Formal Specification with Z

Software Engineering

Andreas Zeller • Saarland University

Forward

$\Delta Editor$
$ch? : CHAR$

$ch? = right\_arrow$
$right \neq \langle\rangle$

$left' = left \frown head(right)$
$right' = tail(right)$

1

These slides are based on Jonathan Jackys “The Way of Z”

Outline

- Why formal specification?
- The Z specification notation
- Elements of Z
- Case Study: Version Control System

2

Why Formal Specification?

Most people accept *bugs* as something *unavoidable*. The *reasons* for this are:

- *Complexity* of the task.
- Insufficiency of the *tests*.
- Deficiencies in the *Environment*.
- *Economic* constraints.
- Lack of *foundations*.

We look at these reasons more closely.

3

Popular Fallacies: Complexity

Assumption

„Programmers can never imagine the countless ways in which a program can fail, or the many different ways a program is used."

— *Leonard Lee, Journalist*

4

Popular Fallacies: Complexity

Assumption

„Programmers can never imagine the countless ways in which a program can fail, or the many different ways a program is used."

— *Leonard Lee, Journalist*

Alternative

Software shouldn't be too complicated. We can produce a *compact* description that explains how a program is supposed to behave.

5

Popular Fallacies: Tests

Assumption

„There are always particular combinations of circumstances that have somehow eluded the test plan"

— *Mitch Kapor, Lotus*

6

Popular Fallacies: Tests

Assumption

„There are always particular combinations of circumstances that have somehow eluded the test plan"

— *Mitch Kapor, Lotus*

Alternative

Software must be made correct by construction – not testing. The proper role of testing is to confirm our understanding of the requirements and the environment.

7

Popular Fallacies: Environment

Assumption

No matter what we do, our programs will sometimes fail anyway. There are bugs in our compilers, operating systems, window managers, and all the other system software on which we depend.

8

Popular Fallacies: Environment

Assumption

No matter what we do, our programs will sometimes fail anyway. There are bugs in our compilers, operating systems, window managers, and all the other system software on which we depend.

9

Popular Fallacies: Economics

Assumption

For most applications, the market does not demand high quality. After all, „it is only software".

— *Anonymous Programmer*

10

Popular Fallacies: Economics

Assumption

For most applications, the market does not demand high quality. After all, „it is only software".

— *Anonymous Programmer*

Alternative

Coping with mediocre software is expensive. Mediocre software can cause sever financial losses or even cost lives. Less spectacular is the daily effort spend to identify and prevent those problems.

11

Popular Fallacies: Foundations

Assumption

„The number of times civil engineers make mistakes is very small. And at first you might think, what's wrong with us? It's because it's like we're building the first skyscraper every time."

— *Bill Gates, Microsoft*

12

Popular Fallacies: Foundations

Assumption

„The number of times civil engineers make mistakes is very small. And at first you might think, what's wrong with us? It's because it's like we're building the first skyscraper every time."
— Bill Gates, Microsoft

Alternative

Computing is a mature science. We can build on a legacy of logical ingenuity that reaches back to antiquity. We can – yes, we *have to* become better.

The specification notation Z…

- is a set of conventions for presenting *mathematical text*
- describes *computing systems* (hardware as well as software).
- was developed 1977–1990 at the University of Oxford with industrial partners (IBM, Inmos)
- is standardized (ANSI, BSI, ISO)
- has its name from *Ernst Zermelo* (axiomatic set theory of Zermelo-Fraenkel)
- is the most widespread specification language today

A First Example in Z

We look at the following C function:

```
int f(int a)
{
    int i, term, sum;

    term = 1; sum = 1;
    for (i = 0; sum <= a; i++) {
        term = term + 2;
        sum = sum + term;
    }
    return i;
}
```

What does this code do?

A First Example in Z

Again, but with documentation:

```
int iroot(int a)  // Integer square root
{
    int i, term, sum;

    term = 1; sum = 1;
    for (i = 0; sum <= a; i++) {
        term = term + 2;
        sum = sum + term;
    }
    return i;
}
```

How does this code work?

A First Example in Z

The name and the comment for `iroot` are not as helpful as they might seem.

- Some numbers don't have integer square roots.. What happens if you call `iroot` with a set to 3?

A First Example in Z

The name and the comment for `iroot` are not as helpful as they might seem.

- Some numbers don't have integer square roots.. What happens if you call `iroot` with a set to 3?
- For negative numbers integer square roots are not defined (in $\mathbb{R}$). What happens if you call `iroot` with a set to -4?

A First Example in Z

The name and the comment for `iroot` are not as helpful as they might seem.

- Some numbers don't have integer square roots.. What happens if you call `iroot` with `a` set to 3?
- For negative numbers integer square roots are not defined (in $\mathbb{R}$). What happens if you call `iroot` with `a` set to -4?

Conclusion: Names and comments are not enough to describe the behavior completely.

A First Example in Z

Here is a specification for `iroot` – in a Z-*paragraph:*

$$
\begin{array}{|l}
iroot : \mathbb{N} \to \mathbb{N} \\
\hline
\forall\, a : \mathbb{N} \bullet iroot(a) * iroot(a) \leq a < (iroot(a) + 1) * (iroot(a) + 1)
\end{array}
$$

This *axiomatic definition* is as a paragraph indented and (typically) the part of an bigger text.

Let's look at the different parts of the definition.

A First Example in Z

The *declaration* of *iroot*

$$iroot : \mathbb{N} \to \mathbb{N}$$

corresponds to the C-declaration

```
int iroot(int a)
```

Recognize: `iroot` doesn't receive negative numbers and also returns none.

A First Example in Z

The *Predicate*

$\forall a : \mathbb{N} \bullet iroot(a) * iroot(a) \leq a < (iroot(a) + 1) * (iroot(a) + 1)$

shows that *iroot* returns the *biggest* integer square root:

$$iroot(3) = 1$$
$$iroot(4) = 2$$
$$iroot(8) = 2$$
$$iroot(9) = 3$$

The predicate corresponds to the C function definition – it describes only *what* the function does without explaining *how* to do it.

Specifying a Text Editor

Let's look at a simple text editor.

We can

- insert text
- move the cursor right and left
- delete the character in front of the cursor

Basic Types

We declare a *basic type* [...] as the set of all characters:

$[CHAR]$

We make no further statement about $CHAR$ – because this is a specification!

Basic Types

We declare a *basic type* [...] as the set of all characters:

$[CHAR]$

We make no further statement about $CHAR$ – because this is a specification!

We introduce $TEXT$ as an *abbreviation definition* for a sequence of characters:

$TEXT == \mathrm{seq}\ CHAR$

Abbreviations are like *macros* in traditional programming languages.

Axiomatic Descriptions

An *axiomatic description* defines a constant for the whole specification – such as the size of the text:

$$\begin{array}{|l} maxsize : \mathbb{N} \\ \hline maxsize \leq 65535 \end{array}$$

$maxsize$ is a constant, however we haven't specified its value.

In Z, functions such as $iroot$ are a kind of a constant.

A State Schema for the Editor

We model the document of the text editor as two texts: $left$ is the text *before* the cursor, and $right$ is text *following* it.

Here is the state schema for the editor:

$$\begin{array}{|l} \textit{Editor} \\ \hline left, right : TEXT \\ \hline \#(left \frown right) \leq maxsize \\ \hline \end{array}$$

$\frown$: concatenation operator
$\#$: count operator

Schemas

A *schema* describes an *aspect* of the specified system.

A schema consists of:

Name. Identifies the schema (often type name!).

Declaration part. Introduces local *state variables*.

Predicate part. Describes

- *State invariants* as well as
- *Relations*
  - between the state variables themselves or
  - between state variables and constants

Initialization Schemas

Every system has a special state in which it starts up. In Z this state is described by a schema conventionally named *Init*.

Init
Editor
$left = right = \langle\,\rangle$

$\langle\,\rangle$: empty sequence
The *Init* schema *includes* the *Editor* schema.
$\Rightarrow$ All definitions from *Editor* apply to *Init* as well.

The inclusion allows an *incremental* specification.

Printing Characters

We want to model the *insertion* of a character.

Therefore, we define *printing* as the set of printing characters.
(as axiomatic definition without predicates)

$printing : \mathbb{P}\, CHAR$

$\mathbb{P}$: Power set (= set of all subsets)

Insert Operation

The insertion is modeled by an *operation schema*.

Operation schemas define the effect of *functions*.

$$
\begin{array}{|l}
\textit{Insert} \\
\hline
\Delta Editor \\
ch? : CHAR \\
\hline
ch? \in printing \\
left' = left \frown \langle ch? \rangle \\
right' = right \\
\hline
\end{array}
$$

$\Delta Editor$: Operation schema on *Editor*
$ch?$: Input variable
$ch? \in printing$: Precondition
$left', right'$: State *after* the operation

Moving the Cursor

We define a *control character*…

$$
\begin{array}{|l}
right\_arrow : CHAR \\
\hline
right\_arrow \notin printing
\end{array}
$$

…and the corresponding operation:

$$
\begin{array}{|l}
\textit{Forward} \\
\hline
\Delta Editor \\
ch? : CHAR \\
\hline
ch? = right\_arrow \\
left' = left \frown head(right) \\
right' = tail(right) \\
\hline
\end{array}
$$

Moving the Cursor

We define a *control character*…

$$
\begin{array}{|l}
right\_arrow : CHAR \\
\hline
right\_arrow \notin printing
\end{array}
$$

…and the corresponding operation:

$$
\begin{array}{|l}
\textit{Forward} \\
\hline
\Delta Editor \\
ch? : CHAR \\
\hline
ch? = right\_arrow \\
left' = left \frown head(right) \\
right' = tail(right) \\
\hline
\end{array}
$$

Why does this definition not always work?

Moving the Cursor

We extend the definition with an additional *explicit precondition*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Forward}} \\
\Delta \textit{Editor} \\
ch? : \textit{CHAR} \\
\hline
ch? = \textit{right\_arrow} \\
\textit{right} \neq \langle\, \rangle \\
\textit{left}' = \textit{left} \frown \textit{head}(\textit{right}) \\
\textit{right}' = \textit{tail}(\textit{right}) \\
\hline
\end{array}
$$

Moving the Cursor

We extend the definition with an additional *explicit precondition*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Forward}} \\
\Delta \textit{Editor} \\
ch? : \textit{CHAR} \\
\hline
ch? = \textit{right\_arrow} \\
\textit{right} \neq \langle\, \rangle \\
\textit{left}' = \textit{left} \frown \textit{head}(\textit{right}) \\
\textit{right}' = \textit{tail}(\textit{right}) \\
\hline
\end{array}
$$

Forward stays *partial*: It only works under special (pre-)conditions.

Moving the Cursor

Goal: *Forward* should work in all situations.

We define a special „end of file“ condition …

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{EOF}} \\
\textit{Editor} \\
\hline
\textit{right} = \langle\, \rangle \\
\hline
\end{array}
$$

…as well as „character is right arrow“.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RightArrow}} \\
ch? : \textit{CHAR} \\
\hline
ch? = \textit{right\_arrow} \\
\hline
\end{array}
$$

Moving the Cursor

Now we define the *total Forward* operation:

$$T\_forward \mathrel{\widehat=} Forward \lor (EOF \land RightArrow \land \Xi Editor)$$

$\mathrel{\widehat=}$: defines a schema consisting of schemas
$\land$: combines states an operations
$\lor$: distinct alternatives
Ξ: schema remains unchanged

Exercise: Complete the specification of the text editor!

The Elements of Z

As every model based specification language,
Z has numerous type constructors and operations.

- Sets and Declarations
- Tuples and Binary Relations
- Enumerations and Sequences
- Predicates

Sets and Declarations

Sets $\{red, yellow, green\}$

Sets and Declarations

Sets $\{red, yellow, green\}$

Declarations $i : \mathbb{Z} \quad signal : LAMP$

40

Sets and Declarations

Sets $\{red, yellow, green\}$

Declarations $i : \mathbb{Z} \quad signal : LAMP$

Tuple $EMPLOYEE == ID \times NAME \times DEPARTMENT$

$$\begin{array}{|l} Andreas, Rahul : EMPLOYEE \\ \hline Andreas = (0019, andreas, computer\_science) \\ Rahul = (0020, rahul, computer\_science) \end{array}$$

41

Pairs and Binary Relations

Pairs $(0019, andreas)$

Binary Relations alternative notation for pairs:
$0019 \mapsto andreas$

$$\begin{array}{|l} phone : NAME \leftrightarrow PHONE \\ \hline phone = \{ \\ \quad naomi \mapsto 64011, \\ \quad andreas \mapsto 64011, \\ \quad andreas \mapsto 64012, \\ \quad rahul \mapsto 64013, \\ \quad \vdots \\ \} \end{array}$$

42

Pairs and Binary Relations

Pairs $(0019, \mathit{andreas})$

Binary Relations alternative notation for pairs:

$0019 \mapsto \mathit{andreas}$

$$\begin{array}{|l}
\mathit{phone} : \mathit{NAME} \mapsto \mathit{PHONE} \\
\hline
\mathit{phone} = \{ \\
\quad \mathit{naomi} \mapsto 64011, \\
\quad \mathit{andreas} \mapsto 64011, \\
\quad \mathit{andreas} \mapsto 64012, \\
\quad \mathit{rahul} \mapsto 64013, \\
\quad \vdots \\
\}
\end{array}$$

$\operatorname{dom} \mathit{phone}$

Pairs and Binary Relations

Pairs $(0019, \mathit{andreas})$

Binary Relations alternative notation for pairs:

$0019 \mapsto \mathit{andreas}$

$$\begin{array}{|l}
\mathit{phone} : \mathit{NAME} \mapsto \mathit{PHONE} \\
\hline
\mathit{phone} = \{ \\
\quad \mathit{naomi} \mapsto 64011, \\
\quad \mathit{andreas} \mapsto 64011, \\
\quad \mathit{andreas} \mapsto 64012, \\
\quad \mathit{rahul} \mapsto 64013, \\
\quad \vdots \\
\}
\end{array}$$

$\operatorname{dom} \mathit{phone} = \{\ldots \mathit{andreas}, \mathit{rahul}, \ldots\}$

Pairs and Binary Relations

Pairs $(0019, \mathit{andreas})$

Binary Relations alternative notation for pairs:

$0019 \mapsto \mathit{andreas}$

$$\begin{array}{|l}
\mathit{phone} : \mathit{NAME} \mapsto \mathit{PHONE} \\
\hline
\mathit{phone} = \{ \\
\quad \mathit{naomi} \mapsto 64011, \\
\quad \mathit{andreas} \mapsto 64011, \\
\quad \mathit{andreas} \mapsto 64012, \\
\quad \mathit{rahul} \mapsto 64013, \\
\quad \vdots \\
\}
\end{array}$$

$\operatorname{dom} \mathit{phone} = \{\ldots \mathit{andreas}, \mathit{rahul}, \ldots\}$

$\operatorname{ran} \mathit{phone}$

Pairs and Binary Relations

Pairs $(0019, andreas)$

Binary Relations alternative notation for pairs:
$0019 \mapsto andreas$

$$\begin{array}{|l}
phone : NAME \leftrightarrow PHONE \\
\hline
phone = \{ \\
\quad naomi \mapsto 64011, \\
\quad andreas \mapsto 64011, \\
\quad andreas \mapsto 64012, \\
\quad rahul \mapsto 64013, \\
\quad \vdots \\
\}
\end{array}$$

$\operatorname{dom} phone = \{\ldots andreas, rahul, \ldots\}$
$\operatorname{ran} phone = \{\ldots 64011, 64012, 64013 \ldots\}$

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!)$

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!) = \{64011, 64013\}$

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!) = \{64011, 64013\}$

Domain Restriction

$\{andreas, naomi\} \lhd phone$

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!) = \{64011, 64013\}$

Domain Restriction

$\{andreas, naomi\} \lhd phone = \{andreas \mapsto 64011, andreas \mapsto 64012, naomi \mapsto 64011\}$

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!) = \{64011, 64013\}$

Domain Restriction

$\{andreas, naomi\} \lhd phone = \{andreas \mapsto 64011, andreas \mapsto 64012, naomi \mapsto 64011\}$

Range Restriction

$phone \rhd \{64011\}$

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!) = \{64011, 64013\}$

Domain Restriction

$\{andreas, naomi\} \lhd phone = \{andreas \mapsto 64011, andreas \mapsto 64012, naomi \mapsto 64011\}$

Range Restriction

$phone \rhd \{64011\} = \{andreas \mapsto 64011, naomi \mapsto 64011\}$

52

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!) = \{64011, 64013\}$

Domain Restriction

$\{andreas, naomi\} \lhd phone = \{andreas \mapsto 64011, andreas \mapsto 64012, naomi \mapsto 64011\}$

Range Restriction

$phone \rhd \{64011\} = \{andreas \mapsto 64011, naomi \mapsto 64011\}$

Update

$phone \oplus \{rahul \mapsto 64014\}$

53

Operators for Relations

Lookup $phone(\!|\ \{rahul, naomi\}\ |\!) = \{64011, 64013\}$

Domain Restriction

$\{andreas, naomi\} \lhd phone = \{andreas \mapsto 64011, andreas \mapsto 64012, naomi \mapsto 64011\}$

Range Restriction

$phone \rhd \{64011\} = \{andreas \mapsto 64011, naomi \mapsto 64011\}$

Update

$phone \oplus \{rahul \mapsto 64014\} = \{andreas \mapsto 64011, andreas \mapsto 64012, rahul \mapsto 64014\}$

54

Enumerations and Sequences

Enumerations as *type definition* (no order)

$DAYS ::= fri \mid mon \mid sat \mid sun \mid thu \mid tue \mid wed$

Enumerations and Sequences

Enumerations as *type definition* (no order)

$DAYS ::= fri \mid mon \mid sat \mid sun \mid thu \mid tue \mid wed$

Sequence axiomatic definition

$$weekday : \mathrm{seq}\, DAYS$$
$$weekday = \langle mon, tue, wed, thu, fri \rangle$$

Enumerations and Sequences

Enumerations as *type definition* (no order)

$DAYS ::= fri \mid mon \mid sat \mid sun \mid thu \mid tue \mid wed$

Sequence axiomatic definition

$$weekday : \mathrm{seq}\, DAYS$$
$$weekday = \langle mon, tue, wed, thu, fri \rangle$$

Operators

$head(weekday) = mon$

Enumerations and Sequences

Enumerations as *type definition* (no order)

$DAYS ::= fri \mid mon \mid sat \mid sun \mid thu \mid tue \mid wed$

Sequence axiomatic definition

$$\begin{array}{|l} weekday : \mathrm{seq}\, DAYS \\ \hline weekday = \langle mon, tue, wed, thu, fri \rangle \end{array}$$

Operators

$head(weekday) = mon$

$week == sun \frown weekday \frown sat$

Enumerations and Sequences

Enumerations as *type definition* (no order)

$DAYS ::= fri \mid mon \mid sat \mid sun \mid thu \mid tue \mid wed$

Sequence axiomatic definition

$$\begin{array}{|l} weekday : \mathrm{seq}\, DAYS \\ \hline weekday = \langle mon, tue, wed, thu, fri \rangle \end{array}$$

Operators

$head(weekday) = mon$

$week == sun \frown weekday \frown sat$

Sequences are just functions whose domains are consecutive numbers, starting with one.

$weekday(3) = wed$

Logic

Predicates restrict the set of possible states.

$$\begin{array}{|l} d_1, d_2 : 1 \mathinner{..} 6 \\ \hline d_1 + d_2 = 7 \end{array}$$

Quantifiers

$$\begin{array}{|l} divides : \mathbb{Z} \leftrightarrow \mathbb{Z} \\ \hline \forall\, d, n : \mathbb{Z} \bullet d \,\underline{\mathrm{divides}}\, n \Leftrightarrow n \bmod d = 0 \end{array}$$

Binary relations (e.g. *divides*) can be written in infix syntax.

Further quantifiers: $\exists$ and $\exists_1$

Boolean Types

Z has *no* built-in *Boolean type.*

Reason – Boolean types often lead to unreadable specifications.

$BOOLEAN ::= true \mid false$

$$\begin{array}{|l} beam, door : BOOLEAN \\ \hline beam \Rightarrow door \qquad ??? \end{array}$$

61

Boolean Types

Z has *no* built-in *Boolean type.*

Reason – Boolean types often lead to unreadable specifications.

$BOOLEAN ::= true \mid false$

$$\begin{array}{|l} beam, door : BOOLEAN \\ \hline beam \Rightarrow door \qquad ??? \end{array}$$

Better:

$BEAM ::= off \mid on$
$DOOR ::= closed \mid open$

62

Boolean Types

Z has *no* built-in *Boolean type.*

Reason – Boolean types often lead to unreadable specifications.

$BOOLEAN ::= true \mid false$

$$\begin{array}{|l} beam, door : BOOLEAN \\ \hline beam \Rightarrow door \qquad ??? \end{array}$$

Better:

$BEAM ::= off \mid on$
$DOOR ::= closed \mid open$

so we can write $beam = on \Rightarrow door = closed$

63

Case Study: Revision Control

Many revision control systems use *locks*. This means that at any time only one person can edit the file.

We model such a system in Z.

First, we define the permissions for the documents:

$[PERSON, DOCUMENT]$

$$permission : DOCUMENT \leftrightarrow PERSON$$

Case Study: Revision Control

Example:

$$\begin{array}{l} doug, aki, phil : PERSON \\ spec, design, code : DOCUMENT \end{array}$$

$$permission = \{(spec, doug), (design, doug), (design, aki), \ldots\}$$

We model who has checked out a document:

$$\begin{array}{|l} \text{\_ } Documents \text{ \_\_\_\_\_\_\_\_\_\_} \\ checked\_out : DOCUMENT \rightharpoonup PERSON \\ \hline checked\_out \subseteq permission \\ \hline \end{array}$$

$\rightharpoonup$: partial function

Case Study: Revision Control

A schema for the *check-out* of a document:

$$\begin{array}{|l} \text{\_ } CheckOut \text{ \_\_\_\_\_\_\_\_\_\_} \\ \Delta Documents \\ p? : PERSON \\ d? : DOCUMENT \\ \hline d? \notin \operatorname{dom} checked\_out \\ (d?, p?) \in permission \\ checked\_out' = checked\_out \cup \{(d?, p?)\} \\ \hline \end{array}$$

Case Study: Revision Control

CheckOut needs to become a total operation:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CheckedOut}} \\
\Xi Documents \\
d? : DOCUMENT \\
\hline
d? \in \operatorname{dom} checked\_out \\
\hline
\end{array}
$$

67

Case Study: Revision Control

CheckOut needs to become a total operation:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CheckedOut}} \\
\Xi Documents \\
d? : DOCUMENT \\
\hline
d? \in \operatorname{dom} checked\_out \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Unauthorized}} \\
\Xi Documents \\
p? : PERSON \\
d? : DOCUMENT \\
\hline
(d?, p?) \notin permission \\
\hline
\end{array}
$$

$$T\_CheckOut \mathrel{\widehat{=}} CheckOut \lor CheckedOut \lor Unauthorized$$

68

Summary

- Z describes the behavior of a system by *schemas*.
- A schema describes an aspect of the specified system.
- Schemas can be composed to bigger schemas.
  - allows incremental definition
  - and incremental presentation
- Rich set of built-in types and operators.
- Basis for program proofs.

69

Literature

The Way of Z (Jonathan Jacky)
– all described examples and tutorials

The Z Notation
(http://www.comlab.ox.ac.uk/archive/z.html)

The Z Glossary (ftp://ftp.comlab.ox.ac.uk/pub/Zforum/zglossary.ps.Z)

Fuzz Type Checker
(http://spivey.oriel.ox.ac.uk/mike/fuzz/)
– Type checker and LaTeX macros for Linux