Introduction to Schemas

Mairead Meagher

September, 2017

Contents

1 The Schema Concept

A specification in Z is made up of an informal, narrative text and formal descriptions written in Z. A schema is a graphical and modular way of grouping mathematics together.

The schema is used to factor out commonality in:

- Describing states
- Describing operations
- Defining types
- Writing predicates
- Stating theorems

1.1 The schema as a unit of structure

A schema describes some variables whose values are constrained.

- The variables are introduced in a declaration.
- The variables are constrained by a predicate.

The schema is thus made up of

- the declaration part and
- the predicate part.

$$S$$
$$a, b : \mathbb{N}$$
$$a < b$$

- This schema is called S.
- It declares two observations or variables a and b.
- It contains a constraining predicate or rule that a must be less than b.
- a and b are local to the schema S i.e. they are not available outside of S?'s scope.

The horizontal lines should extend far enough to the right to reach the longest line of the declarations or of the predicate.

A schema can be written in an equivalent linear form. This can be more convenient, especially in shorter schemas.

$$S == [a, b : \mathbb{N} \mid a < b]$$

The general form of a schema is:

$$\begin{array}{|l} \textit{SchemaNme} \\ \hline \textit{DeclarationPart} \\ \hline \textit{PredicatePart} \end{array}$$

and the linear form of the Schema is

$$\textit{SchemaName} == [\textit{DeclarationPart}\ \textit{PredicatePart}]$$

1.2 Axiomatic Definitions

It is possible to have variables or observations introduced by axiomatic definition. These then become globally defined within the specification.

Let us look at a system concerning enrollment of students on a course. If we wish to introduce a global variable maxOnCourse, we may simply write

$$\begin{array}{|l} \textit{maxOnCourse} : \mathbb{N} \\ \hline 6 \leq \textit{maxOnCourse} \leq 30 \end{array}$$

Later in the specification, schemas may refer to maxOnCourse without explicitly including their defining schemas.

$$\begin{array}{|l} \textit{Course} \\ \hline \textit{numberEnrolled} : \mathbb{N} \\ \hline \textit{numEnrolled} \leq \textit{maxOnCourse} \end{array}$$

1.3 Multiline Schemas

If the declaration part of the schema contains more than on line, each line is regarded (without having to write it) as being terminated with a semi-colon. If the predicate part of the schema contains more than one line, then the lines are regarded as being joined by and operators.

Schemas are usually multiline!

1.4 Simple Schema Calculus

Schemas can be regarded as units which can be manipulated by various operators.

Some examples follow. Some of these examples will be treated in more detail later.

1.5 Decoration

The schema decorated with prime S' is defined to be the same as S with all its variables also decorated with prime. It is used to signify the value of a schema after an operation has been carried out. Similarly for the variables - a' signifies the new value of a after some operation.

$$
\begin{array}{|l}
\hline
S' \\
a', b' : \mathbb{N} \\
\hline
a' < b' \\
\hline
\end{array}
$$

1.6 Decoration of input and output variable:

Conventionally

- input variables are decorated with a question mark, (?).
- output variables are decorated with an exclamation mark (!).

Note: the decoration characters are simply characters in the variables name.

$$
\begin{array}{|l}
\hline
Add \\
a?, b? : \mathbb{N} \\
sum! : \mathbb{N} \\
\hline
sum! = a? + b? \\
\hline
\end{array}
$$

1.7 Schema Inclusion

If the name of a schema is included in the declaration of another schema, this effects a textual import of the included schema. This means that both the declaration and predicate part are now available. The included predicated is and?ed with the including schema?s predicate part. e.g.

$$
\begin{array}{|l}
\hline
IncludeS \\
S \\
c : \mathbb{N} \\
\hline
c \leq 10 \\
\hline
\end{array}
$$

is another way of writing

$$
\begin{array}{|l}
\hline
IncludeS \\
a, b : \mathbb{N} \\
c : \mathbb{N} \\
\hline
a < b \\
c \leq 10 \\
\hline
\end{array}
$$

1.8 Schema conjunction

Two schemas can be joined by an and operator. The effect is to merge the declaration parts and ?and? the predicate parts of the component schemas. (Similar to schema inclusion) Example: Given S as before and T as

$$
\begin{array}{|l}
\hline
T \\
b, c : \mathbb{N} \\
\hline
b < c \\
\hline
\end{array}
$$

$$SAndT == S \land T$$

yields the following schema:

$$
\begin{array}{|l}
\hline
SandT \\
a, b, c : \mathbb{N} \\
\hline
a < b \land \\
b < c \\
\hline
\end{array}
$$

Variables that have the same type become merged as in b above. If the component schemas have each variables of the same name but with a different type, then the above conjunction is illegal.

1.9 Schema disjunction

Similarly, two schemas can be joined by an or operator. The effect in this case is to merge the declaration parts and ?or? the predicate parts of the component schemas. Example: Given S and T as before:

$$SOrT == S \lor T$$

yields the following schema:

$$
\begin{array}{|l}
\hline
SorT \\
a, b, c : \mathbb{N} \\
\hline
a < b \ \lor \\
b < c \\
\hline
\end{array}
$$

1.10 Delta Convention Δ

We have already seen the prime decoration convention. This is used to signify the value of a schema or variable after an operation. Generally, when we need the primed schemas (which implicitly include the primed variables of the schemas), we need the undecorated schema. The delta convention is used to include both the primed and original, unprimed schema. It thus used to signify a change and the usual mathematical delta (for change) symbol, Δ, is used.

$\Delta S == S \land S'$

or

ΔS

$a, b : \mathbb{N}$
$a', b' : \mathbb{N}$

$a < b$
$a' < b'$

Having defined a schema S, it is not necessary to define S? in order to use ΔS.

Having included ΔS in an operation schema, we must define the effect of the operation on each of the observations in S, even if the value of the observation/variable is left unchanged.

1.11 Ξ Convention

We use the greek Ξ to signify
there is no change in any of the observations of the schemas after an operation.
This is useful when specifying for instance a **read-only** type operation.

Δ S is defined as:

ΞS

$a, b : \mathbb{N}$
$a', b' : \mathbb{N}$

$a < b$
$a' < b'$
$a = a'$
$b = b'$

Similarly as for delta, it is not necessary to define ΞS as above, it is conventionally assumed. If, however, you redefine the meaning of Ξ in your specification, then you?ll need to define it as above i.e. explicitly. (This is not recommended!)

1.12 Events, Observations and States in Z

Z specifications describe three different entities

- states
- observations(variables)
- events(operations)

A state is a mathematical structure which models a system. An event is an occurrence which is of interest to the specifier e.g. adding a customer to a customer list. An observation is a set variable whose value can be examined before or after an event has occurred.

There are two types of properties of a state that a Z specification should reflect.

- Static Properties. These are the predicates that always hold no matter what event occurs. These properties are known as invariants.
- Observations/Variables. These characterise the effect of an event.

We will now look at a simple example of a system and show the schemas involved.

2 The Cursor Example [1]

2.1 Description of system

The display of a computer terminal shows lines of characters with each line consisting of a fixed number of columns containing a character in a fixed size typeface. A cursor marks a current position of interest on the display. The user can type keys, some of which directly control the position of the cursor.

2.2 Our area of interest

For the purposes of this example, we shall limit our investigation to the cursor control keys. We will define the state of the system and the operations defined by the various cursor control key presses (events). The cursor control keys are:

- Home key
- Down key
- Up key
- Return key
- Right key
- Left key

2.3 Types and Global Variables

First, look at types and global variables we need:

$$[KEY]$$

the set of keys on the terminal keyboard

$$num\_Lines, num\_Columns : \mathbb{N}$$
$$num\_Lines \geq 1$$
$$num\_Columns \geq 1$$

The lines are numbered from 1 to num_Lines down the display and the columns are numbered 1 to num_Columns across the display.

$$left, right, up, down, home, return : KEY$$
$$disjoint < left, rightup, down, home, return >$$

The operations for moving a cursor involve distinguishing the key pressed. We therefore need to define six special values of type KEY. There is no need to give their actual value. It is sufficient to note that the are different from each other (using disjoint).

2.4 The state

At any time, the cursor is within the bounds of the display. The state of the cursor is now described by the schema Cursor.

Cursor
$$line : \mathbb{N}_1$$
$$column : \mathbb{N}_1$$
$$line \leq num\_Lines$$
$$column \leq num\_Columns$$

2.5 Operations

We will examine the different operations for moving the cursor one at a time. We will finally state that a special key operation is one of the component operations.

2.5.1 HomeKey

$$
\begin{array}{|l}
\hline
HomeKey \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = home \\
\\
line' = 1 \\
column' = 1 \\
\hline
\end{array}
$$

Note that the three statements under the line are all true when the operation $HomeKey$ has been successfully implemented.

2.5.2 Down key

The down key causes the cursor to move to the same column position on the next line down. Note that there are two cases:

- the simple case when the cursor is not on the last row before pressing the down key,
- the case when the cursor is originally on the last line. In this case the cursor should wrap round to the first line of the screen and the column number should remain unchanged.

We will write two corresponding operations to deal with each case, writing appropriate preconditions in each operation.

$$
\begin{array}{|l}
\hline
DownKeyNormal \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = down \\
line \leq num\_Lines \\
\\
line' = line + 1 \\
column' = column' \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
DownKeyAtBottom \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = down \\
line = num\_Lines \\
\\
line' = 1 \\
column' = column' \\
\hline
\end{array}
$$

The full behaviour of the Down key is defined as

$$DownKey == DownKeyNormal \lor DownKeyAtBottom$$

Note that in the predicate part of the schema, the state of each variable after the operation must be defined, even if it is left unchanged by the operation. In both cases above

$$column' = column$$

was needed to explicitly state that column was not changed. We call the DownKey operation robust, because it takes all possibilities into account. Later robust operations will include error trapping, but we don?t treat error input in this example.

2.5.3 Up Key

The operations for the up key are analogous to the down key. This is left as an exercise for the reader.

2.5.4 Return Key

The operation of the return key is to move the cursor to the leftmost column (i.e. column = 1) of the next line, the next line being defined as one of:

- if the cursor is not at the last line, then the next line down, i.e. line + 1
- if the cursor is at the last line, then the cursor must wrap around to the top line

Again, we will look at two separate operations, and write appropriate pre-conditions.

The first schema deals with the case of the cursor being in the ?*middle* (i.e. not at end) of screen.

$$
\begin{array}{|l}
\hline
\textit{ReturnKeyNormal} \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = return \\
line < numLines \\
\\
line' = line + 1 \\
column' = column' \\
\hline
\end{array}
$$

The next schema deals with the case of the cursor starting at the bottom of the screen.

$$
\begin{array}{|l}
\hline
\textit{ReturntKeyAtBottom} \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = return \\
line = numLines \\
line' = 1 \\
column' = 1 \\
\hline
\end{array}
$$

The full functionality of the Return key can be described as:

$$ReturnKey == ReturnKeyNormal \lor ReturnKeyAtBottom$$

Note that the two above component schemas have a lot in common. We could equivalently write the full ReturnKey in one schema (as ReturnKeyFull), by grouping together the common parts:

$$
\begin{array}{|l}
\hline
\textit{ReturntKeyFull} \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = return \\
((line < numLines \land line' = line + 1 \\
\lor \\
(line = numLines? \land line? = 1)) \\
column' = 1 \\
\hline
\end{array}
$$

We tend to write in the 'simpler' form, i.e. using the two separate operations and bringing them together using $\lor$

2.5.5 Right Key

Next, examine the functionality of the right key. For this key, there are three possibilities:

- the normal case i.e. the cursor is originally not at the far right of the display, in this case the cursor is moved on to the right, the line number is not changed.
- the case when the cursor is on the rightmost column, but not on the bottom line. In this case the cursor is wrapped around to the start of the next line, i.e. to column? = 1 and line? = line + 1.
- the case when the cursor is on the bottom right corner of the screen. A right key will cause the cursor to wrap to the left of the top line i.e. line? = 1, column? = 1.

We will describe the three resulting schemas:

$$
\begin{array}{|l}
\hline
RightKeyNormal \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = right \\
column \leq num\_Columns \\
\\
line' = line \\
column' = column + 1 \\
\hline
\end{array}
$$

Next, we look at the second case, when the cursor wraps around to the beginning of the next line, this not being the bottom line.

$$
\begin{array}{|l}
\hline
RightKeyAtEdge \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = right \\
column = num\_Columns \\
line \leq num\_Llines \\
\\
column' = 1 \\
line' = line + 1 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
RightKeyAtBottom \\
\Delta Cursor \\
key? : KEY \\
\hline
key? = right \\
column = num\_Columns \\
line \leq numLines \\
\\
line' = line + 1 \\
column' = 1 \\
\hline
\end{array}
$$

2.5.6 Left Key

The action of LeftKey is analogous to the action of RightKey and is left as an exercise to the reader.

2.6 The Full Solution

The action of the cursor on pressing of any of these cursor-control keys can be defined by the schema CursorControlKey as:

$$CursorControlKey == HomeKey \lor ReturnKey \lor UpKey \lor DownKey \lor LeftKey \lor RightKey$$

This is a fully robust operation.

References

[1] Lightfoot *Formal Specification in Z* 1991, Palgrave, ISBN 0-333-76327-0.