

The first part of this module defines the TLA+ representation of the abstract program *ICen1* of Figure 7.4 of the book “A Science of Computer Programs” by *Leslie Lamport*. The second part defines a program *ICen1_p* by adding a prophecy *p* variable to *ICen1* as described in Section 7.4.1, and it asserts that *ICen1_p* implements *ICen2* under a suitable refinement mapping. For that purpose, it instantiates module *ICen2* that defines program *ICen2*. That second part is commented out (by ending the module before it) when this module is instantiated by *ICen2*, which is done when *ICen2* is the root module.

Proving that *ICen1_p* implements *ICen2* under a refinement mapping shows that $\exists aw : ICen1$ implies $\exists aw : ICen2$. To write that theorem in TLA+, we need to define a new module that declares the constants *Art* and *NotArt* and the variables *inp* and *disp* and instantiates modules *ICen1* and *ICen2* as follows:

$$\begin{aligned} I1(aw) &\triangleq \text{INSTANCE } ICen1 \\ I2(aw) &\triangleq \text{INSTANCE } ICen2 \end{aligned}$$

In that module, we can then write $\exists aw : ICen1$ as $\exists aw : I1(a2)!ICen1$ and can write *ICen2* similarly. However, since *TLC* and *TLAPS* can't handle the \exists operator, there is no practical reason to do that. As the book says in Section 7.1.1:

\exists is of little use in practice. The only role it plays is telling us that, when implementing the program, it doesn't matter how the internal variables are refined. That can be stated just as well in a comment.

EXTENDS *Integers, Sequences*

We declare *Art* and *NotArt* to be constants and assume that *NotArt* is not an element of *Art*. (We don't have to declare that *Art* is a set because in *ZF* set theory, on which TLA+ is based, every value is a set.)

CONSTANT *Art, NotArt*
ASSUME *NotArt* \notin *Art*

VARIABLES *inp, aw, disp*
 $v \triangleq \langle inp, aw, disp \rangle$

It's a good idea to define a type-correctness invariant after declaring the variables to help the reader understand the definitions of the initial predicate and next-step predicate.

$$\begin{aligned} TypeOK &\triangleq \wedge inp \in Art \cup \{NotArt\} \\ &\quad \wedge disp \in Art \times \{0, 1\} \\ &\quad \wedge aw \in Seq(Art) \end{aligned}$$

$$\begin{aligned} Init &\triangleq \wedge inp = NotArt \\ &\quad \wedge aw = \langle \rangle \\ &\quad \wedge disp \in Art \times \{0, 1\} \end{aligned}$$

$$\begin{aligned} Input &\triangleq \wedge (inp = NotArt) \wedge (aw = \langle \rangle) \\ &\quad \wedge inp' \in Art \\ &\quad \wedge aw' = \langle inp' \rangle \\ &\quad \wedge disp' = disp \end{aligned}$$

$$\begin{aligned} DispOrNot &\triangleq \wedge aw \neq \langle \rangle \\ &\quad \wedge \vee disp' = \langle aw[1], 1 - disp[2] \rangle \end{aligned}$$

$$\begin{aligned} & \vee \text{disp}' = \text{disp} \\ & \wedge \text{aw}' = \langle \rangle \\ & \wedge \text{inp}' = \text{inp} \end{aligned}$$

$$\begin{aligned} \text{Ack} & \triangleq \wedge (\text{inp} \in \text{Art}) \wedge (\text{aw} = \langle \rangle) \\ & \wedge \text{inp}' = \text{NotArt} \\ & \wedge (\text{aw}' = \text{aw}) \wedge (\text{disp}' = \text{disp}) \end{aligned}$$

$$\text{Next1} \triangleq \text{Input} \vee \text{Ack} \vee \text{DispOrNot}$$

Although not done in the book, we now define a fairness requirement for the *ICen1* program. It is weak fairness of all the actions except the Input action, since we don't want to require that the artist keep submitting works of art.

$$\text{Fairness1} \triangleq \text{WF}_v(\text{Ack} \vee \text{DispOrNot})$$

$$\text{ICen1} \triangleq \text{Init} \wedge \Box[\text{Next1}]_v \wedge \text{Fairness1}$$

When *ICen1* is the root module, comment out the following line that ends the module at this point.

```

(***** )
(* We now add the simple prophecy variable p to ICen1 as it is done in *)
(* Section 7.4.1 of the book. For simplicity, we use the strings “Yes” *)
(* and “No” instead of the constants Yes and No used in the book. *)
(***** )

VARIABLE p
Pi  $\triangleq$  {“Yes”, “No”}

v_p  $\triangleq$   $\langle \text{inp}, \text{aw}, \text{disp}, p \rangle$ 

TypeOK_p  $\triangleq$  TypeOK  $\wedge$  (p  $\in$  Pi)

Init_p  $\triangleq$  Init  $\wedge$  (p  $\in$  Pi)

Input_p  $\triangleq$  Input  $\wedge \exists i \in \text{Pi} : p' = i$ 

Ack_p  $\triangleq$  Ack  $\wedge$  (p' = p)

DorN(i)  $\triangleq$   $\wedge \text{aw} \neq \langle \rangle$ 
 $\wedge \vee (i = \text{“Yes”}) \wedge (\text{disp}' = \langle \text{aw}[1], 1 - \text{disp}[2] \rangle)$ 
 $\vee (i = \text{“No”}) \wedge (\text{disp}' = \text{disp})$ 
 $\wedge \text{aw}' = \langle \rangle$ 
 $\wedge \text{UNCHANGED inp}$ 

DispOrNot_p  $\triangleq$  DorN(p)  $\wedge$  (p'  $\in$  Pi)

Next1_p  $\triangleq$  Input_p  $\vee$  Ack_p  $\vee$  DispOrNot_p

ICen1_p  $\triangleq$  Init_p  $\wedge \Box[\text{Next1}_p]_{v_p} \wedge \text{Fairness1}$ 

\ *ICen1Live_p  $\triangleq$  ICen1_p  $\wedge$  Fairness1

```

```

(***** )
(* We now define I!ICen2 to be (ICen2 WITH aw  $\leftarrow$  awBar), where awBar is *)
(* defined in Section 7.4.1. *)

```

```

(*****
awBar  $\triangleq$  IF (aw  $\neq$   $\langle \rangle$ )  $\wedge$  (p = "Yes") THEN aw ELSE  $\langle \rangle$ 
I  $\triangleq$  INSTANCE ICen2 WITH aw  $\leftarrow$  awBar
THEOREM ICen1_p  $\Rightarrow$  I!ICen2
-----
\ * Modification History
\ * Last modified Wed Oct 16 15:58:34 CEST 2024 by lamport
\ * Created Fri Oct 21 06:36:45 PDT 2022 by lamport
-----
\ * Modification History
\ * Created Fri May 05 16:41:44 CEST 2023 by lamport

```