This is the BirthdayBook specification, from Spivey [1]. We extend it slightly by adding an extra operation, *RemindOne*, that is non-deterministic.

$$NAME == 1 \mathrel{..} 5$$

$$DATE == 10 \mathrel{..} 15$$

The *BirthdayBook* schema defines the *state space* of the birthday book system.

$$\begin{array}{|l}
\hline
\mathit{BirthdayBook} \\
\mathit{known} : \mathbb{P}\, \mathit{NAME} \\
\mathit{birthday} : \mathit{NAME} \pfun \mathit{DATE} \\
\hline
\mathit{known} = \mathrm{dom}\, \mathit{birthday} \\
\hline
\end{array}$$

This *InitBirthdayBook* specifies the initial state of the birthday book system. It does not say explicitly that *birthday′* is empty, but that is implicit, because its domain is empty.

$$\begin{array}{|l}
\hline
\mathit{InitBirthdayBook} \\
\mathit{BirthdayBook}\,' \\
\hline
\mathit{known}' = \{\} \\
\hline
\end{array}$$

Next we have several operation schemas to define the normal (non-error) behaviour of the system.

$$\begin{array}{|l}
\hline
\mathit{AddBirthday} \\
\Delta \mathit{BirthdayBook} \\
\mathit{name}? : \mathit{NAME} \\
\mathit{date}? : \mathit{DATE} \\
\hline
\mathit{name}? \notin \mathit{known} \\
\mathit{birthday}' = \mathit{birthday} \cup \{\mathit{name}? \mapsto \mathit{date}?\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\mathit{FindBirthday} \\
\Xi \mathit{BirthdayBook} \\
\mathit{name}? : \mathit{NAME} \\
\mathit{date}! : \mathit{DATE} \\
\hline
\mathit{name}? \in \mathit{known} \\
\mathit{date}! = \mathit{birthday}(\mathit{name}?) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{Remind} \\
\hline
\Xi \textit{BirthdayBook} \\
\textit{today}? : \textit{DATE} \\
\textit{cards}! : \mathbb{P}\, \textit{NAME} \\
\hline
\textit{cards}! = \{ n : \textit{known} \mid \textit{birthday}(n) = \textit{today}? \} \\
\hline
\end{array}$$

This *RemindOne* schema does not appear in Spivey, but is included to show how non-deterministic schemas can be animated. It reminds us of just one person who has a birthday on the given day.

$$\begin{array}{|l}
\textit{RemindOne} \\
\hline
\Xi \textit{BirthdayBook} \\
\textit{today}? : \textit{DATE} \\
\textit{card}! : \textit{NAME} \\
\hline
\textit{card}! \in \textit{known} \\
\textit{birthday}\ \textit{card}! = \textit{today}? \\
\hline
\end{array}$$

Now we strengthen the specification by adding error handling.

$$\textit{REPORT} ::= \textit{ok} \mid \textit{already\_known} \mid \textit{not\_known}$$

First we define auxiliary schemas that capture various success and error cases.

$$\begin{array}{|l}
\textit{Success} \\
\hline
\textit{result}! : \textit{REPORT} \\
\hline
\textit{result}! = \textit{ok} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{AlreadyKnown} \\
\hline
\textit{BirthdayBook} \\
\textit{BirthdayBook}\,' \\
\textit{name}? : \textit{NAME} \\
\textit{result}! : \textit{REPORT} \\
\hline
\textit{name}? \in \textit{known} \\
\textit{result}! = \textit{already\_known} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{NotKnown} \\
\hline
\Xi \textit{BirthdayBook} \\
\textit{name}? : \textit{NAME} \\
\textit{result}! : \textit{REPORT} \\
\hline
\textit{name}? \notin \textit{known} \\
\textit{result}! = \textit{not\_known} \\
\hline
\end{array}$$

Finally, we define robust versions of all the operations by specifying how errors are handled. For illustration purposes, we leave the *RemindOne* operation non-robust.

$$\mathit{RAddBirthday} == (\mathit{AddBirthday} \land \mathit{Success}) \lor \mathit{AlreadyKnown}$$
$$\mathit{RFindBirthday} == (\mathit{FindBirthday} \land \mathit{Success}) \lor \mathit{NotKnown}$$
$$\mathit{RRemind} == \mathit{Remind} \land \mathit{Success}$$

Finally, we can (optionally) define an explicit state machine, by identifying the state and initialisation schemas and the operations. This is not necessary for the animator, but is useful to make the roles of the various schemas explicit, rather than relying on informal naming conventions. This *machine* construct is a Java-specific extension of Z and will be ignored by other Z tools (because they do not recognise the `\begin{machine}..\end{machine}` LaTeXenvironment.

Jaza uses this construct for translating the machine into the B specification language or other similar languages and (in the future) for constructing GUI animation interface for the machine.

machine BirthdayBook
  BirthdayBook
init
  InitBirthdayBook
ops
  RAddBirthday; RFindBirthday; RRemind; RemindOne
end

Here is an alternative style of defining a machine – as a schema with fields named *state* and *init*, plus other fields for the operations of the machine. Advantages of this approach include:

1. This *machine* construct is a standard Z schema, so all the usual Z operators can be used to construct machines, add and hide operations etc.

2. The operation names are local to this schema rather than global, so short, non-qualified names can be used. Furthermore, several machines can have the same operation name.

3. Other (non-schema) fields inside the machine can represent global constants of the machine, and the invariant can be used to constrain these constants.

4. A generic schema can represent a parameterised machine.

Disadvantages:

1. The machine name cannot be the same as its state schema.

2. Some restrictions are probably necessary on the invariant of the machine, because it is difficult to see what should be the meaning of a predicate that constrains two of the operations. (This has no obvious translation into B. It is not expressive enough to act as a history invariant.)

BBook

$state : BirthdayBook$
$init : InitBirthdayBook$
$add : RAddBirthday$
$find : RFindBirthday$
$remind : RRemind$
$remind1 : RemindOne$

References

[1] J. Michael Spivey. *The Z Notation: A Reference Manual.* International Series in Computer Science. Prentice-Hall International (UK) Ltd, second edition, 1992.