Verification of Parallel C++ with DIVINE

Vladimír Štill

Masaryk University

Brno, Czech Republic

27th April 2016

Introduction

- Why should we care about verification of (parallel) programs?
- What is DIVINE and how it can help?
- What is my work?

Is Testing Sufficient?

it is important to check that programs do what they are supposed to

- testing is important part of software development
- it can take up to 75 % of software development time

Is Testing Sufficient?

it is important to check that programs do what they are supposed to

- testing is important part of software development
- it can take up to 75 % of software development time

however, testing has its downsides

- it cannot proof absence of bugs
- **not very efficient in problem discovery for parallel programs**

Parallel Programs

parallel programs are much harder to think about and debug

- the timing of the execution can influence results
  - *two threads writing to the same memory location*
- results for the same input can differ
- some results can be much less probable

Parallel Programs

parallel programs are much harder to think about and debug

- the timing of the execution can influence results
  - *two threads writing to the same memory location*
- results for the same input can differ
- some results can be much less probable

$\Rightarrow$ testing is not deterministic

- might not discover a bug
- test cannot be repeated with guarantee of the same results
- hard to debug from failed test

Explicit State Model Checking

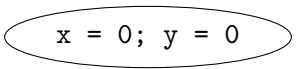
a technique useful for verification of parallel programs

- explores all meaningful interleavings of the program
- can provide deterministic testing procedure
- and prove absence of bugs
- can be very resource consuming

Explicit State Model Checking

a technique useful for verification of parallel programs

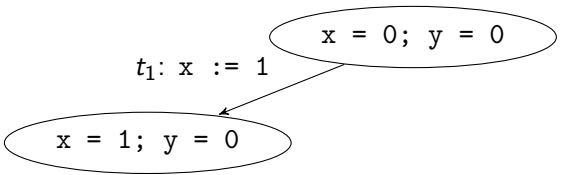
- explores all meaningful interleavings of the program
- can provide deterministic testing procedure
- and prove absence of bugs
- can be very resource consuming



Explicit State Model Checking

a technique useful for verification of parallel programs

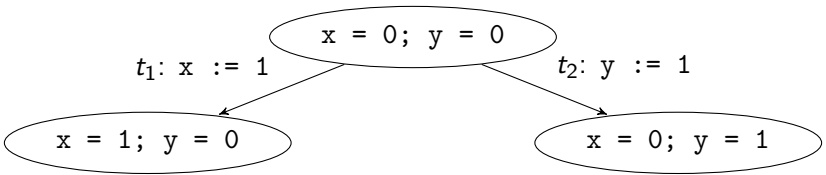
- explores all meaningful interleavings of the program
- can provide deterministic testing procedure
- and prove absence of bugs
- can be very resource consuming



Explicit State Model Checking

a technique useful for verification of parallel programs

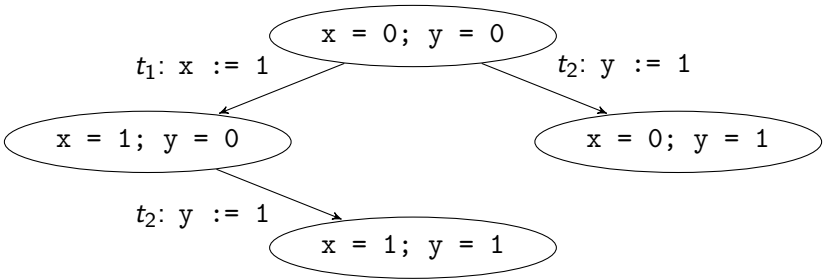
- explores all meaningful interleavings of the program
- can provide deterministic testing procedure
- and prove absence of bugs
- can be very resource consuming



Explicit State Model Checking

a technique useful for verification of parallel programs

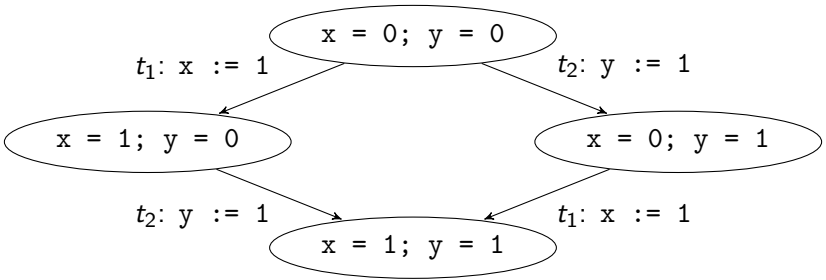
- explores all meaningful interleavings of the program
- can provide deterministic testing procedure
- and prove absence of bugs
- can be very resource consuming



Explicit State Model Checking

a technique useful for verification of parallel programs

- explores all meaningful interleavings of the program
- can provide deterministic testing procedure
- and prove absence of bugs
- can be very resource consuming



Model Checking in Practice?

traditionally, the program has to be translated to a modelling language

- extra effort (usually manual translation)
- not everything can be translated exactly
- translation can introduce or hide errors
- still useful in critical systems (modelling before development)

Model Checking in Practice?

traditionally, the program has to be translated to a modelling language

- extra effort (usually manual translation)
- not everything can be translated exactly
- translation can introduce or hide errors
- still useful in critical systems (modelling before development)

solution: verify code in some programming language directly!

- program is verified directly, no need to model it
- should not limit the programmer to a subset of the language
- even more computationally expensive

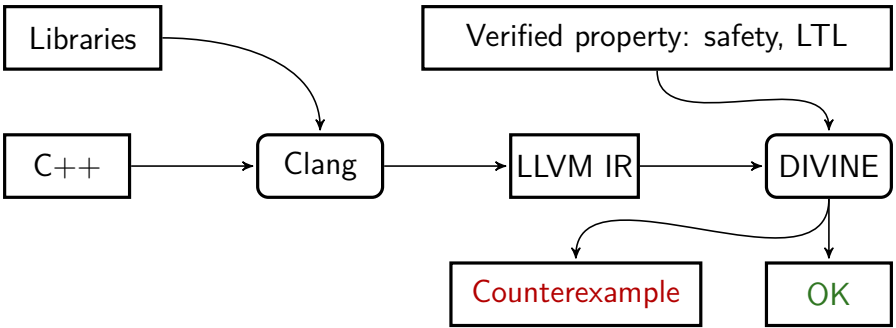
- full support for C and C++ language features
- supports most of the standard C/C++ library and the `pthread` threading library
- effective verification using many reduction techniques
- uses the LLVM intermediate representation as an input

- full support for C and C++ language features
- supports most of the standard C/C++ library and the `pthread` threading library
- effective verification using many reduction techniques
- uses the LLVM intermediate representation as an input

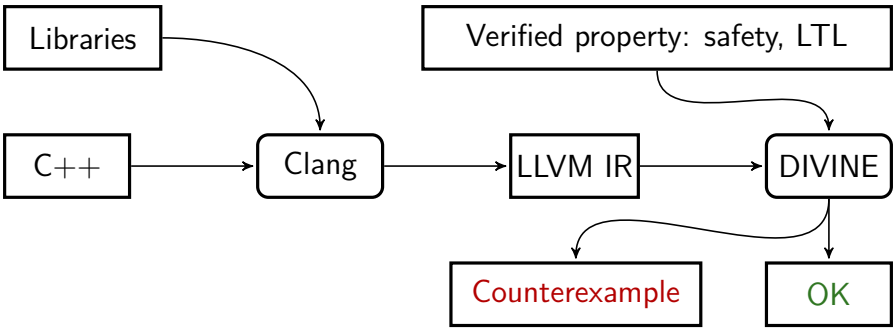
we aim at

- practical usability for unit testing of parallel programs
- and therefore adoption by programmers
- in future also support for programs with inputs

DIVINE Verification Workflow

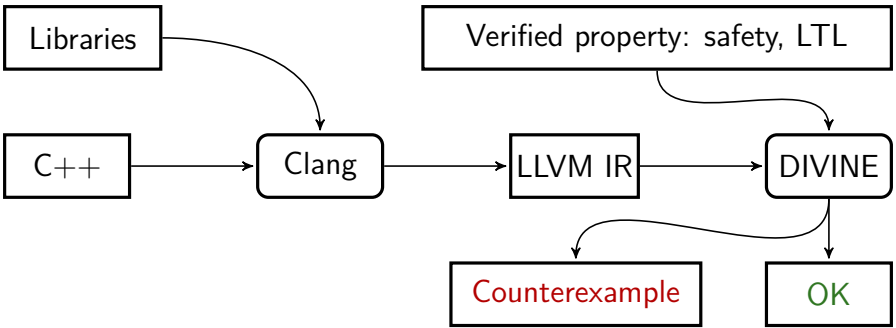


DIVINE Verification Workflow



safety property: assertion safety, memory safety (dereference, array bounds, memory leaks), use of uninitialized memory, deadlocks, ...

DIVINE Verification Workflow



safety property: assertion safety, memory safety (dereference, array bounds, memory leaks), use of uninitialized memory, deadlocks, ...

LTL property: changes in time, i.e.: "after action A occurs, B must happen repeatedly"

My Work on DIVINE

during bachelor & masters studies

- memory-efficient representation of program states
- verification with relaxed memory models
- general maintenance

My Work on DIVINE

during bachelor & masters studies

- memory-efficient representation of program states
- verification with relaxed memory models
- general maintenance

Ph.D. research topic

- optimizations and transformations of the LLVM intermediate representation
  - to allow faster and more efficient verification
  - to be able to verify more properties
- general usability of DIVINE

Summary

- parallel programming is hard, testing not very efficient
- model checking can help in this regards
- C/C++ code can be verified directly

Summary

- parallel programming is hard, testing not very efficient
- model checking can help in this regards
- C/C++ code can be verified directly

Thank you!