Phone Database — Finale

We have examined Z facilities involved in the specification of all of the telephone database operations. However, these specifications describe only the effect of each individual operation, not a system in which these operations may be repeatedly performed. As a last step in this example, we examine Diller's specification of a rudimentary control system.

The first constituent of the control system is a new basic type, [Commands]. Commands can be viewed as an enumerated type

Commands ::= ae | fp | fn | re | am | rm

providing pneumonics for the six basic operations add entry, find phones, find names, remove entry, add member, and remove member.

The schemas describing the control facility are as shown below. First, the operation whose pre-condition is the test for an add entry command.

AddEntry Command

cmd?: Command

---

cmd? = ae

Then there is a schema about carrying out a command once it is identified.

CODoAddEntry ≙ DoAddEntryCommand ⨾ DoAddEntry

There are similar pairs of schema for each of the other operations. Also, there is a schema to cover the exceptional cases.

UnknownCommand

ΞPhoneDB

cmd?: Command

rep!: Report

---

cmd? ∉ {ae,fp, fn, re, am, rn}

rep! = 'Unknown command'

Then the overall control is specified as

PhoneDatabase = CODoAddMember
CODoRemoveMember
CODoAddEntry
CODoRemoveEntry
CODoFindPhones
CODoFindNames
Unknown Command

There is one other schema to be mentioned, one specifying the initial state.

InitPhoneDB
PhoneDB'

members' = ∅
telephones' = ∅

Actually, the initial state for all the rest of the operations is the post-state of this operation. This state can be denoted as InitPhoneDB'.

The Miranda animation includes a realization of these specifications, but goes beyond them. It creates a continuing realization that accepts a sequence of commands. Before we examine this, we should pause to look at the general I/O utility functions.

```
>       before x = takewhile (~=x)
>       after x = tl . dropwhile (~=x)

>       read1 msg g input = msg ++ line ++ "\n" ++ g line input'
>                           where line = before '\n' input
>                                 input' = after '\n' input

>       read2 (msg1,msg2) g = read1 msg1 g1
>                             where
>                             g1 line1 = read1 msg2 g2
>                                        where g2 line2 = g (line1,line2)

>       write msg g input = msg ++ g input
>       end input = ""
```

Using these utilities, the Miranda animation code is

```
>       string == [char]
>       person == string
>       phone  == string          || extraction from input requires string
>       phonedb == ([person], [(person,phone)])

>       go :: string -> string
>       go = phdb empty

>       phdb :: phonedb -> string -> string
>       phdb db = tndb db, if invar db
>                     = write "Invariant violated\n" (tndb db), otherwise

>       invar :: phonedb -> bool
>       invar (mem, tel) = and [member mem n | (n,a) <- tel]

>       empty :: phonedb
>       empty = ([],[])
```

```
>       tndb :: phonedb -> string -> string

>       tndb (mem, tel)
>       = read1 "Command: " cocmd
>         where            || corrections in all cases of cocmd except first
>         cocmd "end" = write "Exit program\n" end
>         cocmd "ae" = read2 ("Name? ", "Extension? ") (doAddEntry (mem, tel))
>         cocmd "fp" = read1 "Name? " (doFindPhones (mem, tel))
>         cocmd "fn" = read1 "Extension? " (doFindNames (mem, tel))
>         cocmd "re" = read2 ("Name? ","Extension? ")
>                            (doRemoveEntry (mem, tel))
>         cocmd "am" = read1 "Name? " (doAddMember (mem, tel))
>         cocmd "rm" = read1 "Name? " (doRemoveMember (mem, tel))
>         cocmd other = write "Unknown command\n" (donothing (mem, tel))
```