

A Save Area

Using Z
Jim Davies & Jim Woodcock

A save area

A save area is a module with two operations: save and restore.

Packets of information are stored in a last-in first-out manner: in this respect, the module behaves as a stack.

Information

The structure of packets of information does not concern us at this level of abstraction:

[Record]

State

<i>SaveArea</i>	_____
<i>save_area</i>	: seq <i>Record</i>

<i>InitSaveArea</i>	_____
<i>SaveArea'</i>	
<i>save_area'</i>	= ⟨⟩

<i>Save₀</i>
$\Delta SaveArea$
$record? : Record$
$status! : Status$
$save_area' = save_area \hat{\ } \langle record? \rangle$
$status! = ok$

<i>SaveFullErr</i>
$\exists SaveArea$
$status! : Status$
$status! = full$

$Save \hat{=} Save_0 \vee SaveFullErr$

Question

Should this be a total operation?

<i>Restore₀</i>
$\Delta SaveArea$
$r! : Record$
$status! : Status$
$save_area \neq \langle \rangle$
$save_area = save_area' \hat{\ } \langle r! \rangle$
$status! = ok$

<i>RestoreEmptyErr</i>
$\exists \text{SaveArea}$
<i>status!</i> : <i>Status</i>
<i>save_area</i> = $\langle \rangle$
<i>status!</i> = empty

$\text{Restore} \hat{=} \text{Restore}_0 \vee \text{RestoreEmptyErr}$

Design

Introduce two levels of memory: main and secondary.

Let n be the number of records that we can save in main memory:

$$\frac{n : \mathbb{N}}{n \geq 1}$$

Operation	Precondition
<i>Save</i> <i>Save</i> ₀	<i>true</i>
<i>SaveFullErr</i>	<i>true</i>
<i>Save</i>	<i>true</i>
<i>Restore</i> <i>Restore</i> ₀	<i>save_area</i> ≠ $\langle \rangle$
<i>RestoreEmptyErr</i>	<i>save_area</i> = $\langle \rangle$
<i>Restore</i>	<i>true</i>

Preconditions for the save area

Question

Save was defined nondeterministically; n has been defined loosely.

What is the difference?

$[X]$
 $bseq : \mathbb{P}(\text{seq } X)$
 $fseq : \mathbb{P}(\text{seq } X)$
 $bseq = \{ s : \text{seq } X \mid \#s \leq n \}$
 $fseq = \{ s : \text{seq } X \mid \#s = n \}$
 $C\text{SaveArea}$
 $main : bseq[\text{Record}]$
 $secondary : \text{seq}(fseq[\text{Record}])$
 Retrieve
 SaveArea
 $C\text{SaveArea}$
 $save_area = (\wedge / secondary) \wedge main$

Question

In the first specification, *save_area* is initialised to the empty sequence. Can you calculate a suitable initialisation for our new level of design?

 $C\text{SaveArea}'$

Question

In the first specification, the *Save* operation appended a record to the sequence *save_area*. What should the effect be now, in terms of *main* and *secondary*?

 $\Delta C\text{SaveArea}$
 $record? : \text{Record}$
 $status! : \text{Status}$

Question

Can you use the information from the state invariant to simplify your answer?

Retrieve1 _____

CSaveArea

CSaveArea1

$main = (1 .. count) \triangleleft array$

Further design

The main memory storage will be implemented using an array and a counter:

CSaveArea1 _____

$array : Array[Record]$

$count : 0 .. n$

$secondary : seq(fseq[Record])$

where

$[X]$ _____

$Array : \mathbb{P}(\mathbb{N} \rightarrow X)$

$Array = (1 .. n) \rightarrow X$

Question

What should *Save₀* do now?

$\Delta CSaveArea1$

$record? : Record$

$status! : Status$

$CCSaveFullErr$ $\exists CSaveArea1$ $status! : Status$
$status! = full$

$$CCSave \hat{=} CCSave_0 \vee CCSaveFullErr$$

Refinement to code

We break the $CCSave_0$ operation into two disjuncts:

- $CCUpdateMM$: an operation that updates the main memory
- $CCUpdateSM$: an operation that updates the secondary memory

$CCUpdateMM$ $\Delta CSaveArea1$ $record? : Record$ $status! : Status$
$count < n$ $count' = count + 1$ $array' = array \oplus \{count + 1 \mapsto record?\}$ $secondary' = secondary$ $status! = ok$

$CCUpdateSM$ $\Delta CSaveArea1$ $record? : Record$ $status! : Status$
$count = n$ $count' = 1$ $array' 1 = record?$ $secondary' = secondary \hat{\setminus} \langle array \rangle$ $status! = ok$

The save operation

$$\text{save} \hat{=} \text{CSaveArea1}, \text{status!} : [\text{true}, \text{CCSave}]$$

Refinement

$$\text{save} = \text{CSaveArea1}, \text{status!} : \left[\begin{array}{c} \text{CCUpdateMM} \\ \vee \\ \text{CCUpdateSM} \\ \vee \\ \text{true}, \text{CCSaveFullErr} \end{array} \right]$$

$$\begin{array}{l} \text{if } \text{count} < n \rightarrow \\ \quad \text{CSaveArea1}, \text{status!} : [\text{count} < n, \text{CCUpdateMM}] \quad [\triangleleft] \\ \square \text{count} = n \rightarrow \\ \quad \text{CSaveArea1}, \text{status!} : \left[\begin{array}{c} \text{CCUpdateSM} \\ \vee \\ \text{count} = n, \text{CCSaveFullEr} \end{array} \right] \quad [\dagger] \\ \text{fi} \end{array}$$

\triangleleft

$$\text{count}, \text{array}, \text{status!} : \left[\begin{array}{c} \text{count}' = \\ \quad \text{count} + 1 \\ \text{array}' = \\ \quad \text{array} \oplus \{ \text{count} + 1 \mapsto \text{record?} \} \\ \text{status!} = \\ \text{count} < n, \quad \text{ok} \end{array} \right]$$

$$count, array, status! := count + 1, array \oplus \{count + 1 \mapsto record?\}, ok$$

†

$$count, array, secondary, status! : \left[\begin{array}{l} count' = 1 \wedge \\ array' 1 = record? \wedge \\ secondary' = \\ secondary \hat{\ } \langle array \rangle \wedge \\ status! = ok \\ \vee \\ count' = count \wedge \\ array' = array \wedge \\ secondary' = secondary \wedge \\ count = n, \quad status! = full) \end{array} \right]$$

```

if count < n →
    count, array, status! := count + 1, array \oplus \{count + 1 \mapsto record?\}, ok
□ count = n →

status!, secondary : \left[ \begin{array}{l} status! = ok \wedge \\ secondary' = secondary \hat{\ } \langle array' \rangle \\ \vee \\ true, status! = full \wedge secondary' = secondary \end{array} \right]

;
if status! = ok →
    count, array := 1, array \oplus \{1 \mapsto record?\}
□ status! = full →
    skip
fi
fi

```