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.

⌐[NAME, DATE]

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

┌BirthdayBook known : ℙ NAME
birthday : NAME ⇸ DATE
|
known = dom birthday

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.

┌InitBirthdayBook BirthdayBook ’
|
known’ = {}

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

┌AddBirthday ΔBirthdayBook
name? : NAME
date? : DATE
|
name? ∉ known
birthday’ = birthday ∪ {name? ↦ date?}

┌FindBirthday Ξ BirthdayBook
name? : NAME
date! : DATE
|
name? ∈ known
date! = birthday(name?)

┌Remind Ξ BirthdayBook
today? : DATE
cards! : ℙ NAME
|
cards! = {n : known | birthday(n) = today?}

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.

┌RemindOne Ξ BirthdayBook
today? : DATE

card! : NAME
|
card! ∈ known
birthday card! = today?

Now we strengthen the specification by adding error handling.

REPORT ::= ok | already_known | not_known

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

Success result! : REPORT
|
result! = ok

AlreadyKnown ≙ BirthdayBook
name? : NAME
result! : REPORT
|
name? ∈ known
result! = already_known

NotKnown ≙ BirthdayBook
name? : NAME
result! : REPORT
|
name? ∉ known
result! = not_known

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.

RAddBirthday == (AddBirthday ∧ Success) ∨ AlreadyKnown
RFindBirthday == (FindBirthday ∧ Success) ∨ NotKnown
RRemind == Remind ∧ Success