Formal Verification of C/C++ Programs

Vladimír Štill

ParaDiSe
Parallel & Distributed Systems Laboratory

Masaryk University

Brno, Czech Republic

18th March 2016

The Current Situation

My Work on DIVINE

- compression of the state space
  - reduces memory requirements for LLVM verification, roughly $100 - 500\times$ for reasonably sized programs (efficiency grows with program size)
  - bachelor's thesis, published in SEFM 2015

My Work on DIVINE

- compression of the state space
  - reduces memory requirements for LLVM verification, roughly $100 - 500\times$ for reasonably sized programs (efficiency grows with program size)
  - bachelor's thesis, published in SEFM 2015
- export of explicit state space from DIVINE
  - useful for chaining with other tools
  - case study for probabilistic verification to appear in ACM SAC 2016

My Work on DIVINE

- compression of the state space
  - reduces memory requirements for LLVM verification, roughly $100 - 500\times$ for reasonably sized programs (efficiency grows with program size)
  - bachelor's thesis, published in SEFM 2015
- export of explicit state space from DIVINE
  - useful for chaining with other tools
  - case study for probabilistic verification to appear in ACM SAC 2016
- verification under more realistic memory models
  - verification closer to behaviour of real-world memory hierarchies
  - master's thesis, preliminary version in MEMICS 2015, extended version submitted for publication

My Work on DIVINE

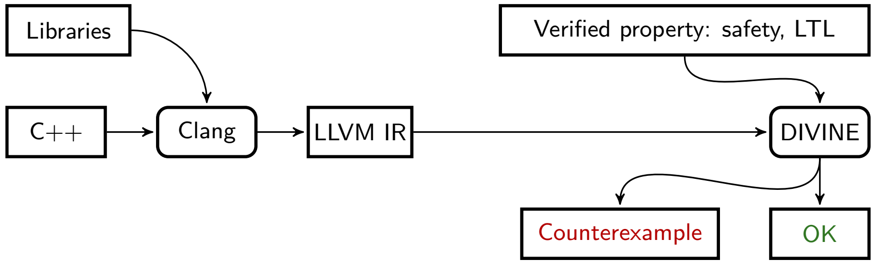
- compression of the state space
  - reduces memory requirements for LLVM verification, roughly $100 - 500\times$ for reasonably sized programs (efficiency grows with program size)
  - bachelor's thesis, published in SEFM 2015
- export of explicit state space from DIVINE
  - useful for chaining with other tools
  - case study for probabilistic verification to appear in ACM SAC 2016
- verification under more realistic memory models
  - verification closer to behaviour of real-world memory hierarchies
  - master's thesis, preliminary version in MEMICS 2015, extended version submitted for publication
- extended and fixed state space reductions
  - up to $3\times$ extra reduction
  - master's thesis

My Work on DIVINE

- compression of the state space
  - reduces memory requirements for LLVM verification, roughly $100 - 500\times$ for reasonably sized programs (efficiency grows with program size)
  - bachelor's thesis, published in SEFM 2015
- export of explicit state space from DIVINE
  - useful for chaining with other tools
  - case study for probabilistic verification to appear in ACM SAC 2016
- verification under more realistic memory models
  - verification closer to behaviour of real-world memory hierarchies
  - master's thesis, preliminary version in MEMICS 2015, extended version submitted for publication
- extended and fixed state space reductions
  - up to $3\times$ extra reduction
  - master's thesis
- code maintenance

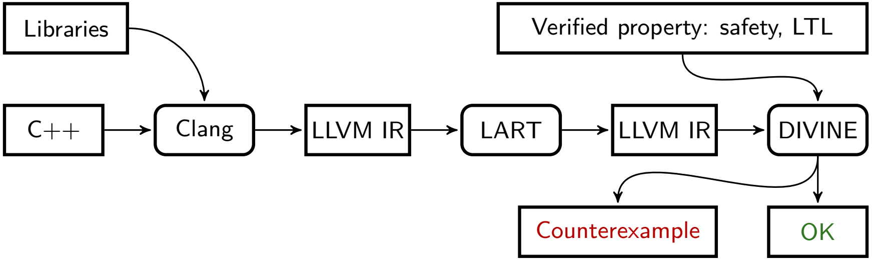
LLVM Transformations

- LLVM IR can be easily transformed before the verification
- can be used to extend model checker's abilities, reduce state space



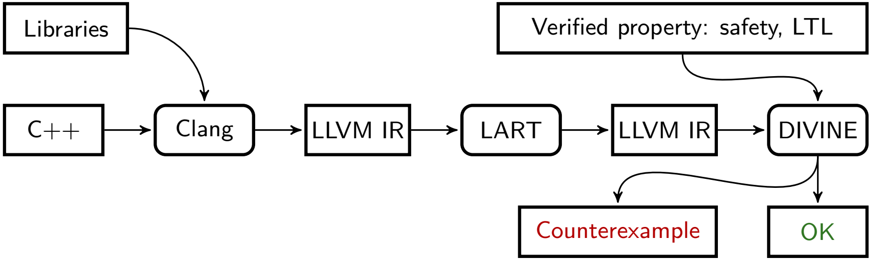
LLVM Transformations

- LLVM IR can be easily transformed before the verification
- can be used to extend model checker's abilities, reduce state space



LLVM Transformations

- LLVM IR can be easily transformed before the verification
- can be used to extend model checker's abilities, reduce state space



- case study: verification of weak memory models through LLVM transformation

Weak Memory Models

- in CPU a write performed by one thread need not be visible to other thread immediately
- writes can be reordered – with reads or with reads and writes
- verifiers often omit this

Weak Memory Models

- in CPU a write performed by one thread need not be visible to other thread immediately
- writes can be reordered – with reads or with reads and writes
- verifiers often omit this

Solution

- the program is instrumented to simulate delayed/reordered writes
- adds more nondeterminism to the program
- LLVM transformation

Plans

Plans – Overview

Long Term

- improve practical usability of model checking for development of parallel programs
- explore the usage of static analysis for pre-processing of programs for DIVINE

Short Term (this year)

- more robust compilation of programs for DIVINE
- register allocation for LLVM
- verification of programs with inputs using SMT (merge of SymDIVINE into DIVINE)

Compilation of Programs for DIVINE

- currently, DIVINE facilitates a simple wrapper over clang for compilation
    - together with tweaked LLVM-based linker
- DIVINE has to provide own implementation of C/C++/`thread`/... libraries
- system configuration and even system headers can leak into DIVINE compilation
- hard to integrate into nontrivial build processes (makefiles, cmake,...)

Compilation of Programs for DIVINE

- currently, DIVINE facilitates a simple wrapper over clang for compilation
  - together with tweaked LLVM-based linker
- DIVINE has to provide own implementation of C/C++/`thread`/... libraries
- system configuration and even system headers can leak into DIVINE compilation
- hard to integrate into nontrivial build processes (makefiles, cmake,...)

Solution

- an isolated environment which can access only user-provided sources and DIVINE libraries
- DIVINE compiler which can be used as a drop-in replacement for GCC/clang
- ideally it would produce both LLVM bitcode for DIVINE and ELF binary
  - allow build processes which feature code generating programs

Register Allocation for LLVM

- LLVM uses Static Single Assignment (registers not reused)
- wastes memory in DIVINE
- can prevent state merging (e.g. in optimized busy-waiting cycles)

Register Allocation for LLVM

- LLVM uses Static Single Assignment (registers not reused)
- wastes memory in DIVINE
- can prevent state merging (e.g. in optimized busy-waiting cycles)

Solution

- allocate registers into slots, reuse slots
- differs from register allocation in code generator of a compiler
    - the number of registers is not fixed
    - should consider program semantics

Verification of Programs with Inputs

- programs with inputs cannot be fully verified by DIVINE
- SymDIVINE can do this for simple programs
    - a proof-of-concept tool for verification of LLVM programs with inputs

Verification of Programs with Inputs

- programs with inputs cannot be fully verified by DIVINE
- SymDIVINE can do this for simple programs
    - a proof-of-concept tool for verification of LLVM programs with inputs

Solution

- merge SymDIVINE into DIVINE using an LLVM transformation
- the program is to be changed so that it manipulates (parts of) data symbolically
- this hybrid program is then executed by DIVINE which uses special algorithm to explore state space of such programs

Plans – Overview

Long Term

- improve practical usability of model checking for development of parallel programs
- explore the usage of static analysis for pre-processing of programs for DIVINE

Short Term (this year)

- more robust compilation of programs for DIVINE
- register allocation for LLVM
- verification of programs with inputs using SMT (merge of SymDIVINE into DIVINE)

Thanks for your attention!