The operation schema above does not complete the specification of the AddEntry operation. As we have already observer with algebraic specification, it is of vital importance to anticipate possible error/exception conditions. A specification should both clearly identify these conditions, and describe the intended behavior in these circumstances. In Z we can “add on” this part after the “normal case” specification quite easily.

The next several schemas are auxiliary schemas that enable us to complete the specification of the AddEntry operation in all the various circumstances. The following is also an operation schema, but it promises *not* to change the state.

NotMember
ΞPhoneDB
name?: Person
rep!: Report
---
name? ∉ members
rep! = 'Not a member'

The importing convention for the declaration ΞPhoneDB is the same as for Δ schemas, but promises to leave the state *unchanged* — we imagine *implicit* post-conditions X' = X for each state variable.

The decoration '!' is attached to the rep variable to denote that it is an **output**, or **result**, variable for this operation. With the “define before use” convention of Z, a definition of the type Report should have appeared before we see it here. Z includes “enumeration types”, and this is the intention for Report. However, until we complete the specification, we are unsure of all the messages we may wish to issue from the implementation. Therefore we defer presenting this until we know what we require. It will eventually be

Report ::= 'Not a member' | 'Okay' | …

We have two more pieces to provide before we can complete the specification of the first operation.

EnrtyAlreadyExists
ΞPhoneDB
name?: Person
newnumber?: Phone
rep!: Report
---
name? ↦ newnumber? ∈ telephones
rep! = 'Entry already exists'

This is another operation that does not change the state. It has two arguments and one result. There are two explicit conditions, one implicit invariant from PhoneDB, and two implicit post-conditions.

We have one last auxiliary schema

Success
rep!: Report
---
rep! = 'Okay'

This is a particularly simple scheme — it involves no state, has no argument, and produces a constant output.

This is the last of the constituents needed for our completed AddEntry operation. This provides an example of definition by **schema-formula**, namely

DoAddEntry ≙ AddEntry □ Success
NotMember
EntryAlreadyExists