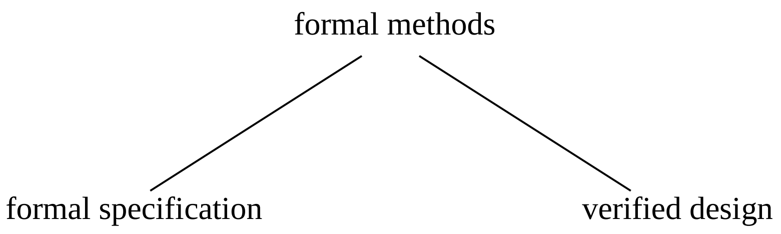
Overview

- What is a Formal Method?
- What is Formal Specification?
- What is a Z specification?

What is a FORMAL METHOD?

One categorization of formal methods is that formal methods are made up of two parts:



Other categorizations state that formal methods are those methods that apply mathematics to software engineering.

- Mathematics is used to define the behaviour of the system.

- Mathematics makes it possible to reason about the system.

How is mathematics used?

Specification
- Statement of **what** the software is to do in a precise and unambiguous manner.

Verification
- Using rules of logic, it can be proved formally that a program matches its specification. This is potentially mechanical and therefore can be automated using automated program provers.

Why is mathematics used?

- Mathematics is a precise language. There is no ambiguity. This gives the software engineer the precision needed.

- Mathematics is concise. Complex ideas can be expressed very concisely.

Mathematics is expressively very powerful. Most ideas are capable of being expressed in terms of mathematics.

- Mathematics facilitates formal reasoning. The consistency of what has been written can be checked.

- Using mathematics, simple, coherent, unifying features can be found

What is a FORMAL SPECIFICATION?

One of the most important documents that is produced as part of a systems life cycle is the specification. It models the behaviour of the system and is referred to constantly throughout the development process. It is important that the specification must be precise and unambiguous and ideally, therefore, should be written in mathematics. Such a specification is called a formal specification. A formal specification is the mathematical encapsulation of informal requirements.

Practitioners have found that writing and debugging a formal specification leads to a fuller understanding and knowledge of the application. It is also very effective in exposing gaps and inconsistencies in that knowledge.

The formal specification uses the notion of abstraction:

- Says ***what*** is to be achieved, not ***how*** it is to be achieved
- Defines whatever level of detail is necessary
- Uses application- oriented data types and structures, not computer-oriented ones.

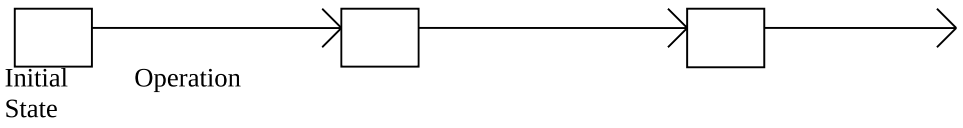
Types of Formal Specification Notations

Model-Based Specifications

(Often called State-Based)

Examples are Z, VDM[1], B[2]

Data types derived from set theory, such as sets, relations, functions and sequences are used to represent the components of the system. Logic is used to define what must be true about these components and the relationship between them. Operations are defined in terms of effect on state. Each operation is defined by the relationship which must hold between the state of the system before and after the operation.



Algebraic Specification Languages

[1] Systematic Software Development Using VDM, Jones, 2e, Prentice Hall, 1990.

[2] *The B-Book: Assigning Programs to Meanings*, J.-R. Abrial, Cambridge University Press, 1996. ISBN 0-521-49619-5.

Examples are Clear[3], ACT ONE[4], Larch[5].

These notations allow the description of a system and its constituent parts by means of sets of equations which capture behaviour directly. For large systems, mechanisms need to be provided for allowing the re-use of common components.

Process Algebra's

Examples are CCS[6] (Calculus of Communicating Systems), CSP[7] (Communicating Sequential Processes). The programming language occam[8] is based on CSP.

These allow a system to be modeled by a collection of processes which communicate with each other. Process Algebra's are therefore useful for the study of concurrent systems.

The role of specification in the lifecycle

The lifecycle of a system can vary depending on methodology used, type of system etc. The generic phases of a lifecycle are as follows:

- Requirements Analysis
- Design
- Implementation
- Testing
- Delivery
- Maintenance

The deliverable from the Requirements Analysis phase is the **specification of requirements**.

A formal specification is an encapsulation of informal requirements. It is refined to a **design**, which says (for the **first** time) **how** the requirements are to be satisfied, and thence to an implementation in a programming language.

Limitations of Formal Specifications

- Do not define non-functional attributes

[3]The semantics of Clear, a Specification Language, Burstall and Goguen, in Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification.

[4]Fundamentals of Algebraic Specification 1, Ehrig and Mahr, Springer-Verlag, 1985.

[5]Larch in Five Easy Pieces, Guttag and Horning, Digital Systems Research Center Report, July, 1985.

[6]A Calculus of Communicating Systems, Milner, Lecture Notes in Computer Science vol. 92, Springer-Verlag, 1980.

[7]Communicating Sequential Processes, Hoare, Prentice Hall , 1985.

[8]occam Programming Manual, INMOS Ltd., Prentice Hall, 1984.

- Performance
- Usability
- Size
- Availability

- Do not tell you **how** to design the system. It is possible to specify systems which cannot be implemented.

What is a Z Specification?

A Z Specification is a mixture of

- Formal mathematical statements
- Explanatory text

The formal mathematical statements have a **structure** imposed and there are **conventions** used. The mathematical component of the specification is based on set theory and one of Z's most distinguishing features is a structuring device called a **schema**.

Where did Z come from?

The Z notation was developed at Oxford University's Programming Research Group in the late 70's and early 80's by Jean-Raymond Abrial, Bernard Sufrin and Ib Sorenson. From the start, Z was used in industry. IBM used it to re-specify their Customer Information Control System (CICS)[9].
After experimentation, work in large systems in an industrial environment and theoretical investigation, a standard core language has been defined and can be found in Spivey [2].

The Mathematics of Z

Z is based on standard mathematical notation.
- Logic

- First order predicate calculus (the most basic form of formal logic).
- Set theory

- Strongly typed.
- Relations, functions, sequences all viewed as sets of couplets.
- Sets can be of any application-oriented type. This means that we can introduce sets at any level of abstraction. e.g. we could introduce the sets [NAME, BOOK] without further specifying what internal structures are involved.
- Extensibility
  - Minimal built in constructs.
  - Extensibility mechanisms.
  - Generic definitions.
  - Basic library.

[9]For further details on CICS see Hayes [3]

Structuring Specifications

On of Z's characteristic features is the structuring of specifications. The schema is one of the basic constructs. We can develop partial specifications and their combination. Fragments of mathematics can be named and later referred to. By using calculus of schemas, i.e. building up complex constructs from more simple schemas, we can build up large and complex specifications using simple building blocks.

Top-level structure

A Z formal specification is made up of formal text and explanatory English text. The formal text is made up of:

- **basic type definitions**, which define the values which constants and variables may take;
- **axiomatic descriptions**, which introduce global constants and variables;
- **constraints** on (previously declared) global variables;
- **schemas**, which define the state and operations of the system;
- **abbreviation definitions**, which introduce new global constants.

Each construct in the sequence may use names that are in scope.

A short example

Look at a simple **Birthday Book** system[10], i.e. a system which records peoples' birthdays and is able to issue a reminder on a person's birthday.
We need to deal with peoples' names and dates. We don't have to worry about the form the names and dates take, we can introduce them as basic types of the specification:

[NAME, DATE]

We need to describe the system's state space and we do this using a schema:

[10] This example is used as a running example in Spivey [2].

$$\begin{array}{|l}
\hline \textit{BirthdayBook} \\
\textit{known}: \mathbb{P}\textit{NAME} \\
\textit{birthday} : \textit{NAME} \mathrel{\ooalign{$\rightarrow$\cr\hidewidth$+$\hidewidth}} \textit{DATE} \\
\hline
\textit{known} = \textit{dom birthday} \\
\hline
\end{array}$$

We have just described a system with two components

- *known* - the set of known names
- *birthday* - a function which links a name to a date (that person's birthday)

We have also defined a relationship between these two components, a relationship which will not change for any state of the system. We say that the set *known* is the same as the domain of the *birthday* or the set of names for which we can validly check birthdays.
We call this special relationship the **invariant**.

Conventions
Conventions make reading and writing specifications easier
Allows compact definitions of state changes, inputs and outputs.

Conventions Example

$$\begin{array}{|l}
\hline \textit{Remind} \\
\Xi \textit{BirthdayBook} \\
\textit{today}? : \textit{DATE} \\
\textit{cards}! : \mathbb{P}\textit{PERSON} \\
\hline
\textit{cards}! = \{n{:}\textit{known} \mid \textit{birthday}(n) = \textit{today}? \bullet n\} \\
\hline
\end{array}$$

This shows the use of several conventions which we will see later.

Input parameters have a ?

Output parameters have a !.

Example of a Formal Specification

Introduction:

The RDS (Revised Defence Standard) describes the process which a Contractor must follow in order to supply a system with safety-critical attributes. The main definition in this specification is given by the schema `CompleteProjectState` which comprises all the objects which a Contractor must define. The state is built by introducing several schemas, named `ProjState1` through to `ProjState8`, each of which includes its predecessor. A later section summarizes the requirements.

Hazard Identification
The Contractor must first identify all hazards that are applicable to the system under analysis, together with the associated accidents and accident sequences. Hazards, accidents and events are modeled by abstract sets:

$$[HAZARD, ACCIDENT, EVENT]$$

The sets `hazards` and `accidents` model the identified hazards and accidents respectively; the accident sequences are modeled by the function `accSeqs`, and the identified accidents are those caused by at least one accident sequence.

_*ProjState*______________________________________

$hazards: \mathbb{P} HAZARD$

$accidents: \mathbb{P} ACCIDENT$

$accSeqs: ACCIDENT \leftrightarrow \mathrm{seq}_1\ EVENT$

$accidents = dom\ accSeqs$

Thus, for each accident, we record at least one non-empty sequence of events that could cause it.

..... and so on .....

Further Reading

1. The Essence of Z, Ed Currie, 1999, Pearson Education, ISBN 013749839X .

2. An Introduction to Formal Specification and Z, Potter, 2e,Sinclair and Till, 1996, Prentice Hall, ISBN 0-13-242207-7.

3. The Z Notation - A Reference Manual, 2nd Edition, Spivey, Prentice Hall, 1992, ISBN 0-13-978529-9.

4. Specification Case Studies, 2nd Edition, Hayes, Prentice Hall, 1993, ISBN 0-130832544-8.

5. Formal Specification in Z, Lightfoot, 2e, Palgrave, 1991, ISBN 0-333-76327-0.

6. Software Engineering Mathematics, Woodcock and Loomes, Pitman, 1988, 0-273-02673-9.

7. Z : An introduction to Formal Methods, Diller, Wiley, 1990, ISBN 0-471-97489-X.

8. Introducing Specification Using Z- A Practical Case Study Approach, Ratcliff, McGraw-Hill - Series in Software Engineering, ISBN 0-07-707965.

9. Z User Workshop, 1989 - 1997, (series of proceedings of Z User meetings), Springer-Verlag, ISBN various.

10.Z in Practice. Barden, Stepney, & Cooper, Prentice-Hall - The Practitioner Series, 1994, ISBN 0-13-124934-7.

11. Z Guide for beginners, McMorren & Powell, Blackwell Scientific, 1993, ISBN -0-632-03117-4.

12.An Introduction to Discrete Mathematics, Formal System Specification and Z, 2e, D.C. Ince ,Oxford Applied Mathematics and Computing Science Series, 1992, ISBN 0-19-853836-7.

13.Using Z : Specification, Refinement and Proof, 1996, Woodcock & Davies, Prentice-Hall, ISBN - 0-13-948472-8.

14.Software Development with Z - A Practical Approach to Formal Methods in Software Engineering, Addison-Wesley - International Computer Science Series, 1992, ISBN 0-201-62757-4.

15. Applications of Formal Methods, Eds Mike Hinchey, Jonathon Bowen, Prentice-Hall, 1996, ISBN - 0-13-366949-1.

Web Site
http://www.comlab.ox.ac.uk/archive/z.html