```
\documentclass{article}
\usepackage{oz}   % oz or z-eves or fuzz styles

\begin{document}
This is the BirthdayBook specification, from
Spivey~\cite{spivey:z-notation2}.  We extend it slightly
by adding an extra operation, $RemindOne$, that is non-deterministic.

\begin{zed}
  [NAME, ~ DATE]
\end{zed}

The $BirthdayBook$ schema defines the \emph{state space} of
the birthday book system.

\begin{schema}{BirthdayBook}
   known: \power NAME \\
   birthday: NAME \pfun DATE
\where
   known=\dom birthday
\end{schema}

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{schema}{InitBirthdayBook}
   BirthdayBook~'
\where
   known' = \{ \}
\end{schema}

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

\begin{schema}{AddBirthday}
   \Delta BirthdayBook \\
   name?: NAME \\
   date?: DATE
\where
   name? \notin known
\\
   birthday' = birthday \cup \{name? \mapsto date?\}
\end{schema}

\begin{schema}{FindBirthday}
   \Xi BirthdayBook \\
   name?: NAME \\
   date!: DATE
\where
   name? \in known
\\
```

```
    date! = birthday(name?)
\end{schema}

\begin{schema}{Remind}
    \Xi BirthdayBook \\
    today?: DATE \\
    cards!: \power NAME
\where
    cards! = \{ n: known | birthday(n) = today? \}
\end{schema}
```

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{schema}{RemindOne}
    \Xi BirthdayBook \\
    today?: DATE \\
    card!: NAME
\where
    card! \in known \\
    birthday ~ card! = today?
\end{schema}
```

Now we strengthen the specification by adding error handling.

```
\begin{zed}
    REPORT ::= ok | already\_known | not\_known
\end{zed}
```

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

```
\begin{schema}{Success}
    result!: REPORT
\where
    result! = ok
\end{schema}

\begin{schema}{AlreadyKnown}
    \Xi BirthdayBook \\
    name?: NAME \\
    result!: REPORT
\where
    name? \in known \\
    result! = already\_known
\end{schema}



\begin{schema}{NotKnown}
    \Xi BirthdayBook \\
    name?: NAME \\
```

```
    result!: REPORT
\where
    name? \notin known \\
    result! = not\_known
\end{schema}

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.

\begin{zed}
    RAddBirthday == (AddBirthday \land Success) \lor AlreadyKnown \\
    RFindBirthday == (FindBirthday \land Success) \lor NotKnown \\
    RRemind == Remind \land Success
\end{zed}

\bibliography{spec}
\begin{thebibliography}{1}
\bibitem{spivey:z-notation2}
J.~Michael Spivey.
\newblock {\em The Z Notation: A Reference Manual}.
\newblock International Series in Computer Science. Prentice-Hall International
  (UK) Ltd, second edition, 1992.
\end{thebibliography}
\end{document}
```