Introduction to Schemas

Mairead Meagher

Waterford Institute of Technology,

2017

Waterford Institute *of* Technology

INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introduction to Schemas

- What are schemas?
- How to write schemas
- First system using schemas

What are Schemas

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.
We use a schema is used to factor out commonality in:

- Describing states
- Describing operations
- Defining types

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.

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

The schema contd.

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

- 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.

Linear form of schemas

An equivalent linear form:

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

We usually use the linear form when it's a short (one predicate) schema.
Schemas are usually multiline!

Axiomatic Definitions

It is possible to have variables or observations introduced by axiomatic definition. These then become globally defined within the specification.
e.g. introduce a global variable maxOnCourse:

$$maxOnCourse : \mathbb{N}$$
$$\vdash 6 \leq maxOnCourse \leq 30$$

Now globally defined.

Simple Schema Calculus

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

Schema 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.

S'

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

$a' < b'$

Decoration of input and output variable:

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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' names.

Add

$$a?, b? : \mathbb{N}$$
$$sum! : \mathbb{N}$$

$$sum! = a? + b?$$

Schema Inclusion

IncludeS

S

$c : \mathbb{N}$

$c \leq 10$

is another way of writing

IncludeS

$a, b : \mathbb{N}$

$c : \mathbb{N}$

$a < b$

$c \leq 10$

Schema conjunction

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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:

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

Schema disjunction

Given S and T as before:

$$SorT == S \lor T$$

yields the following schema:

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

Delta Convention Δ

The delta convention is used to include both the before and after state.

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

or

ΔS

$a, b : \mathbb{N}$

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

$a < b$

$a' < b'$

Now, 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.

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:

$$\Xi S$$

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

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

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.

Reflecting Properties

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.

The Cursor Example

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.

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

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.

$$left, right, up, down, home, return : KEY$$

$$\underline{disjoint} < \{left\}, \{right\}, \{up\}, \{down\}, \{home\}, \{return\} >$$

System State

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).
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

$$line : \mathbb{N}_1$$
$$column : \mathbb{N}_1$$

$$line \leq num\_Lines$$
$$column \leq num\_Columns$$

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.

HomeKey

$\Delta Cursor$

$key? : KEY$

$key? = home$

$line' = 1$

$column' = 1$

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

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.

DownKeyNormal

$\Delta Cursor$

$key? : KEY$

$key? = down$

$line < num\_Lines$

$line' = line + 1$

$column' = column$

DownKeyAtBottom

$\Delta Cursor$

$key? : KEY$

$key? = down$

$line = num\_Lines$

$line' = 1$

$column' = column$

The full behaviour of the Down key is defined as

$$DownKey == DownKeyNormal \lor DownKeyAtBottom$$

Alternative DownKey

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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.

UpKey

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

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.

Return Key

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

ReturnKeyNormal

$$\Delta Cursor$$
$$key? : KEY$$

$$key? = return$$
$$line < numLines$$

$$line' = line + 1$$
$$column' = 1$$

Return Key

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

ReturntKeyAtBottom

$\Delta Cursor$

$key? : KEY$

$key? = return$

$line = numLines$

$line' = 1$

$column' = 1$

Return Key

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

$$ReturnKey == ReturnKeyNormal \vee ReturnKeyAtBottom$$

Return Key

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:

ReturnKeyFull

$$\Delta Cursor$$
$$key? : KEY$$

$$key? = return$$
$$((line < numLines \land line' = line + 1$$
$$\lor$$
$$(line = numLines \land line' = 1))$$
$$column' = 1$$

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

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:

RightKeyNormal

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

RightKeyNormal

$\Delta Cursor$

$key? : KEY$

$key? = right$

$column < num\_Columns$

$line' = line$

$column' = column + 1$

RightKeyAtEdge

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.

RightKeyAtEdge

$$\Delta Cursor$$
$$key? : KEY$$

$$key? = right$$
$$column = num\_Columns$$
$$line < num\_Llines$$

$$column' = 1$$
$$line' = line + 1$$

RightKeyAtBottom

RightKeyAtBottom

$\Delta Cursor$

$key? : KEY$

$key? = right$

$column = num\_Columns$

$line = numLines$

$line' = 1$

$column' = 1$

Left Key

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

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.

Any questions?