Formal Methods in software development

a.y. 2017/2018

Prof. Anna Labella

Speaking of formal methods: where Computer Science is situated?

- Between analytical a priori (as mathematics)
- and synthetic a posteriori (as earth sciences) . . .
- Where computer science is?

Gilles Dowek (2013): computer science is an analytical a posteriori science.

Why?

- What we are reasoning about?
- About what is a materialisation of our own thinking
- Software is a materialisation of our thinking
- Software Engineering is the way of better organizing our software using our more abstract tools

The opinion by industry

Microsoft's Research in Software Engineering in Redmond, USA.
"Our mission is to advance the state of the art in SE,
to bring those advances to Microsoft's business, and to take care of those SE technologies that are critical to the company,
but not inherently linked to particular products."

Research in Software Engineering (RiSE)

RiSE understands the importance of logic and
builds powerful inference engines that
help understand complex systems.▪
Tools: Z3 ...
Projects: DKAL LAI L&F M3

Explaining the title: Formal Methods

What is a method? (1)

- Something that gives the means to achieve a certain goal
- Some examples: physics

In physics we introduce measures (mathematical objects) for the phenomena we are going to observe and then mathematical tools (e.g. differential equations) to manipulate them in order to obtain some previsions

What is a method? (2)

- Chemistrians have successfully tried the same way (both are sciences in the strict sense of the term because they use mathematical tools)
- Sociologists didn’t find the correct mathematical representation as yet, i.e. a quantisation of their phenomena that allows mathematical computation (statistics?)

What is a method? (3)

- In computer science, we are still in a relatively primitive stadium: what we usually have is only a formal technique

We express our “facts” in a language and try to manipulate this language in order to make “calculations”.

An example (the first one in c.s.) of working method: compilers

A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code.
 The most common reason for converting source code is to create an executable program.

The name "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language or machine code).

In order to construct a compiler, first the grammar of the source language is defined in the BNF using formal rules.

Hence the mathematical substratum is
formal languages theory

A compiler is likely to perform many or all of the following operations: lexical analysis, preprocessing, parsing, semantic analysis (syntax-directed translation), code generation, and code optimization.

Program faults caused by incorrect compiler behaviour can be very difficult to track down and work around; therefore, compiler implementors invest significant effort to ensure compiler correctness.

What is the use of formal methods?

- Introducing new concepts
- Avoiding errors

Introducing new concepts

- Software development - life cycle
- Costing of the various phases
- Evaluation of the cost of design errors

Avoiding errors

- The errors are not only expensive and difficult to correct
- But can lead to critical situations

- The first flight of the Ariane 5 (Ariane 5 flight 501) held June 4, 1996 failed and the rocket self-destructed 40 seconds after launch because of a malfunction of the control software, created by one of the most famous bug in the history.
- A value in 64-bit floating point, probably the one of the pressure was converted in a 16-bit integer with a sign. This caused a trap (operational error) in the processor: the number was too large to be represented with a 16-bit integer.
- **Efficiency reasons** had pushed designers to disable the control software (written in Ada) on the trap, although other similar conversions in the code were correct.
- This error unleashed a reactions chain that caused the destructive deviation of the rocket because of the enormous aerodynamic forces.
- It was necessary almost a **year and a half** to understand what was the malfunction that led to the destruction of the rocket.

Why formalisation?

- In terms of specific requirements it is useful to understand what actually is needed and
- to check the validity of certain properties that are required (rapid prototyping)

- Suppose you are working for a software company and your task is to write programs which are meant to solve sophisticated problems, or computations.
- Typically, such a project involves an outside customer – a utility company, for example – who has written up an informal description, in plain English, of the real-world task that is at hand. In this case, it could be the development and maintenance of a database of electricity accounts with all the possible applications of that – automated billing, customer service etc. Since the informality of such descriptions may cause ambiguities which eventually could result in serious and expensive design flaws, it is desirable to condense all the requirements of such a project into formal specifications.
- These formal specifications are usually symbolic encodings of real-world constraints into some sort of logic.

Thus, a framework for producing the software could be:

- Convert the informal description R of requirements for an application domain into an 'equivalent' formula ϕ_R of some symbolic logic;
- Write a program P which is meant to realise ϕ_R in the programming environment supplied by your company, or wanted by the particular customer;
- Prove that the program P satisfies the formula ϕ_R .

The next phase of the software development framework involves constructing the program P and after that the last task is to verify that P satisfies ϕ_R. Here again, our framework is oversimplifying what goes on in practice, since often proving that P satisfies its specification ϕ_R goes hand-in-hand with inventing a suitable P. (From Huth Ryan book)

- It is necessary to choose a suitable programming language and, according to its primitives,

- Build a mathematical model

But, beyond formalisation, a proof method

A specification language must:

- Be formal
- Founded on a mathematical theory (semantics)
- Allow proofs of properties
- Enable the development of support tools

This course is intended to give an introduction
to formal methods for the specification,
development and testing of software

It aims to introduce the main formal methodologies,
and show how these are applied in software development.

Topics could change from one year to another one,
has it already happened

I will propose you a logical, though many-faceted,
approach, but other ones could be proposed as well

The problems that we face (1)

- need for formalization of requirements for the production of the software,

- i.e. translate requirements expressed in the current language form in conditions for the development of software

The problems that we face (2)

- the problem of producing a secure software in the sense of absolute reliability: examples of critical situations
- Economy in formalisation

Advantages

- The verification of the properties to meet is made a priori
- This will not be a test by cases, but
- a proof that came with the software product

Difficulties

- Stiffness of any formalization, especially in complex cases
- Acquisition of a new "language ”
- Slow start

An overview

- Which kind of theory?
  - Transition systems
  - Set theory
  - Universal Algebra
  - Lambda-calculus, etc.
- An application field
  - Data processing
  - Real-time systems
  - Protocols, etc.
- A community of use

Specific and general methods

- Shaping the individual aspects of the system software: architectures, interfaces, visible behaviors, algorithms …
- Provide a mathematical context: choices are differed, but often not so “methodological"

Specifying and verifying

- We will deal with the problem of specifying and verifying and of relationships between them

- Though their actual implementation will be left to other courses; e.g.

Automatic Software Verification Methods

Great variety of mathematical theories

- Algebraic Methods (terms are objects)
- Logical Methods (terms are proofs)

Algebraic methods

- Terms are objects in an algebraic structure (denotational semantics)
- We compose and decompose them via operations

Distributed computing

- Processes (operational semantics: abstract machine metaphor)

Logical methods

- Terms are elementary inferences: states and transitions
- We compose them in proofs

The strategy (1)

•From a semi-formal and graphical language as: statecharts and UML to “describe” to a formal language to “speak about”.

•We are looking for a language to model and reason about computer systems. To this purpose we need to improve our logical knowledge.

The strategy (2)

•Maybe in the future the special kinds of logics we are going to expose, will be obsolete, but we are trying to establish a method.

•Already now we had to choose between several possible formalisations.

What is logic?

- In the ancient times a calculus for our way of reasoning;
- maybe more than one way of reasoning: mathematics, retorics, philosophy, theology, ….

- More recently, a calculus to play games

- In our approach:

Logic is the calculus of computation

(Amir Pnueli)

Program of the course

- Temporal Logics and Model checking
- Hoare Logic and verification of programs
- Reactive systems and Hennessy-Milner logic
- Denotational semantics (algebraic approach)

Preliminaries

- Propositional and predicative logics in the form of sequent calculus: from that we will introduce temporal logic
- Order structures: from that we will introduce cpo

- Textbook:

there is not a real textbook; all lessons will be available
on the site of the course
http://twiki.di.uniroma1.it/twiki/view/MFS/WebHome

- A survey can be found in:

"Understanding formal methods" by J.-F. Monin Springer, ©2003
in our library

- But it is more useful to follow some chapters of

"Logic in Computer Science" by Huth Ryan (2nd edition)
freely downloadable at
ftp://ftp.cs.bham.ac.uk/pub/authors/M.D.Ryan/tmp/Anongporn/Ch1+3.pdf

- The rest of the material can be found in a Dropbox special folder

(you will contact me via e.mail in order to have the link)

…..final examination

- a project in collaboration (prof. Mari will explain the modalities at the end of this lesson) +

an interview over a part of the program,

- the interview on a part of the program is alternative to the solution of some exercises performed in the last half an hour of every lesson
- to this purpose, please, everytime bring with you some paper

…..final examination

or

- an interview over all the program