

A Save Area

Using Z
Woodcock & Davies

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

SaveArea

save_area : seq Record

InitSaveArea

SaveArea'

save_area' = ()

*Save*₀

$\Delta \text{SaveArea}$

record? : Record

status! : Status

save-area' = *save-area* \cap $\langle \text{record?} \rangle$

status! = ok

Question

Should this be a total operation?

SaveFullErr

$\exists \text{SaveArea}$

status! : *Status*

status! = full

Save $\hat{=}$ *Save*₀ \vee *SaveFullErr*

*Restore*₀

$\Delta \text{SaveArea}$

$r! : \text{Record}$

$status! : \text{Status}$

$\text{save_area} \neq \langle \rangle$

$\text{save_area} = \text{save_area}' \cap \langle r! \rangle$

$status! = \text{ok}$

RestoreEmptyErr

$\exists \text{SaveArea}$

$\text{status!} : \text{Status}$

$\text{save_area} = \langle \rangle$

$\text{status!} = \text{empty}$

$\text{Restore} \triangleq \text{Restore}_0 \vee \text{RestoreEmptyErr}$

Operation	Precondition
<i>Save</i>	<i>Save</i> ₀
	<i>SaveFullErr</i>
	<i>Save</i>
<i>Restore</i>	<i>Restore</i> ₀
	<i>RestoreEmptyErr</i>
	<i>Restore</i>

Preconditions for the save area

Design

Introduce two levels of memory: main and secondary.

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

$$\begin{array}{l} n : \mathbb{N} \\ \hline n \geq 1 \end{array}$$

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 = { $s : \text{seq } X \mid \#s = n$ }

CSaveArea

main : bseq[Record]

secondary : seq(fseq[Record])

Retrieve

SaveArea

CSaveArea

save_area = (∩ / secondary) ∩ 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?

CSaveArea'

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*?

$\Delta CSaveArea$

record? : Record

status! : Status

Question

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

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

Retrieve1

CSaveArea

CSaveArea1

main = (1 .. count) ⊲ array

Question

What should $Save_0$ do now?

$\Delta CSaveArea1$

$record? : Record$

$status! : Status$

$CCSaveFullErr$ $\exists CCSaveArea1$ $status! : Status$ $status! = \text{full}$ $CCSave \triangleq CCSave_0 \vee CCSaveFullErr$

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 \cap {array}

status! = ok

The *save* operation

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

Refinement

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

if $count < n \rightarrow$

$CSaveArea1, status! : [count < n, CCUpdateMM]$ [▷]

□ $count = n \rightarrow$

$CSaveArea1, status! : \left[\begin{array}{c} CCUpdateSM \\ \vee \\ count = n , CCSaveFullEr \end{array} \right]$ [†]

fi

▷

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

count, array, status! := count + 1, array ⊕ {count + 1 ↦ record?}, ok

†

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

```
if  count < n →  
    count, array, status! := count + 1, array ⊕ {count + 1 ↦ record?}, ok  
□  count = n →  
    status!, secondary : [status! = ok ∧  
                           secondary' = secondary ∩ ⟨array'⟩]  
                           ∨  
                           true , status! = full ∧ secondary' = secondary ]  
;  
if  status! = ok →  
    count, array := 1, array ⊕ {1 ↦ record?}  
□  status! = full →  
    skip  
fi  
fi
```

