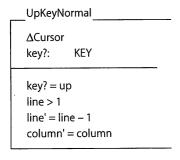
## Formal specification using Z

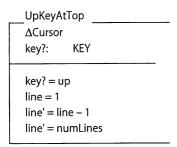
Cursor lines!: N

| lines! = numLines – line | line' = line | column | column

**2.** *UpKey*. This schema deals with what happens when the cursor is not on the top line of the display:



The next schema deals with what happens when the cursor is on the top line of the display:



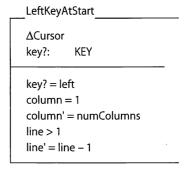
Note that the cursor has been defined to *wrap* round to the bottom line of the display. The full behaviour is given by:

UpKey == UpKeyNormal v UpKeyAtTop

**3.** *LeftKey.* The operation for moving left is given. It is easiest to deal first with what happens when the cursor is not at the far left of the display:

LeftKeyNormal	
ΔCursor key?: KEY	
key? = left column > 1 column' = column – 1 line' = line	

The next schema deals with the cursor's being at the left of a line other than the top line of the display. Note that the cursor wraps round to the start of the previous line:



Finally, a separate schema deals with the cursor being at the left of the top line. The cursor wraps round to the right of the bottom line:

```
LeftKeyAtTop

ΔCursor
key?: KEY

key? = left
column = 1
column' = numColumns
line = 1
line' = numLines
```

These schemas can be combined to form one schema which defines the response of the cursor to a left-move key in all initial positions of the cursor.

LeftKey == LeftKeyNormal ∨ LeftKeyAtStart ∨ LeftKeyAtTop

### **4.** NewDownKey

NewDownKeyAtBottom
ΔCursor
key?: KEY

key? = down
line = numLines
line' = numLines
column' = column

## **5.** NewRightKey

NewRightKeyAtRight
ΔCursor
key?: KEY

key? = right
column = numColumns
column' = numColumns
line' = line

NewRightKey == RightKeyNormal V NewRightKeyAtRight

#### 6.

(a) Yes, the Prime Minister must be a Member of Parliament because he or she is a member of the Cabinet and the Cabinet is a subset of the Members of Parliament.

(b)

HP

MPs: PPERSON
Cabinet: PPERSON
DPM, PM: PERSON

Cabinet ⊆ MPs
PM ∈ Cabinet
DPM ∈ Cabinet
DPM ≠ PM

- (c) The new Prime Minister may not be the same person as the old Prime Minister.
- (d) The new Prime Minister does not have to be chosen from the Cabinet.
- (e) The outgoing Prime Minister does not have to leave the Cabinet.
- (f) The outgoing Prime Minister may not leave the Cabinet.

#### 7.

- (a) The members of the new Cabinet must all be MPs, to maintain the invariant that the Cabinet is a subset of the MPs.
- (b) *ChangeCabinet2* requires a complete change of personnel in the new Cabinet.
- (c) The error is that the PM is unchanged and is always a member of the Cabinet, so the Cabinet cannot change completely.

# Chapter 7

1. [PERSON] the set of all uniquely identifiable persons

Computer	
users, loggedIn:	PPERSON
loggedIn <u></u> users	
InitComputer	
Computer'	
$\begin{array}{c} \operatorname{loggedIn'} = \varnothing \\ \operatorname{users'} = \varnothing \end{array}$	

RESPONSE ::=
OK | AlreadyAUser | NotAUser | Loggedin | NotLoggedin

#### **2.** Add user

AddUs	er <sub>0</sub>
ΔComp p?:	outer PERSON
	sers = users ∪ {p?} In' = loggedin

## Formal specification using Z

```
AddUserError

EComputer
p?: PERSON
reply!: RESPONSE

p? ∈ users
reply! = AlreadyAUser
```

AddUser ==  $(AddUser_0 \land [reply!: RESPONSE | reply! = OK]) \lor AddUserError$ 

3.

```
RemoveUser<sub>0</sub>
\Delta Computer
p?: PERSON
p? \in users
p \notin loggedIn
users' = users \setminus \{p?\}
loggedIn' = loggedIn
```

$$\label{eq:RemoveUser} \begin{split} & \mathsf{RemoveUser} == \\ & (\mathsf{RemoveUser}_0 \land [\mathsf{reply!} : \mathsf{RESPONSE} \mid \mathsf{reply!} = \mathsf{OK}]) \lor \\ & \mathsf{RemoveUserError} \end{split}$$

```
4. Log in
```

```
Login<sub>0</sub>
\Delta Computer
p?: PERSON

p? \in users
p? \notin loggedIn
loggedIn' = loggedIn \cup \{p?\}
users' = users
```

```
LoginError

EComputer
p?: PERSON
reply!: RESPONSE

(p? ∉ users ∧
reply! = NotAUser)
∨
(p? ∈ users ∧ p? ∈ loggedIn ∧
reply! = LoggedIn)
```

 $(\text{Login}_0 \land [\text{reply!: RESPONSE} \mid \text{reply!} = \text{OK}]) \lor \text{LoginError}$ 

**5.** Log out

```
Logout<sub>0</sub>
\Delta Computer
p?: PERSON
p? \in users
p? \in loggedIn
loggedIn' = loggedIn \setminus \{p?\}
users' = users
```

EComputer
p?: PERSON
reply!: RESPONSE

(p? ∉ users ∧
reply! = NotAUser)
∨
(p? ∈ users ∧
p? ∉ loggedIn ∧
reply = NotLoggedIn)

 $\label{eq:logout} \begin{tabular}{ll} Logout_0 \land [reply!: RESPONSE \ | \ reply! = OK]) \lor \\ LogoutError \end{tabular}$ 

# Chapter 8

- 1. loggedIn ⊆ users
- **2.** ∀i: **Z**•i\*i≥0
- **3.** ∃n: Z · n \* n = n
- 4.  $\{n: \mathbb{N} \mid (\forall m: \mathbb{N} \mid m \neq 1 \land m \neq n \cdot n \mod m \neq 0) \cdot n\}$

# Chapter 9

- Latin: LANGUAGE
   Latin ∉ ran speaks
- 2. # speaks (Switzerland))= 4
- EU: PCOUNTRY speaksInEU: COUNTRY ↔ LANGUAGE speaksInEU = EU ⊲ speaks
- **4.**grandParent: PERSON ↔ PERSON grandParent = parent; parent

- **5.** firstCousin: PERSON ↔ PERSON firstCousin = (grandParent ; grandParent ^) \ sibling
- **6.** Students are either from EU or overseas, but not both. Students study and teachers teach. Only offered modules can be studied. Those modules that are taught are studied.
- **7.** studies ({p})
- **8.** #(teaches({p}))
- **9.** Inverse of *studies* relates modules to persons studying them.
- **10.** The composition relates students to the teachers who teach modules the students study.
- 11. (studies ; teaches~)({p})
- **12.** #((studies; teaches~)( $\{p\}$ )  $\cap$  (studies; teaches~)( $\{q\}$ ))
- **13.** inter ⊲ studies
- 14.  $((\text{teaches}; \text{studies} \sim))(\{p\}) \cap (\text{teaches}; \\ \text{studies} \sim))(\{q\})) \rhd \text{inter} \neq \emptyset$
- 15.
  - (a) delegates ⊆ dom speaks
  - (b)  $ran \ speaks \cap official \neq \varnothing$
  - (c)
  - ∃ lang: LANGUAGE (∀del: PERSON | del ∈ delegates del speaks lang)
  - (d)

∃ del: PERSON • del ∈ delegates •

(∃ lang: LANGUAGE • del speaks lang ∧

(∀otherDel: PERSON | otherDel ∈ delegates \ {del} •

¬(otherDel speaks lang)))