

# **Formal Methods in Software Development**

**Watcharee Jumpamule, Ph.D.**

**Department of Computer Science  
Chiangmai University**

# Outlines

- Overview of Formal Methods (3 hours)
- Formal Specification with Z (9 hours)
- Applications of Z specification (6 hours )
  - User Interface for mobile application
  - Neural Network of Intrusion Detection System

# Formal Methods

- *A **formal method** in software development is a method that provides a formal language for describing a software artifact (e.g. specifications, designs, source code) such that formal proofs are possible, in principle, about properties of the artifact so expressed.*
- Techniques based upon mathematics can be used at every stage of software development.
- Examples include: probability theory; relational calculus; context-free grammars.

- **abstract** *adj.* to do with or existing in thought rather than matter. *v.* take out of, extract, remove; summarize.
- **abstraction** *n.* the act or an instance of abstracting or taking away; an abstract or visionary idea.

# Example

When the first map of the London Underground was published in 1908, it was faithful to the geography of the lines: all the twists and turns of the tracks and the relative distances between stations were recorded faithfully and to scale.

However, the purpose of the map was to show travellers the order of stations on each line, and the various interchanges between lines; the density of the map made it difficult to extract this information.



- In 1933, the map was replaced by a more abstract representation, called the Diagram, which showed only the connectivity of stations.
- Abstracted were:
  - surface detail
  - distances between stations
  - orientation of lines



- The Diagram gives people a good conceptual model; it is how we see the London Underground network. It is a specification that allows people to make sense of a complex implementation.
- Furthermore, although revised regularly to reflect changes in the network, it is still the same diagram proposed in 1931 by engineering draughtsman Harry Beck.
- The success of the diagram is due to
  - an appropriate choice of abstraction
  - an elegant presentation

# Activities of Formal Methods

The main activities of Formal methods are,

- Writing a formal specification
- Proving properties about the specification
- Constructing a program by mathematically manipulating the specification
- Verifying a program by mathematical argument

# Seven Myths of Formal Methods

Seven myths about formal methods, and their rebuttal, are:

1. **Formal methods can guarantee that software is perfect.** Rather: they are very helpful at finding errors early on and can nearly eliminate some classes of error.
2. **They are all about program proving.** Rather: they work largely by making you think very hard about the system you propose to build.
3. **They are useful only for safety-critical systems.** Rather: they are useful for almost any application.
4. **They require highly trained mathematicians.** Rather: they are based on mathematical specifications, which are much easier to understand than programs.
5. **They increase the cost of development.** Rather: they can decrease the cost.
6. **They are unacceptable to clients.** Rather: they help clients understand what they are buying.
7. **They are not used on real, large scale software.** Rather: they are being used successfully on practical projects in industry.

Quoted from: J. Anthony Hall, Seven myths of formal methods, *IEEE Software*, 7(5):11-19, September 1990.

# Seven More Myths of Formal Methods

Seven myths about formal methods are:

1. Formal Methods delay the development process.
2. Formal Methods are not supported by tools..
3. Formal Methods mean forsaking traditional engineering design methods.
4. Formal Methods only apply to software.
5. Formal Methods are not required.
6. Formal Methods are not supported.
7. Formal Methods people always use Formal Methods.

Quoted from: Seven More Myths of Formal Methods:Dispelling Industrial Prejudices written by Jonathan P. Bowen and Michael G. Hinchey

# Safety Critical systems

- Applications with zero tolerance
- No failure is acceptable
  - Leads to catastrophic errors (irreversible)
- Design and implementation must be proved to be 100% correct!
- Examples
  - Flight navigation system, embedded systems in satellites, robotic controllers, ...

# What is needed?

- Requirements
  - must be stated clearly, precisely and unambiguously
  - must be shown to be consistent
  - must be expressed in a medium amenable for thorough analysis
  - must serve as the basis for deriving design
- Conclusion: Use a well-defined notation (the one with a strong mathematical foundation) to express the requirements
- Formal methods support precise and rigorous specifications of those aspects of a computer system capable of being expressed in the language.

# About formal specifications

- Formal specifications
  - are precise and unambiguous
  - can be proved to be consistent – universally accepted methods exist to prove consistency
  - can be used to analyze and prove properties of the application – with the use of the underlying formalism for the chosen specification language
  - can be used to derive a provably correct design – with the use of mathematical transformation
- After all, a program itself is a formal specification language!

# Integrating formal methods into software life cycle

- Type of application
  - Formal methods are not generally applied to all software projects
    - Some application domains are well understood and there may exist some known and well behaving products for these applications
    - Some applications are small and well understood and so application of formal methods may be a waste of time
  - The complexity of the application domain must be discussed before deciding to use formal methods

# Integrating formal ... (*cont.*)

- Type of application
  - Examples:
    - A hand-held calculator with simple arithmetic
    - A personal phone diary
    - A scheduling software for allocating classroom teaching in a university
    - A flight reservation system
    - A web-based data mine
  - Decide, for each case, whether or not the use of formal methods can be justified.

# Integrating formal ... (*cont.*)

- Choice of formal method and type of analysis
  - A number of formal methods available, each having a different mathematical formalism
  - The type of analysis required for an application may dictate the choice of formal method
  - Examples:
    - Information-oriented (Z, VDM, B, Object-Z, ...)
    - Property-oriented (OBJ3, Larch, ...)
    - Concurrency (CCS, CSP, ...)
    - Real-time (TRIO, RTOZ, ...)

# Integrating formal ... (cont.)

- Level of complexity
  - Formal method can be applied only to a portion of a specification
    - The portion that seems to be critical and definitely needs careful analysis and evaluation
    - Example: GUI design is generally not specified formally.
  - Different portions of the specifications can be specified using different formal methods
    - Compatibility problems and integration issues
- In fact, practitioners of formal methods frequently use formal methods solely for recording precise specifications, not for formal verifications ([Hall 90] and [Place 90]).

# Integrating formal ...(*cont.*)

- Tool support
  - Appropriate tool support is essential for the use of a formal method
    - Syntax and type checker
    - Theorem prover
    - Pretty printing of specifications
    - Design and/or code generator
    - ...

# Some Formal Methods

Some of the most well-known formal methods consist of or include specification languages for recording a system's functionality. These methods include:

- Z (pronounced "Zed")
- Communicating Sequential Processes (CSP)
- Vienna Development Method (VDM)
- Larch
- Formal Development Methodology (FDM)

# Classification of formal specification languages

- Property-oriented
  - Used to describe certain properties of the chosen application
    - Axiomatic – OBJ3, Larch
    - Algebraic – CCS, CSP
- Model-based
  - Enables the specifier to derive design from a specification
  - State space of the application and operations on the state space
    - Procedural – VDM-SL, Z, B
    - Object-Oriented – Z++, MooZ, VDM++, Object-Z

# Classification ... (continued)

- Application-oriented
  - Real-time – RTL, RTOZ, TRIO, TRIO++
  - Concurrency – CCS, B, FDR
  - Distributed systems – B
  - Business applications - BTOZ

# Reasoning about a Formal Description

- Usable formal methods provide a variety of techniques for reasoning about specifications and drawing implications.
- The completeness and consistency of a specification can be explored.
  - Does a description imply a system should be in several states simultaneously?
  - Do all legal inputs yield one and only one output?
  - What surprising results, perhaps unintended, can be produced by a system?
- Formal methods support formal verification, the construction of formal proofs that an implementation satisfies a specification.

# Limitations of Formal Methods

- **The Requirements Problem**
  - "You cannot go from the informal to the formal by formal means."
  - In particular, formal methods can prove that an implementation satisfies a formal specification, but they cannot prove that a formal specification captures a user's intuitive informal understanding of a system.
  - In other words, formal methods can be used to verify a system, but not to validate a system.
- One influential field study, [Curtis 88], found that the three most important problems in software development are:
  - The thin spread of application domain knowledge
  - Changes in and conflicts between requirements
  - Communication and coordination problems.
- Successful projects were often successful because of the role of one or two key exceptional designers, who had a deep understanding of the applications domain and could map the applications requirements to computer science concerns.

# Limitations of Formal Methods

## Physical Implementations

- The second major gap between the abstractions of formal methods and concrete reality lies in the nature of any actual physically existing computer.
- Formal methods can verify that an implementation satisfies a specification when run on an idealized abstract machine, but not when run on any physical machine.
- Some of the differences between typical idealized machines and physical machines are necessary for humanly-readable correctness proofs. For instance,
  - an abstract machine might be assumed to have an infinite memory, while every actual machine has some upper limit.
  - Similarly, physical machines cannot implement real numbers, as axiomatically described by mathematicians, while proofs are most simply constructed assuming the existence of mathematically precise real.
- No reason in principle exists why formal methods cannot incorporate these limitations

# Conclusion

A good specification should be

- abstract and complete
- clear and unambiguous
- concise and comprehensible
- easy to maintain and cost-effective

Above all, it should be useful.

Formal specifications cannot replace knowledge. However, we can use an abstract representation as a basis for discussion. The lack of ambiguity makes consensus harder to achieve, but more valuable.

# The Z specification Language

- Pronounced as *zed*
- Developed at Oxford University, England
- Evolved over many years – syntax and semantics changed considerably from the initial version
- Currently being standardized by ISO
- Most popular because of its simplicity!
- Allows informal and formal descriptions mixed together in a specification
- Supported by extensive literature and tool support

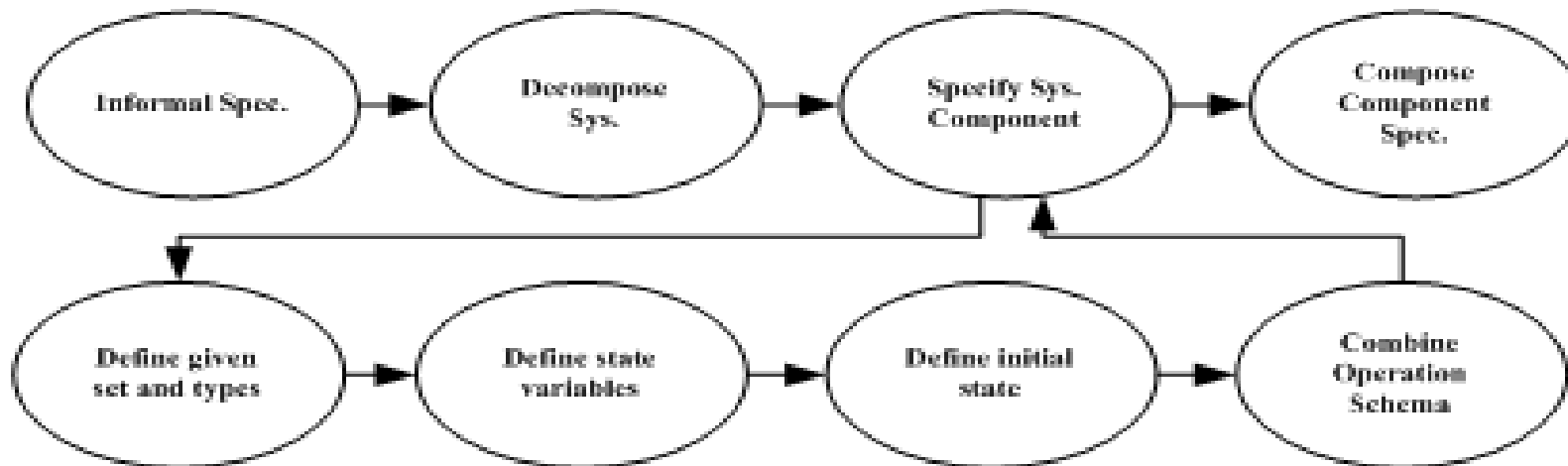
# The Z specification language (cont.)

- Oxford University won Queen's Award twice for demonstrating the industrial application of Z
- Ontario Hydro in Canada was another major industrial application that used Z
- Competitors: VDM-SL and B
- OO Extensions: Z++, ZEST, MooZ, Object-Z

# The Z specification language (cont.)

- Model-based
  - Specifies state space and operations
- Uses set theory and first-order predicate logic (classical or two-valued logic)
- Operations specified using pre and post-conditions
- Uses a schema notation – simple and easy to write and to understand; also uses schema calculus to combine schemas
- Built-in mathematical toolkit – enables users to add new mathematical definitions – extendibility

# Z Specification Process



1. Write Informal specification of system.
2. Decompose a specification into small pieces called *schemas*.
3. Variables are declared and typed in the top part of *schema*
4. A predicate restraining the possible values of the declared variables is given in the bottom part of the *schema*.
5. Operation expressed by relationship between values of variables before, and values after, the operation.
6. Compose component specification.