or

```
LinesRemaining
ΔCursor
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:

```
UpKeyNormal
ΔCursor
key?:    KEY
--------------------
key? = up
line > 1
line' = line – 1
column' = column
```

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

```
UpKeyAtTop
ΔCursor
key?:    KEY
--------------------
key? = up
line = 1
line' = line – 1
line' = numLines
```

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 ∨ 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:

```
LeftKeyAtStart
ΔCursor
key?:    KEY
--------------------
key? = left
column = 1
column' = numColumns
line > 1
line' = line – 1
```

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
```

NewDownKey ≙ DownKeyNormal ∨ NewDownKeyAtBottom

5. *NewRightKey*

```
NewRightKeyAtRight
  ΔCursor
  key?:    KEY
  ----------------
  key? = right
  column = numColumns
  column' = numColumns
  line' = line
```

NewRightKey == RightKeyNormal ∨ 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 ⊆ users
```

```
InitComputer
  Computer'
  ----------------
  loggedIn' = ∅
  users' = ∅
```

RESPONSE ::=
OK | AlreadyAUser | NotAUser | LoggedIn | NotLoggedIn

2. Add user

```
AddUser0
  ΔComputer
  p?:    PERSON
  ----------------
  p? ∉ users
  users' = users ∪ {p?}
  loggedIn' = loggedIn
```

```
__AddUserError______________
| ΞComputer
| p?:       PERSON
| reply!:   RESPONSE
|_____________________
| p? ∈ users
| reply! = AlreadyAUser
|______________________
```

AddUser ==
(AddUser$_0$ ∧ [reply!: RESPONSE | reply! = OK]) ∨
AddUserError

3.

```
__RemoveUser0______________
| ΔComputer
| p?:       PERSON
|_____________________
| p? ∈ users
| p ∉ loggedIn
| users' = users \ {p?}
| loggedIn' = loggedIn
|______________________
```

```
__RemoveUserError__________
| ΞComputer
| p?:       PERSON
| reply!:   RESPONSE
|_____________________
| (p? ∉ users ∧
| reply! = NotAUser)
| ∨
| (p? ∈ users ∧
| p? ∈ loggedIn ∧
| reply! = LoggedIn)
|______________________
```

RemoveUser ==
(RemoveUser$_0$ ∧ [reply!: RESPONSE | reply! = OK]) ∨
RemoveUserError

4. Log in

```
__Login0___________________
| ΔComputer
| p?:       PERSON
|_____________________
| p? ∈ users
| p? ∉ loggedIn
| loggedIn' = loggedIn ∪ {p?}
| users' = users
|______________________
```

```
__LoginError_______________
| ΞComputer
| p?:       PERSON
| reply!:   RESPONSE
|_____________________
| (p? ∉ users ∧
| reply! = NotAUser)
| ∨
| (p? ∈ users ∧ p? ∈ loggedIn ∧
| reply! = LoggedIn)
|______________________
```

Login ==
(Login$_0$ ∧ [reply!: RESPONSE | reply! = OK]) ∨ LoginError

5. Log out

```
__Logout0__________________
| ΔComputer
| p?:       PERSON
|_____________________
| p? ∈ users
| p? ∈ loggedIn
| loggedIn' = loggedIn \ {p?}
| users' = users
|______________________
```

```
LogoutError
  ΞComputer
  p?:      PERSON
  reply!:  RESPONSE
  ──────────────────
  (p? ∉ users ∧
  reply! = NotAUser)
  ∨
  (p? ∈ users ∧
  p? ∉ loggedIn ∧
  reply = NotLoggedIn)
```

Logout ==
$(Logout_0 \wedge [reply!: RESPONSE \mid reply! = OK]) \vee$
LogoutError

Chapter 8

1.

$loggedIn \subseteq users$

2.

$\forall i: \mathbb{Z} \bullet i * i \geq 0$

3.

$\exists n: \mathbb{Z} \bullet n * n = n$

4.

$\{n: \mathbb{N} \mid (\forall m: \mathbb{N} \mid m \neq 1 \wedge m \neq n \bullet n \bmod m \neq 0) \bullet n\}$

Chapter 9

1.

Latin: LANGUAGE
$Latin \notin \mathrm{ran}\ speaks$

2.

$\#\ speaks(\{Switzerland\}) = 4$

3.

EU: $\mathbb{P}COUNTRY$
speaksInEU: COUNTRY $\leftrightarrow$ LANGUAGE
$speaksInEU = EU \lhd speaks$

4.

grandParent: PERSON $\leftrightarrow$ PERSON
$grandParent = parent\ ;\ parent$

5.

firstCousin: PERSON $\leftrightarrow$ PERSON
$firstCousin = (grandParent\ ;\ grandParent^{\sim}) \setminus sibling$

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.

$(studies\ ;\ teaches^{\sim})(\{p\})$

12.

$\#((studies\ ;\ teaches^{\sim})(\{p\}) \cap (studies\ ;\ teaches^{\sim})(\{q\}))$

13.

$inter \lhd studies$

14.

$((teaches\ ;\ studies^{\sim})(\{p\}) \cap (teaches\ ;\ studies^{\sim})(\{q\})) \rhd inter \neq \varnothing$

15.

(a)

$delegates \subseteq \mathrm{dom}\ speaks$

(b)

$\mathrm{ran}\ speaks \cap official \neq \varnothing$

(c)

$\exists\ lang: LANGUAGE \bullet$
$(\forall del: PERSON \mid del \in delegates \bullet del\ speaks\ lang)$

(d)

$\exists\ del: PERSON \bullet del \in delegates \bullet$
$(\exists\ lang: LANGUAGE \bullet del\ speaks\ lang\ \wedge$
$(\forall otherDel: PERSON \mid otherDel \in delegates \setminus \{del\} \bullet$
$\neg(otherDel\ speaks\ lang)))$