DIVINE

Vladimír Štill

ParaDiSe
Parallel & Distributed
Systems Laboratory

Masaryk University

Brno, Czech Republic

11th March 2015

DIVINE

- explicit-state model checker for verification of parallel programs
    - main focus on C++/LLVM
    - also many other inputs: DVE, CESMI, timed automata
- verifies safety and LTL properties
    - specification depends on the input formalism
- parallel (and distributed) verification
- reduction strategies
- `https://divine.fi.muni.cz`

DIVINE: Input Formalisms

LLVM

- for verification of C and C++
- more later

DVE

- simple input formalism for communicating processes
- each process has finitely many states
- channels, guards for communication and synchronization

CESMI

- user-implemented, compiled models, using C API

Timed automata

- using UPPALL formalism

DIVINE: State Space Output

DIVINE can either run a verification algorithm over the state space, *or calculate and save the state space*

DESS (DIVINE Explicit State Space)

- binary format of explicit state space
- DIVINE can materialize any state space into DESS
- state flags (assertion violation, atomic propositions) can be read from DESS

DIVINE: LLVM as an Input Language

- used for verification of C and C++ programs
- since DIVINE 3
- C and C++ library, C++ exceptions support (since 3.2)
- threads through `pthreads` or C++11
- state space size reductions
- safety properties: assertion safety, memory safety, uninitialized variables tracking, `pthreads` deadlock detection
- rudimentary LTL support

DIVINE: Input Formalisms

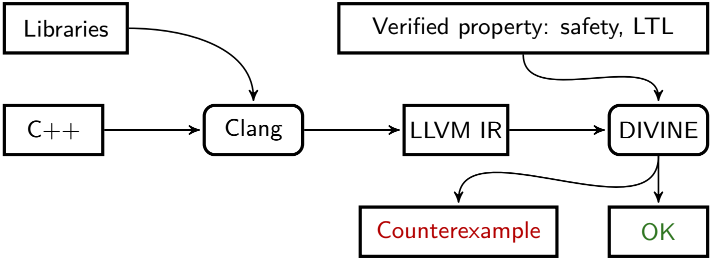
LTL in LLVM

- explicitly activated atomic propositions
- hold only in the state they were raised
    - two APs cannot hold at the same time
- better support is planned in DIVINE 4

```
#include <divine.h>
enum APs { c1in, c1out, c2in, c2out };
LTL(exclusion,
    G( (c1in -> (!c2in W c1out))
      && (c2in -> (!c1in W c2out))) );

void critical1() { AP( c1in ); AP( c1out ); }
void critical2() { AP( c2in ); AP( c2out ); }
```

DIVINE Verification Workflow



```
divine compile --lib # needed only once
divine compile --pre=. test.cpp --cflags="-std=c++11"
divine info test.bc # list properties
divine verify test.bc --compress --display-counterex
```

C++/LLVM as a Modelling Language

LLVM interpreter in DIVINE supports several intrinsic functions (as of DIVINE 3.3.2)

- `__divine_choice( int n )` splits state space into `n` copies, in each returns a number from $[0, n)$
- `__divine_interrupt_mask()` starts an atomic section
    - `__divine_interrupt` should be called before it
    - atomic sections ends when the function which called `__divine_interrupt_mask` returns
    - everything from the call until the atomic section ends is uninterruptible (one edge in the state space)
- `__divine_interrupt_unmask` immediately ends the atomic section
    - should be called only in functions which called `__divine_interrupt_mask`
- `__divine_problem` report a problem to the interpreter

C++/LLVM as a Modelling Language

```
void *malloc( size_t size ) {
    if ( __divine_choice( 2 ) )
        return __divine_malloc( size );
    return NULL;
}
```

C++/LLVM as a Modelling Language

```
#include <divine/problem.h>
#include <divine.h>
struct Mutex {
    void unlock() {
        __divine_interrupt(); __divine_interrupt_mask();
        if ( _locktid == 0 )
            __divine_problem( Other, "mutex not locked" );
        _locktid = 0;
    }
    void lock() {
        __divine_interrupt(); __divine_interrupt_mask();
        while ( _locktid ) {
            if ( _locktid == __divine_get_tid() + 1 )
                __divine_problem( Other, "mutex re-locked" );
            __divine_interrupt_unmask(); // allow other threads to run
            __divine_interrupt_mask();
        }
        _locktid = __divine_get_tid() + 1;
    }
    private: int _locktid;
};
```

Demo I

LLVM Transformations

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

Weak Memory Model Verification

- more realistic memory access
  - in CPUs, write from one thread need not be visible by other threads immediately

-
  ```
  divine compile --pre=. test.cpp
  lart test.bc test-tso.bc weakmem:tso
  divine verify test-tso.bc
  ```
- in development
- `https://divine.fi.muni.cz/2016/weakmem/`

Demo II: Memory Models