Formal Verification

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2018/2019

What is a (formal) logic?

Logic is defined as the study of the principles of reasoning. One of its branches is symbolic logic, that studies formal logic.

- A formal logic is a language equipped with rules for deducing the truth of one sentence from that of another.
- A logic consists of
  - A *logical language* in which (well-formed) sentences are expressed.
  - A *semantics* that defines the intended interpretation of the symbols and expressions of the logical language.
  - A *proof system* that is a framework of rules for deriving valid judgments.
- Examples: propositional logic, first-order logic, higher-order logic, modal logics, ...

What is a logical language?

A logical language consists of

- *logical symbols* whose interpretations are fixed
- *non-logical symbols* whose interpretations vary

These symbols are combined together to form *well-formed formulas*.

Logic and computer science

- Logic and computer science share a symbiotic relationship
  - Logic provides language and methods for the study of theoretical computer science.
  - Computers provide a concrete setting for the implementation of logic.
- Formal logic makes it possible to calculate consequences at the symbolic level, so computers can be used to automate such symbolic calculations.
- Moreover, logic can be used to model the situations we encounter as computer science professionals, in such a way that we can reason about them formally.

Formal methods

The central problem of formal methods

To be able to guarantee the behaviour of a given computing system following some rigorous approach.

Specification

A specification is a model of a system that contains a description of its desired behaviour – *what* is to be implemented, by opposition to *how*.

Specifications vs implementations problem

– How to obtain, from a specification, an implementation with the same behaviour?

- **Program Extraction**
  - Extract computer programs from constructive proofs, based on the *propositions-as-types principle*.
  - For instance, from a proof of a logical property the Coq system is capable of extracting a program.
- **Program Derivation**
  - Stepwise refinement from specifications to programs (B, VDM, Z)
  - Two approaches to correctness:
    - The refinement steps generate *proof obligations* that must be discharged. Derivations are thus formally verified.
    - The refinement process is itself verified to be correct. The derived programs are then *correct by construction*.

Specifications vs implementations problem

– Given an implementation, how can it be guaranteed that it has the same behaviour as the specification?

Program Verification

- Given a program and a specification, check that the former conforms to the latter.
- This is the only applicable method in many situations.
- Two main approaches:
  - *Deductive Program Verification*
    - A correct and complete form of static checking w.r.t. to a specification, based on a program logic.
  - *Software Model Checking*
    - (safety) properties proved about transition system models extracted from the code.

Course topics

Logic and Proof Systems

- Automatic theorem provers
  - propositional logic; SAT-solvers
  - first-order theories; SMT-solvers
- Proof assistants
  - high-order logic; the Coq proof assistant

Software Verification

- Deductive program verification
  - Hoare logic; VCGen; safety verification; functional verification
  - the Why3 platform for deductive program verification
  - ACSL annotations; the WP plug-in of the Frama-C platform
- Automatic program verification
  - bounded model checking of software; CBMC