

1 Introduction

CA648 Formal Programming

Lecturer: Geoff Hamilton

Office: L255

Phone: 5017

Email: hamilton@computing.dcu.ie

WWW: <http://www.computing.dcu.ie/~hamilton>

Course Page: ["/teaching/CA648](#)

Textbook: There is no recommended book for this course. All the necessary reading material will be made available on the course web page as required.

1.1 Formal Methods

Formal Methods

- This course is about the application of mathematics (*set theory and logic*) to specify, design and implement software (or more generally systems) in such a way that the resulting code has been *proved* to be *consistent* with the original specification.
- We will look in particular at the *Event-B* method, in which a specification is a mathematical model of the required behaviour of a system. These specifications are generally abstract.
- These specifications are then transformed through a sequence of formally defined *refinement* steps towards a *concrete* implementation.
- During this process a number of *proof obligations* may have to be *discharged*.

1.2 Conventional Methods

Connection with “Conventional” Methods

- Conventional software development methods usually express the requirements informally:
 - either structured or unstructured English,
 - or using some structured notation: dataflow, entity-relationship diagrams, the unified modelling language (UML), etc.
- Specifications are frequently expressed directly in programming code.
- Design then consists of “fleshing out” the code to produce an implementation.
- In contrast, when using a formal development method, the specification is an abstract description of the requirements, expressing *what* behaviour is required, rather than *how* to produce that behaviour.
- It follows the design phase must effect a radical transformation of the specification in order to obtain executable code.

1.3 Testing

Program Errors

- Programmers are human and make mistakes.
- Many competent programmers spend as much as half their time detecting and correcting errors.
- Commonly used programming languages and tools do little to prevent error.
- Software products are often delivered late, and with a functionality that requires years of evolution to meet original customer requirements.
- The US Department of Commerce in 2002 estimated that the cost to the US economy of avoidable software errors is between 20 and 60 billion dollars every year.

Testing In Traditional Engineering Disciplines

- Consider traditional engineering disciplines, such as electrical engineering, or civil engineering.
- Designs are based on a mathematical theory of materials, components, used in the implementation of bridges, electronic circuits, etc..
- Testing consists of physical testing of the implementation, or a model of the implementation.
- Successful strategy because domains are described by continuous mathematics: if a model conforms for some specific test input, it will also conform for input that is “less than” that input. Hence we need only test for extreme values.
- *This strategy would not work for discrete valued domains.*

Contrast with Testing of Software

- Software executes over discrete domains, and testing usually consists of probing points within that space.
- Thus testing can only confirm conformance of behaviour at specific points.
- Only trivial programs can be tested exhaustively
- Testing is therefore incapable, in general, of demonstrating conformance over the complete application domain.
- Coverage tools can help
- *Thus testing may confirm the presence of bugs, but not their absence.*

1.4 Formal Methods Overview

Levels of Formal Methods

- *Formal Specification* - using mathematical notation to give a precise description of what a program should do.
- *Formal Verification* - using precise rules to mathematically prove that a program satisfies a formal specification.
- *Formal Development (Refinement)* - developing programs in a way that ensures mathematically they meet their formal specifications.
- Formal Methods should be used in conjunction with testing, not as a replacement.

Why Use Formal Methods?

- They can be expensive (though can be applied in varying degrees of effort).
- There is a trade-off between expense and the need for correctness.
- It may be better to have something that works most of the time than nothing at all.
- For some applications, correctness is especially important (e.g. nuclear reactor controllers, car braking systems, fly-by-wire aircraft, software controlled medical equipment).
- Formal proof of correctness provides a way of establishing the absence of bugs when exhaustive testing is impossible.
- This proof of correctness is thus the counterpart of testing in other engineering disciplines (a proof validates behaviour in a complete domain, not simply at a single point).

What Has Been Done With Formal Methods?

- Formal methods have been used in various mission-critical applications, for example:
 - MÉTÉOR - the automatic train operating system for the first driverless metro in the city of Paris (Matra Transport Intl., France)
 - speed control system for the SNCF TGV (GEC Alsthom Transport, France)
 - avionics system (GEC-Marconi Avionics, UK)
 - smart card application (Gemplus, France)
- In some countries the use of formal methods is mandatory for certain critical systems.
- The most famous and significant use of B is for the control system of the Météor line, a new driverless train line of the Paris metro opened in October 1998.

1.5 Formal Methods Papers

Seven Myths of Formal Methods

A. Hall, “Seven Myths of Formal Methods”, IEEE Software 7(5):11–19, September 1990

1. Formal methods can guarantee that software is perfect
2. Formal methods are all about program proving
3. Formal methods are only useful for safety critical systems
4. Formal methods require highly trained mathematicians
5. Formal methods increase the cost of development
6. Formal methods are unacceptable to users
7. Formal methods are not used on real, large scale software

Seven More Myths of Formal Methods

J. Bowen and M. Hinchey, “Seven More Myths of Formal Methods”, IEEE Software 12(4):34–41, July 1995

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

Ten Commandments of Formal Methods

J. Bowen and M. Hinchey, “Ten Commandments of Formal Methods”, IEEE Computer 28(4):56–63, April 1995

1. You shall choose an appropriate notation
2. You shall formalize but not over-formalize
3. You shall estimate costs
4. You shall have a formal methods guru on call
5. You shall not abandon your traditional development methods
6. You shall document sufficiently
7. You shall not compromise your quality standards
8. You shall not be dogmatic
9. You shall test, test, and test again
10. You shall reuse

1.6 Formal Methods Research

The Verified Software Grand Challenge

C.A.R. Hoare, “The Verifying Compiler: A Grand Challenge for Computing Research”, ACM Journal 50(1):63–69, January 2003

- A long-term vision (50 years from now)
- Idea proposed by Tony Hoare
- Final goals: verifying compilers
- Pilot projects
- Worldwide repository of tools, verified code, etc