Applying NVP and RB patterns

A. Iliasov and A. Romanovsky

Newcastle University, Newcastle Upon Tyne, England

To demonstrate how the NVP and RB patterns transform an input model we apply each of them to a specification representing a simple storage device. The abstract specification was created manually. The two refinements were produced automatically by the patterns without any further editing or changes. The abstract model is as following

system *memory*
variables
  mem
invariant
  $mem \in 0 \mathinner{..} 255$
initialisation
  $mem :\in 0 \mathinner{..} 255$
events
  store = **begin**
    $mem :\in 0 \mathinner{..} 255$
  end

  reset = **begin**
    $mem := 0$
  end

In this model, a memory can be updated with the store operation. The read operation is implicit since it does not affect the model state.

The following concrete model is obtained by applying the recovery blocks pattern with configuration $(b = store, n = 2)$

system *memory_rb*
variables
$mem, store\_branch, store\_stage, mem\_chkp$
invariant
$mem \in 0 \,..\, 255$
$store\_branch \in \mathbb{N}$
$store\_stage \in 0 \,..\, 2$
$mem\_chkp \in 0 \,..\, 255$
initialisation
$mem :\in 0 \,..\, 255$
$store\_branch := 0$
$store\_stage := 0$
$mem\_chkp :\in 0 \,..\, 255$

$store$ = **when** $(store\_stage = 2)$ **then** $mem := mem\_chkp$ **end**

$reset$ = **when** $(store\_branch = 2)$ **then** $mem := 0$ **end**

$store\_chkp$ = **when** $(store\_stage = 0)$ **then** $mem\_chkp := mem$ $store\_stage := 1$ **end**

$store\_alt\_1$ = **when** $(store\_stage = 1) \wedge (store\_branch = 0)$ **then** $mem\_chkp :\in 0 \,..\, 255$ $store\_stage := 2$ **end**

$store\_test\_fail$ = **when** $(store\_stage = 2) \wedge (mem\_chkp \notin 0 \,..\, 255)$ **then** $store\_branch := store\_branch + 1$ $store\_stage := 0$ **end**

$store\_alt\_2$ = **when** $(store\_stage = 1) \wedge (store\_branch = 1)$ **then** $mem\_chkp :\in 0 \,..\, 255$ $store\_stage := 2$ **end**

The new model has a checkpointing, two alternative branches - which are identical at this stage, - and a voting event. A further refinement of this model may diversify alternatives by introducing

different writing algorithms (e.g. an optimistic and pessimistic). It is also possible to decompose the model at this step, using the Event-B decomposition method, and conduct independent developments for each of the alternatives.

The N-version programming pattern can be used to model redundancy when the same information is stored in different ways. By applying the pattern to the abstract model with the configuration $(b = store, n = 3)$ we get the following model

system *memory_nvp*
variables
$mem, store\_stage, ready\_1, mem\_1, ready\_2, mem\_2, ready\_3, mem\_3, store\_result, store\_failure$
invariant
$mem \in 0 \mathinner{..} 255 \mid store\_stage \in 0 \mathinner{..} 2 \parallel ready\_1 \in BOOL$
$mem\_1 \in 0 \mathinner{..} 255 \parallel ready\_2 \in BOOL \parallel mem\_2 \in 0 \mathinner{..} 255$
$ready\_3 \in BOOL \parallel mem\_3 \in 0 \mathinner{..} 255 \parallel store\_result \in \mathbb{N} \pfun (0 \mathinner{..} 255)$
$store\_failure \in BOOL$
initialisation
$mem :\in 0 \mathinner{..} 255 \parallel store\_stage := 0 \parallel ready\_1 := FALSE$
$mem\_1 :\in 0 \mathinner{..} 255 \parallel ready\_2 := FALSE \parallel mem\_2 :\in 0 \mathinner{..} 255$
$ready\_3 := FALSE \parallel mem\_3 :\in 0 \mathinner{..} 255 \parallel store\_result := \{\}$
$store\_failure := FALSE$
events
$reset$ = **begin**
$mem := 0$
end

alt_1 = **when**
$(store\_stage = 0) \land (ready\_1 = FALSE)$
then
$mem\_1 :\in 0 \mathinner{..} 255$
$ready\_1 := TRUE$
end

alt_2 = **when**
$(store\_stage = 0) \land (ready\_2 = FALSE)$
then
$mem\_2 :\in 0 \mathinner{..} 255$
$ready\_2 := TRUE$
end

alt_3 = **when**
$(store\_stage = 0) \land (ready\_3 = FALSE)$
then
$mem\_3 :\in 0 \mathinner{..} 255$
$ready\_3 := TRUE$
end

$store\_collect$ = **when**
$(store\_stage = 0) \land (ready\_1 = TRUE \land ready\_2 = TRUE \land ready\_3 = TRUE)$
then
$store\_result := (\{1 \mapsto (mem\_1)\}) \cup (\{2 \mapsto (mem\_2)\}) \cup (\{3 \mapsto (mem\_3)\})$
$store\_stage := 1$
end

$store$ = **any** $store\_k, mem\_t$ **where**
$(store\_stage = 1) \land$
$(\forall j \cdot (j \in dom(store\_result) \land j \neq store\_k \Rightarrow$
$card(store\_result^{-1}[\{store\_result(store\_k)\}]) \geq$
$card(store\_result^{-1}[\{store\_result(j)\}]))) \land$
$((mem\_t) = store\_result(store\_k)) \land store\_k \in dom(store\_result) \land mem\_t \in 0 \mathinner{..} 255$
then
$mem := mem\_t$
$store\_stage := 2$
$ready\_1 := FALSE$
$ready\_2 := FALSE$
$ready\_3 := FALSE$
$store\_failure := bool(card(store\_result^{-1}[\{store\_result(store\_k)\}]) < 2)$
end

To hide the details introduced by the pattern, the model can be decomposed into three parts, each dealing with its own version.

We can also easily combine the two patterns by applying them one after another however the pretty-printed representation of such models is rather lengthy (note that users of the RODIN platforms can focus on the specific parts of a model without having to see the rest).