Software Verification

Maria João Frade

HASLab - INESC TEC
Dep. Informática, Universidade do Minho

Formal Methods

The central problem of formal methods is to guarantee the behaviour of a given computing system following some rigorous approach.

What is a 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***.

Program Verification

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

- Program verification techniques aim to answer this problem.
- Given a program and a specification, check that the former conforms to the latter.
- In many situations, this is the only applicable method.

Program Verification

Two main approaches:

- **Software Model Checking**
  - Safety properties proved about transition system models extracted from the code.
  - Typically allows only for simple properties, expressed as assertions in the code, but is fully automated.
- **Deductive Program Verification**
  - Based on the use of a program logic and the design-by-contract principle.
  - Gives full guarantees and allows for expressing properties using a rich behaviour specification language, but it is not fully automated.

Software Model Checking

- The basic idea is to determine if a correctness property holds by exhaustively exploring the reachable states of a program.
- If the property does not hold, the model checking algorithm generates a *counterexample*, an execution trace leading to a state in which the property is violated.

State explosion problem

- The state space of software programs is typically too large to be analyzed explicitly.
- To overcome this problem:
  - model checking is often combined with abstraction techniques
  - depth-bounded exploration of the state space — *Bounded Model Checking*

Soundness and Completeness

- A software verifier is **sound** if it reports every property violation. All existing bugs are reported. There are no missing bugs. In other words, if it says the program is correct, then it really is correct.
- A verifier is **complete** if all the violations it reports are indeed errors. No spurious warnings are produced. In other words, if it says the program is incorrect, then it really is incorrect.
- Abstraction techniques introduce false positives, sacrificing completeness. Bounded Model Checking only checks execution paths with size up to a fixed bound, sacrificing soundness. Bugs that require longer paths are missed.

Bounded Model Checking of SW

- The key idea is to encode bounded behaviours of the program that enjoy some given property as a logical formula which is passed to a SAT solver. Models of the formula, if any, describe execution paths leading to a violation of the property.
- The properties to be established are assertions on the program state, included in the program through the use of assert statements.
- This technique explores program behaviour exhaustively, but only up to a given depth. Bugs that require longer paths are missed. Nevertheless, the technique is successful, as many bugs have been identified that would otherwise have gone unnoticed.
- We will work with **CBMC** — a Bounded Model Checker for C programs.

Deductive Verification

- A sound and complete form of static checking w.r.t. to a specification, based on a program logic and the design-by-contract principle.
- It is the user's responsibility to provide contracts and other information required for verification to proceed, such as loop invariants.
- We will work with **`Frama-C`** a platform for static analysis of C code, in particular, with the **WP plugin** based on Hoare logic and weakest precondition calculus.