LLVM-based Software Model Checking

Petr Ročkai

Masaryk University
Czech Republic

DTEDI
March 16, 2011

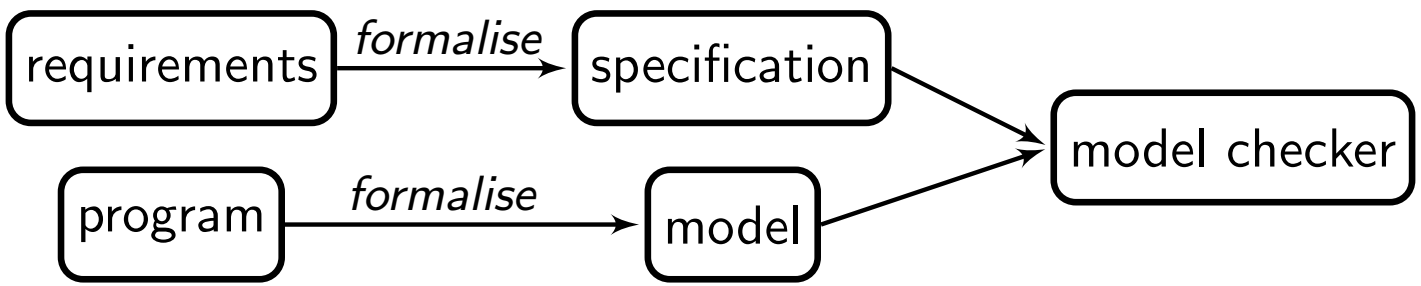
Model Checking

Validation and Verification

- Inevitable part of SW design process.
- Testing is good, but incomplete.
- Automated formal verification is an option.

Model Checking

- *Proves* properties of discrete systems.
- Push-button procedure, for formalised inputs.



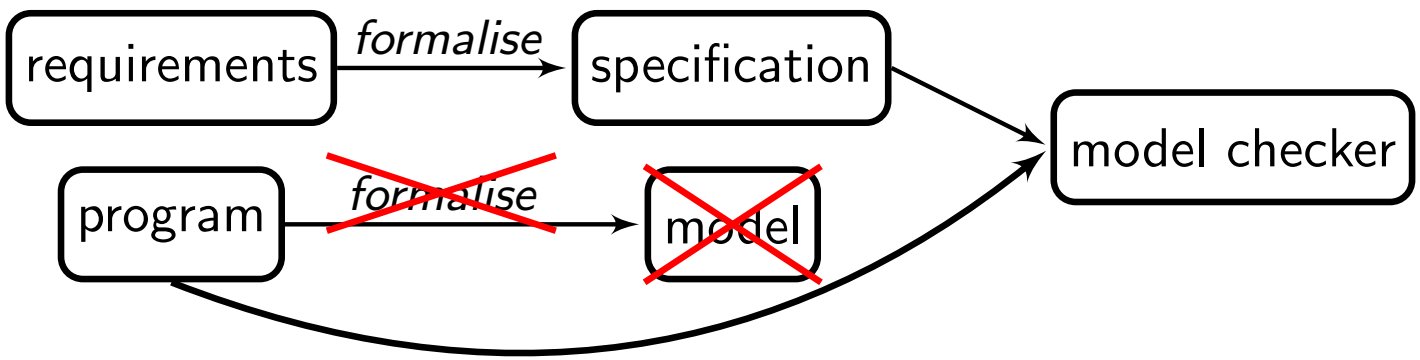
Obtaining Models

The Problem

- The step to formalise the inputs (the model and the specification) can still be expensive.

Solutions

- Model extraction
- **Direct model checking of software**



Model Checking Software

- The model checker directly interprets the source code of the application:
- as a consequence, the expensive modelling step is either omitted or vastly reduced.

Problems

- The resulting state space can be extremely large,
- the model checker needs intimate knowledge of the target programming language.

Problem 1

Extremely large state spaces

Solution

- **DiVinE**, a distributed-memory model checker,
- with good support for state space reductions.

Challenges

- We still need LLVM-specific reductions...
- some of these need to be invented first!

Problem 2

Model checker needs knowledge of target language

Solution

- **LLVM**, a language-agnostic compiler framework,
- with a well-specified intermediate representation.

Challenges

- Model checker needs to trap into the program.
- Thread-level parallelism needs to be tackled.

Case study

Logical Volume Manager

- a component in the storage subsystem
- of general-purpose server operating systems (RHEL, SLES &c.)

Hard to verify by testing:

- concurrent operations & locking,
- internal parallelism,
- cluster coordination.

A good candidate for **model checking**.

Conclusions

We propose...

- A new model-checking backend for **DiVinE**,
- based on **LLVM**,
- augmented with additional reduction techniques.

Goal Summary

- Reduce costs of deploying a rigorous, formal verification method (model checking) in the context of software verification.
- Ultimately, improve quality of new and existing software.