A File System

A programming interface

We will model the programming interface to a file system. This is a list of operations upon the file system, complete with a description of their intended effects.

We will divide the operations into two groups: those that affect the data within a single file, and those that affect the file system as a whole.

File operations

- `read`: used to read a piece of data from a file
- `write`: used to write a piece of data to a file
- `add`: used to add a new piece of data to a file
- `delete`: used to delete a piece of data from a file

File system operations

- `create`: used to create a new file
- `destroy`: used to destroy an existing file
- `open`: used to make a file available for the reading and writing of data
- `close`: used to make a file unavailable for reading and writing

Files (first half of the operations)

$[Key, Data]$

File

$contents : Key \rightarrow\!\!\!\!\!+ Data$

Initialisation

FileInit

$File'$

$contents' = \{\}$

read

$Read_0$

$\Xi File$
$k? : Key$
$d! : Data$

$k? \in \operatorname{dom} contents$
$d! = contents\ k?$

write

$Write_0$

$\Delta File$
$k? : Key$
$d? : Data$

$k? \in \operatorname{dom} contents$
$contents' = contents \oplus \{k? \mapsto d?\}$

add

Add_0

$\Delta File$
$k? : Key$
$d? : Data$

---

delete

$Delete_0$

Key errors

$Report ::= \mathsf{keyInUse} \mid \mathsf{keyNotInUse} \mid \mathsf{okay}$

$KeyError$

$\Xi File$
$k? : Key$
$r! : Report$

A failed operation upon the file state will always produce a report as output.

$KeyNotInUse$

$KeyError$

$k? \notin \operatorname{dom} contents$
$r! = \mathsf{keyNotInUse}$

Error because the specified key is not in use.

$KeyInUse$

$KeyError$

$k? \in \operatorname{dom} contents$
$r! = \mathsf{keyInUse}$

Error because the specified key is in use.

Success

$$
\begin{array}{l}
\textit{Success} \\
\hline
r! : \mathit{Report} \\
\hline
r! = \mathsf{okay} \\
\hline
\end{array}
$$

Successful operation will always produce a report of the same value.

A collection of total operations: schemas in which the state before may be any valid file.

$$\mathit{Read} \mathrel{\widehat{=}} (\mathit{Read}_0 \land \mathit{Success}) \lor \mathit{KeyNotInUse}$$

$$\mathit{Write} \mathrel{\widehat{=}} (\mathit{Write}_0 \land \mathit{Success}) \lor \mathit{KeyNotInUse}$$

$$\mathit{Add} \mathrel{\widehat{=}} (\mathit{Add}_0 \land \mathit{Success}) \lor \mathit{KeyInUse}$$

$$\mathit{Delete} \mathrel{\widehat{=}} (\mathit{Delete}_0 \land \mathit{Success}) \lor \mathit{KeyNotInUse}$$

Otherwise

(could prove to be an overwhelming task if not partitioned into smaller pieces)

$$
\begin{array}{l}
\mathit{contents}, \mathit{contents}' : \mathit{Key} \rightarrowtail \mathit{Data} \\
k? : \mathit{Key} \\
d! : \mathit{Data} \\
r! : \mathit{Report} \\
\hline
(\ k? \in \operatorname{dom} \mathit{contents} \land \\
\quad d! = \mathit{contents}\ k? \land \\
\quad \mathit{contents}' = \mathit{contents} \land \\
\quad r! = \mathsf{okay}\ ) \\
\lor \\
(\ k? \notin \operatorname{dom} \mathit{contents} \land \\
\quad \mathit{contents}' = \mathit{contents} \land \\
\quad r! = \mathsf{keyNotInUse}\ ) \\
\hline
\end{array}
$$

File system

(second half of the operations)

$[\mathit{Name}]$ It's important that the system should not associate the same name with two different files: file must always be functional.

$$
\begin{array}{l}
\textit{System} \\
\hline
\mathit{file} : \mathit{Name} \rightarrowtail \mathit{File} \\
\mathit{open} : \mathbb{P}\ \mathit{Name} \\
\hline
\mathit{open} \subseteq \operatorname{dom} \mathit{file} \\
\hline
\end{array}
$$

open is a set of names of those files currently open

Initialisation

$$
\begin{array}{l}
\textit{SystemInit} \\
\hline
\mathit{System}' \\
\hline
\mathit{file}' = \varnothing \\
\hline
\end{array}
$$

When the file system is initialized, there are no files. The partial function 'file' is empty, as is the set 'open.'

As the state invariant insists that every open file is also recorded in file, it is enough to insist that file = ø.

Promotion

Since the state of the file system includes indexed copies of File, we may promote the operations defined above. The local state is described by File, the global state is described by System...

$$
\begin{array}{l}
\textit{Promote} \\
\hline
\Delta \mathit{System} \\
\Delta \mathit{File} \\
n? : \mathit{Name} \\
\hline
n? \in \mathit{open} \\
\mathit{file}\ n? = \theta \mathit{File} \\
\mathit{file}' = \mathit{file} \oplus \{ n? \mapsto \theta \mathit{File}' \} \\
\mathit{open}' = \mathit{open} \\
\hline
\end{array}
$$

File operations

Based on the promotion, we may define the following four opeations...

$$\mathit{KeyRead}_0 \mathrel{\widehat{=}} \exists\, \Delta \mathit{File} \bullet \mathit{Read} \land \mathit{Promote}$$

$$\mathit{KeyWrite}_0 \mathrel{\widehat{=}} \exists\, \Delta \mathit{File} \bullet \mathit{Write} \land \mathit{Promote}$$

$$\mathit{KeyAdd}_0 \mathrel{\widehat{=}} \exists\, \Delta \mathit{File} \bullet \mathit{Add} \land \mathit{Promote}$$

$$\mathit{KeyDelete}_0 \mathrel{\widehat{=}} \exists\, \Delta \mathit{File} \bullet \mathit{Delete} \land \mathit{Promote}$$

Although each local operation is total, the file in question may not be open. The resulting global operations are partial.

File access

$FileAccess$
$\Delta System$
$n? : Name$
$n? \in \operatorname{dom} file$
$file' = file$

Open and close change the availability of a file for reading and writing. The FileAccess operation leaves the file Function unchanged. The input component n? describes a file that is known to the system.

$Open_0$
$FileAccess$
$n? \notin open$
$open' = open \cup \{n?\}$

A successful open operation adds a name to the list of open files (strictly partial, and fails if the name supplied denotes a file that is already open)

Closing a file

$Close_0$
$FileAccess$
$n? \in open$
$open' = open \setminus \{n?\}$

A successful close operation removes a name from the list of open files, is strictly partial and will fail if the name supplied does not denote an open file.

File management

$FileManage$
$\Delta System$
$n? : Name$
$open' = open$

FileManage is used to describe the information that is common to both operations (create, destroy).

$Create_0$
$FileManage$
$\exists FileInit' \bullet$
$\quad n? \notin \operatorname{dom} file \wedge$
$\quad file' = file \cup \{n? \mapsto \theta File'\}$

Successful create operation adds a new name to the list of files know to the system.

Immediately after this operation, the state of the file associated with name n? is described by the binding to Theta-FileInit (instead of File'). I.e., n? is associated with a binding of schema type File in which contents is bound to empty set.

Destroying a file

$Destroy_0$

We may also want to insist that n? is not an element of open, thus preventing the destruction of open files. However, this condition is already enforced by the predicate part of FileMange (which insists that this operation should not affect the list of open files). Acting in combination with our state invariant 'open is a subset of dom file' Thus, if we cannot remove n? from open, then we cannot remove n? from the domain of file.

More reports

$Report ::=$ keyInUse | keyNotInUse | okay |
fileExists | fileDoesNotExist |
fileIsOpen | fileIsNotOpen

Free type of report messages is extended to take account of the errors that may ocur in file access ad file management operations.

File errors

$FileError$
$\Xi System$
$n? : Name$
$r! : Report$

Information common to each of the error cases.

$FileExists$
$FileError$
$n? \in \operatorname{dom} file$
$r! = \text{fileExists}$

If we attempt to create a file using a name that is arelady in use we will receive a report complaining that a file with that name exists.

File system operations

$$KeyRead \mathrel{\widehat{=}} KeyRead_0 \lor \boxed{FileIsNotOpen} \lor \boxed{FileDoesNotExist}$$

$$KeyWrite \mathrel{\widehat{=}} KeyWrite_0 \lor \boxed{FileIsNotOpen} \lor \boxed{FileDoesNotExist}$$

$$KeyAdd \mathrel{\widehat{=}} KeyAdd_0 \lor \boxed{FileIsNotOpen} \lor \boxed{FileDoesNotExist}$$

$$KeyDelete \mathrel{\widehat{=}} KeyDelete_0 \lor \boxed{FileIsNotOpen} \lor \boxed{FileDoesNotExist}$$

There are four operations involving the contents of files: KeyRead, KeyWrite, KeyAdd, and KeyDelete. In each case if the file exists and is open, then the effect of the operation is described by a promoted file operation.

The complete set of definitions for the access and management operations using a similar combination of error cases:

$$Open \mathrel{\widehat{=}} (Open_0 \land Success) \lor FileIsOpen \lor FileDoesNotExist$$

$$Close \mathrel{\widehat{=}} (Close_0 \land Success) \lor FileIsNotOpen \lor FileDoesNotExist$$

$$Create \mathrel{\widehat{=}} (Create_0 \land Success) \lor FileExists$$

$$Destroy \mathrel{\widehat{=}} (Destroy_0 \land Success) \lor FileDoesNotExist \lor FileIsOpen$$

Formal analysis

- consistency of requirements
- operation preconditions

Initialisation theorem

$$\exists\, System' \bullet SystemInit$$

Lets check that our state invariant contains no contradictions. We establish this by proving that initialization theorem. I.e., that there exists a binding of file and open which satisfies the constraint part of SystemInit.

Skip

Proof

$$\dfrac{\dfrac{\dfrac{}{\varnothing \in Name \nrightarrow File} \quad \dfrac{\dfrac{}{\varnothing \in \mathbb{P}\, Name} \quad \dfrac{}{\varnothing \subseteq \operatorname{dom} \varnothing}}{\exists\, open' : \mathbb{P}\, Name \bullet open' \subseteq \operatorname{dom} \varnothing}\,[\exists\text{–intro}]}{\exists\, file' : Name \nrightarrow File;\ open' : \mathbb{P}\, Name \mid open' \subseteq \operatorname{dom} file' \bullet file' = \varnothing}\,[\text{one-point}]}{\exists\, System' \bullet SystemInit}\,[\text{definition}]$$

Since Key and Data are basic types they cannot be empty. Hence the empy relation (PartialFunction: key --> data) and the initial state exists. The second part of the investigation involves calculating the precondition of each operation.

Precondition

The second part of the investigation involves calculating the precondition of each operation.

$$KeyRead \mathrel{\widehat{=}} KeyRead_0 \lor FileDoesNotExist \lor FileIsNotOpen$$

The 'pre' operator distributes through disjunction:

$$\operatorname{pre} KeyRead = \operatorname{pre} KeyRead_0 \lor \operatorname{pre} FileDoesNotExist \lor \operatorname{pre} FileIsNotOpen$$

pre *FileIsNotOpen*

$$
\begin{array}{l}
System \\
n? : Name \\
\hline
\exists\, r! : Report \bullet \\
\quad \boxed{\begin{array}{l} n? \notin open \land \\ n? \in \operatorname{dom} file \land \end{array}} \\
\quad r! = \mathsf{fileIsNotOpen}
\end{array}
$$

Precondition/constraint

pre *KeyRead*

$$KeyRead_0 \mathrel{\widehat{=}} \exists\, \Delta File \bullet Read \land Promote$$

$$\operatorname{pre} KeyRead_0 = \exists\, Local \bullet \operatorname{pre} Read \land \operatorname{pre} Promote$$

$$\operatorname{pre} KeyRead \Leftrightarrow true$$

Result

Operation		Precondition
KeyRead	$KeyRead_0$	$n? \in open$
	FileIsNotOpen	$n? \in (\operatorname{dom} file) \setminus open$
	FileDoesNotExist	$n? \notin \operatorname{dom} file$
	KeyRead	*true*