Z Notations

Dr. Rong Qu

rxq@cs.nott.ac.uk

http://www.cs.nott.ac.uk/~rxq/#g53fsp
Introduction

We use mathematical notation so that we will be able to prove certain properties of the system directly from the specification. i.e. it is consistent and it is complete. Answer questions about the system. i.e. ”Can such and such a situation ever arise?”

produce computer programs directly from the specification, or confirm that an existing program conforms the specification.
But

There remains always, of course, the problem of proving that our mathematics actually represents the real-world problem that we are trying to represent.
Schema

The specification is broken down into small units called schema.

Each schema will have

- declaration part and
- a logical or predicate part
Identifiers

Identifiers followed by a prime ’ indicate the values of objects after the action has taken place

Identifiers followed by a question mark ? indicate input values identifiers

Identifiers followed by a exclamation ! indicate output values
A State Schema

Assume a particular possible state of our system to be

\[ \text{known} = \{Joy, Eric\} \]

known as a set of names
A State Schema

\[height = \{(Joy, 6\text{feetand3inches}),
(Eric, 5\text{feetand2inches})\}\]

a function mapping names to heights

\[weight = \{(Joy, 7\text{stonesand2pounds}),
(Eric, 17\text{stonesand10pounds})\}\]

a function mapping names to weights
A State Schema

The function $height$, $weight$ could be equally written

$$height = \{(Joy \mapsto 6 \text{ feet and } 3 \text{ inches}), \ (Eric \mapsto 5 \text{ feet and } 2 \text{ inches})\}$$

$$weight = \{(Joy \mapsto 7 \text{ stones and } 2 \text{ pounds}), \ (Eric \mapsto 17 \text{ stones and } 10 \text{ pounds})\}$$
State-space Schema

describes the logic of the overall state of our system

\[ \text{NAME, HEIGHT, WEIGHT} \]

\[
\text{Height_and_Weight} \quad \text{______________________}
\]

\[
\text{known_height} : P \ \text{NAME}
\]

\[
\text{known_weight} : P \ \text{NAME}
\]

\[
\text{height} : \text{NAME} \rightarrow \text{HEIGHT}
\]

\[
\text{weight} : \text{NAME} \rightarrow \text{WEIGHT}
\]

\[
\text{known_height} = \text{dom height}
\]

\[
\text{known_weight} = \text{dom weight}
\]
The Declaration Part

The initial line

\[ \text{NAME, HEIGHT, WEIGHT} \]

declares that \textit{NAME, HEIGHT and WEIGHT} are three basic data types

we will not be defining them further in this specification
The Declaration Part

\[
\begin{align*}
\text{known\_height} & : \mathcal{P} \text{ NAME} \\
\text{known\_weight} & : \mathcal{P} \text{ NAME} \\
\text{height} & : \text{NAME} \rightarrow \text{HEIGHT} \\
\text{weight} & : \text{NAME} \rightarrow \text{WEIGHT}
\end{align*}
\]

declares that

\[
\begin{align*}
\text{known\_height} \text{ and } \text{known\_weight} & \text{ are to be sets of } \text{NAMEEs} \\
\text{height} \text{ and } \text{weight} & \text{ are to be partial functions which will act on a } \text{NAMEEs} \text{ to give a } \text{HEIGHT} \text{ or a } \text{WEIGHT} \text{ respectively}
\end{align*}
\]
The Predicate Part

The lower part of the schema

\[
\text{known\_height} = \text{dom height} \\
\text{known\_weight} = \text{dom weight}
\]

consists of logical statements which define the system

the set \text{known\_height} is to be exactly equal to the domain of the \text{height} function

the set \text{known\_weight} is to be exactly equal to the domain of the \text{weight} function
The Predicate Part

This part of the schema declares logical statements which are always true, and are invariants of the system.

If there are several statements in the predicate part, their order is immaterial; they all represent conditions which must be true.

Note that known_height and known_weight are derived objects.
Operations and Their Schema

We can declare the action of adding a new height to the list as the schema

This is known as an operation schema since it describes the change to the system brought about by a given operation or event.
Operations and Their Schema

\begin{align*}
\text{New\_Height} \\
\Delta \text{Height\_and\_Weight} \\
\text{name?} : \text{NAME} \\
\text{hgt?} : \text{HEIGHT} \\
\text{name?} \notin \text{known\_height} \\
\text{height}' = \text{height} \cup \{ \text{name?} \mapsto \text{hgt?} \} \\
\text{weight}' = \text{weight}
\end{align*}
Included Schema

$\Delta Height_{and\_Weight}$

state that the schema $Height_{and\_Weight}$ will be used with both its declarations and predicates.

The symbol $\Delta$ in front of the name indicates that we wish to use this schema in association with a state change.

In any state change, a primed identifier indicates the value after change, the unprimed identifier represents the value before the change.
The △ Inclusion

By including the schema using

△Height_and_Weight

we are automatically including all the declarations

known_height, known_height' : P NAME
known_weight, known_weight' : P NAME
height, height' : NAME → HEIGHT
weight, weight' : NAME → WEIGHT
The \( \Delta \) Inclusion

The \( \Delta \text{Height\_and\_Weight} \) also causes the predicates from \( \text{Height\_and\_Weight} \) to be included in the predicate part

\[
\begin{align*}
\text{known\_height} &= \text{dom height} \\
\text{known\_weight} &= \text{dom weight} \\
\text{known\_height'} &= \text{dom height'} \\
\text{known\_weight'} &= \text{dom weight'}
\end{align*}
\]
New_Height

We also declare that there will be an input argument \textit{name}? of type \textit{NAME}, and a second input argument \textit{hgt}? of the type \textit{HEIGHT}.
Pre- and Post- Conditions

\[ \text{name?} \notin \text{known\_height} \]

This predicate is known for obvious reasons as a pre-condition, defining conditions which must hold when the operation starts.

The second and third predicates are post-conditions

\[ \text{height}' = \text{height} \cup \{ \text{name?} \mapsto \text{hgt?} \} \]
\[ \text{weight}' = \text{weight} \]
Pre- and Post- Conditions

We could also describe

\[
\text{known\_height} = \text{dom\_height} \\
\text{known\_weight} = \text{dom\_weight}
\]

as pre-conditions, and

\[
\text{known\_height}' = \text{dom\_height}' \\
\text{known\_weight}' = \text{dom\_weight}'
\]

as post-conditions
Consistency Checks

After the operation has taken place, we would expect that

\[ \text{known\_height}' = \text{known\_height} \cup \{ \text{name}? \} \]
Observation Schema

An observation schema is one which provides information about the state of the system, without changing the state.

To find a given person’s weight, for example, we use the schema.
Observation Schema

\[
\text{Find\_Weight}
\]
\[
\exists \text{Height\_and\_Weight}
\]
\[
\text{name}\,? : \text{NAME}
\]
\[
\text{wgt}\,! : \text{WEIGHT}
\]
\[
\text{name}\,? \in \text{known\_weight}
\]
\[
\text{wgt}\,! = \text{weight}\, \text{name}\,?
\]
Invariant $\Xi$ Inclusion

$\Xi Height\_and\_Weight$

is an extension of the $\Delta Height\_and\_Weight$ idea introduced earlier.

It introduces the $\Delta Height\_and\_Weight$ schema and, since we have an observation schema with no change in the system data, it provides the additional predicates.
Invariant $\Xi$ Inclusion

\[
\begin{align*}
\text{known\_height}' &= \text{known\_height} \\
\text{known\_weight}' &= \text{known\_weight} \\
\text{height}' &= \text{height} \\
\text{weight}' &= \text{weight}
\end{align*}
\]

The exclamation mark in $wgt!$ indicates that this is an output object.
A Query Schema

It is possible to construct a schema which will have as inputs as specific height and weight and will have as output the set of people who have both that height and that weight.
A Query Schema

Who_is_that_high_and_that_tall

\[ \exists \text{Height\_and\_Weight} \]

\[ hgt? : \text{HEIGHT} \]

\[ wgt? : \text{WEIGHT} \]

\[ \text{names!} : \text{P NAME} \]

\[ \text{names!} = \{ n : \text{known\_height} \mid \text{height } n = hgt? \} \]

\[ \cap \{ n : \text{known\_weight} \mid \text{weight } n = wgt? \} \]
Error Messages and the Like

We need a free type definition as follows

\[
REPORT ::= \text{ok} | \text{height\_already\_anown} |
\text{height\_not\_known} | \text{weight\_already\_known} |
\text{weight\_not\_know}
\]

We need one extra schema to define a successful result

\[
\begin{array}{c}
\text{Success} \\
\hline
\text{report!} : REPORT \\
\hline
\text{report!} = \text{ok}
\end{array}
\]
Error Messages and the Like

\(\text{New\_Height} \land \text{Success}\)

gives a schema which adds

\(\text{report}! : \text{REPORT}\)

to the \(\text{New\_Height}\) predicate part.

\(\text{report}! = \text{ok}\)

to the \(\text{New\_Height}\) declaration part.
**Height_Already_Known Schema**

\[
\begin{align*}
\text{Height\_Already\_Known} & \quad \exists \text{Height\_and\_Weight} \\
\text{name}? : \text{NAME} & \\
\text{report!} : \text{REPORT} \\
\text{name?} \in \text{known\_height} & \\
\text{report!} = \text{height\_already\_known}
\end{align*}
\]
Height_Already_Known Schema

Now combine the schema

\[(\text{New Height} \land \text{Sccuess}) \lor \text{Height Already Known}\]

A full definition is

\[\text{Full New Height} \triangleq \]
\[(\text{New Height} \land \text{Sccuess}) \lor \text{Height Already Known}\]
The Full Equivalent

\[
\begin{align*}
\text{Full}_\text{New}_\text{Height} \\
\text{known}_\text{height}, \text{known}_\text{height}' : P \text{ NAME} \\
\text{known}_\text{weight}, \text{known}_\text{weight}' : P \text{ NAME} \\
\text{height}, \text{height}' : \text{NAME} \rightarrow \text{HEIGHT} \\
\text{weight}, \text{weight}' : \text{NAME} \rightarrow \text{WEIGHT} \\
\text{name}? : \text{NAME} \\
\text{hgt}? : \text{HEIGHT} \\
\text{report}! : \text{REPORT}
\end{align*}
\]

\[
\begin{align*}
\text{name}? \notin \text{known}_\text{height} \land \text{height}' &= \text{height} \cup \{\text{name}? \mapsto \text{hgt}?\} \land \text{weight}' = \text{weight} \land \\
\text{known}_\text{height} &= \text{dom height} \land \text{known}_\text{weight} = \text{dom weight} \land \\
\text{known}_\text{height}' &= \text{dom height}' \land \\
\text{known}_\text{weight}' &= \text{dom weight}' \land \text{report}! = \text{ok}) \\
\lor \ (\text{name}? \in \text{known}_\text{height} \land \text{height}' &= \text{height} \land \text{weight}' = \text{weight} \land \\
\text{known}_\text{height} &= \text{dom height} \land \text{known}_\text{weight} = \text{dom weight} \land \\
\text{known}_\text{height}' &= \text{dom height}' \land \text{known}_\text{weight}' &= \text{dom weight}' \land \\
\text{report}! &= \text{heigh\_already\_known})
\end{align*}
\]
**Weight_Not_Known Schema**

\[ \exists \text{Height_and_Weight} \]

\[ \text{name?} : \text{NAME} \]

\[ \text{report!} : \text{REPORT} \]

\[ \text{name?} \notin \text{known_weight} \]

\[ \text{report!} = \text{weight_not_known} \]
**Full_Find_Weight Schema**

For a full version of the \textit{Find\_Weight} schema, we can define

\[
Full\_Find\_Weight \; \equiv \\
(Find\_Weight \land Success) \lor Weight\_Not\_Known
\]
Pre- and Post- Conditions

The transaction operation will update the value of the global variable `till_state` upon input of one integer parameter `transaction`.

If `transaction` is greater than or equal to 1000, then `till_state` is to be set to 2; otherwise `till_state` will be set to the value 1.

The value of `transaction` will be greater than zero on entry, and will not be changed by the procedure. The value of `till_state` on entry will be 1 or 2.
Pre- and Post- Conditions

Pre-condition

\[ \text{transaction} \geq 0 \land (\text{till\_state} = 1 \lor \text{till\_state} = 2) \]

Post-condition

\[
(\text{transaction} \geq 1000 \Rightarrow \text{till\_state}' = 2) \\
\land (\text{transaction} < 1000 \Rightarrow \text{till\_state}' = 1) \\
\land \text{transaction}' = \text{transaction}
\]
The statement "For all values of \( x \) in the set \( S \) the logical expression \( P(x) \land Q(x) \) holds" is written as

\[
\forall x : S \bullet P(x) \land Q(x)
\]
Notational Difference

For sets consisting of all the integers in a given numeric range, we write the set of integers from 1 to 100 inclusive as "1..100".

For the set of natural numbers including zero (0, 1, 2, ...) we write $\mathbb{N}$

For the natural numbers starting at 1 we write $\mathbb{N}_1$

For all integers (negative, zero and positive) $\mathbb{Z}$
Notational Difference

We also have multiple variables ranging over the same set

$$\forall i, j, k : S_1 \bullet \ldots$$

$$\forall i, j, k : S_1; x, y, z : S_2 \bullet \ldots$$
Unique Exist Quantifiers

Unique exists

\[ ∃! x : S \bullet < \text{logical expr} > \]

There exists exactly one \( x \) in \( S \) such that ...

\[ ∀ n : N \bullet ∃! m : N \bullet m = \text{succ}(n) \]

Every natural number has a unique number which follows it.
Counting Quantifier in Z

How many exist. This is written

$$\Omega x : S \bullet < \text{logical expr} >$$

$$\exists x : S \bullet P(x) \iff (\Omega x : S \bullet P(x)) > 0$$

$$\exists!x : S \bullet P(x) \iff (\Omega x : S \bullet P(x)) = 1$$

$$\Omega \text{ account : all_accounts} \bullet \text{balance account} < 0$$
Summation Quantifier

Summation

\[ \sum \ x : S \bullet < \text{numeric expr} > \]

\[ \sum \ \text{account} : \text{all_accounts} \bullet \text{balance account} \]
Note 1

You should be careful using the above that you use logical and numerical expressions.

\[ \forall, \exists \text{ and } \Omega \text{ are followed by a logical expression} \]

\[ \sum \text{ uses a numeric expression} \]

and results

\[ \forall \text{ and } \exists \text{ deliver logical results} \]

\[ \Omega \text{ and } \sum \text{ deliver numeric results} \]

in their correct places
Note 2

Conventions for the empty set (written \{\}) are that

\[ \forall x: \{\} \bullet P(x) \text{ is true} \]

\[ \exists x: \{\} \bullet P(x) \text{ is false} \]

\[ \exists! x: \{\} \bullet P(x) \text{ is false} \]

\[ \Omega x: \{\} \bullet P(x) \text{ is zero} \]

\[ \sum x: \{\} \bullet N(x) \text{ is zero} \]
Summary

Schema Introduction

State Schema (Declaration & predicate parts)

Operation Schema (inclusion $\Delta$)

Observation (invariant inclusion $\Xi$)

Others
Error message, Pre- and post- condition, Notational Differences