
 "Controller y z
  ≡
  (ts y)
  ⟶
  (∃ l. Controller_L y (fin_inf_append [Zero] l) l z)"
```

constdefs
```
  ControlSystemArch :: "nat istream ⇒ bool"
 "ControlSystemArch s
  ≡
  ∃ x z :: bit istream. ∃ y :: nat istream.
    ( SteamBoiler x s y ∧ Controller y z ∧ Converter z x )"
```

end

6 Proof of the Steam Boiler System Properties

To show that the specified system fulfill the requirements we need to show that the specification *ControlSystemArch* is a refinement of the specification *ControlSystem*. It follows from the definition of behavioral refinement that in order to verify that

$$ControlSystem \leadsto ControlSystemArch \qquad (1)$$

it is enough to prove that

$$[\![ControlSystemArch]\!] \;\Rightarrow\; [\![ControlSystem]\!] \tag{2}$$

Therefore, we have to define and to prove a *lemma* (let us call it `L0_ControlSystem`), that says the specification *ControlSystemArch* is a refinement of the specification *ControlSystem*:

```
lemma L0_ControlSystem:
"⋀ s. ⟦ ControlSystemArch s⟧ ⟹ ControlSystem s"
```

Proof of this lemma as well as proofs of all auxiliary lemmas are given in the Isabelle/HOL theory below.

```
theory SteamBoiler_proof = Main + SteamBoiler:
```

6.1 Properties of Controller Component

```
lemma L1_Controller:
"⋀ y l z.
⟦ Controller_L y (fin_inf_append [Zero] l) l z;
  l t ≠ Zero ⟧
 ⟹ last (fin_make_untimed (inf_truncate z t)) = One"
  apply (induct t)
  apply (simp add: Controller_L_def)
  apply clarify
  apply (erule_tac x="0" in allE)
  apply (simp split add: split_if_asm)
  apply atomize
  apply (erule_tac x="y" in allE)
  apply (erule_tac x="l" in allE)
  apply (erule_tac x="z" in allE)
  apply simp
  apply (subgoal_tac
   "Controller_L y (fin_inf_append [Zero] l) l z")
    prefer 2
    apply simp
  apply (simp add: Controller_L_def)
  apply clarify
  apply (erule_tac x="Suc t" in allE)
  apply (erule_tac x="t" in allE)
  apply (simp split add: split_if_asm)
  apply clarify
  apply (simp add: fin_make_untimed_append_empty fin_inf_append_def)
  apply (simp add: fin_make_untimed_def)+
  apply clarify
  apply (simp add: fin_inf_append_def)
  apply (simp add: fin_make_untimed_def)+
  apply clarify
  apply (simp add: fin_inf_append_def)
  done
```

```
lemma L2_Controller:
"⋀ y lIn lOut z.
⟦ Controller_L y (fin_inf_append [Zero] l) l z;
  l t = Zero ⟧
```

```
 ⟹ last (fin_make_untimed (inf_truncate z t)) = Zero"
  apply (induct t)
  apply (simp add: Controller_L_def)
  apply clarify
  apply (erule_tac x="0" in allE)
  apply (simp split add: split_if_asm)
  apply (simp add: fin_make_untimed_def)+
  apply atomize
  apply (erule_tac x="z" in allE)
  apply simp
  apply (subgoal_tac
    "Controller_L y (fin_inf_append [Zero] l) l z")
    prefer 2
    apply simp
  apply (simp add: Controller_L_def)
  apply clarify
  apply (erule_tac x="Suc t" in allE)
  apply (erule_tac x="t" in allE)
  apply (simp split add: split_if_asm)
  apply (simp add: correct_fin_inf_append1)+
  done


lemma L3_Controller:
"⟦ Controller_L y (fin_inf_append [Zero] l) l z⟧
 ⟹
 last (fin_make_untimed (inf_truncate z t)) =  l t"
  apply (case_tac "l t")
  apply (simp add: L1_Controller L2_Controller)+
  done

lemma L4_Controller:
"⟦ Controller_L s (fin_inf_append [Zero] l) l z ⟧
 ⟹ fin_make_untimed (inf_truncate z i) ≠ []"
  apply (simp add: Controller_L_def)
  apply (subgoal_tac "∀ i. 0 ≤ i ⟶ fin_make_untimed (inf_truncate z i) ≠
[]")
    prefer 2
    apply (simp add: fin_make_untimed_inf_truncate_Nonempty_all0a)
  apply simp
  done
```

6.2 Properties of the System

```
lemma L1_ControlSystem:
"⋀ s. ControlSystemArch s ⟹ (200::nat) ≤ hd (s i)"
  apply (simp only: ControlSystemArch_def)
  apply clarify
  apply (induct i)
  apply (simp add: SteamBoiler_def)
  apply (simp add: Converter_def)
  apply atomize
  apply (erule_tac x="s" in allE)
  apply (erule_tac x="x" in allE)
  apply (erule_tac x="z" in allE)
  apply (erule_tac x="y" in allE)
  apply simp
  apply (simp add: SteamBoiler_def Controller_def Converter_def)
  apply clarify
  apply (erule_tac x="i" in allE)+
  apply clarify
  apply (simp split add: split_if_asm)

  apply (simp add: L4_Controller)

  apply (subgoal_tac "last (fin_make_untimed (inf_truncate z i)) =  l i")
    prefer 2
    apply (simp add: L3_Controller)
  apply (simp add: Controller_L_def)
  apply clarify
  apply (erule_tac x="i" in allE)
  apply (simp split add: split_if_asm)
  apply arith
  apply (simp add: fin_make_untimed_nth_length)
  apply (simp add: last_nth_length)
  apply arith
  done


lemma L2_ControlSystem:
"⋀ s. ControlSystemArch s ⟹ hd (s i) ≤ (800:: nat)"
  apply (simp only: ControlSystemArch_def)
  apply clarify
  apply (induct i)
  apply (simp add: SteamBoiler_def)
  apply (simp add: Converter_def)
  apply atomize
  apply (erule_tac x="s" in allE)
  apply (erule_tac x="x" in allE)
  apply (erule_tac x="z" in allE)
  apply (erule_tac x="y" in allE)
  apply simp
  apply (simp add: SteamBoiler_def Controller_def Converter_def)
  apply clarify
  apply (erule_tac x="i" in allE)
  apply (erule_tac x="i" in allE)
  apply clarify
  apply (simp split add: split_if_asm)

  apply (simp add: L4_Controller)

  apply (subgoal_tac "last (fin_make_untimed (inf_truncate z i)) =  l i")
    prefer 2
    apply (simp add: L3_Controller)
  apply (simp add: Controller_L_def)
  apply clarify
  apply (erule_tac x="i" in allE)
  apply (simp split add: split_if_asm)
  apply arith+

  apply (subgoal_tac "last (fin_make_untimed (inf_truncate z i)) =  l i")
    prefer 2
    apply (simp add: L3_Controller)
  apply (simp add: Controller_L_def)
  apply clarify
  apply (erule_tac x="i" in allE)
  apply (simp split add: split_if_asm)
  apply (simp add: last_nth_length)+
done
```

6.3 Proof of the Refinement Relation

```
lemma L0_ControlSystem:
"⋀ s. ⟦ ControlSystemArch s⟧ ⟹ ControlSystem s"
   apply (simp add: ControlSystem_def)
   apply auto
   apply (simp add: ControlSystemArch_def)
   apply (simp add: SteamBoiler_def)
   apply (simp add: Converter_def)
   apply auto
   apply (simp add: L1_ControlSystem)
   apply (simp add: L2_ControlSystem)
   done
```

7 Conclusions

This paper represents the extension of the steam boiler specification from [1], its translation in Isabelle/HOL [2] and its formal verification according to the methodology "Focus on Isabelle/HOL".The case study has shown the feasibility of the methodology "Focus on Isabelle/HOL": the translation from a timed Focus specification to the specification in Isabelle/HOL can be done in the schematical way. The corresponding proof in Isabelle/HOL has shown that the Focus specification of the steam boiler architecture is a refinement of the steam boiler requirements specification and the proof that the architecture specification fulfill the requirements specification, i.e. is its behavioral refinement.

Proving this relation in Isabelle/HOL, we found out that to argue about properties of the Controller component of the system we need an additional (vs. original specification from [1]) assumption about the input stream y – that the stream y is a time-synchronous one.

The correctness of the input/output relations was also proven for all components of the system.

References

[1] Broy, M., and Stølen, K. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* 2001.

[2] Nipkow, T., Paulson, L. C., and Wenzel, M. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, vol. 2283 of *LNCS.* Springer, 2002.

[3] Wenzel, M. *The Isabelle/Isar Reference Manual.* Technische Universität München, 2004. Part of the Isabelle distribution.