Heavy-Duty Program Analysis with DIVINE

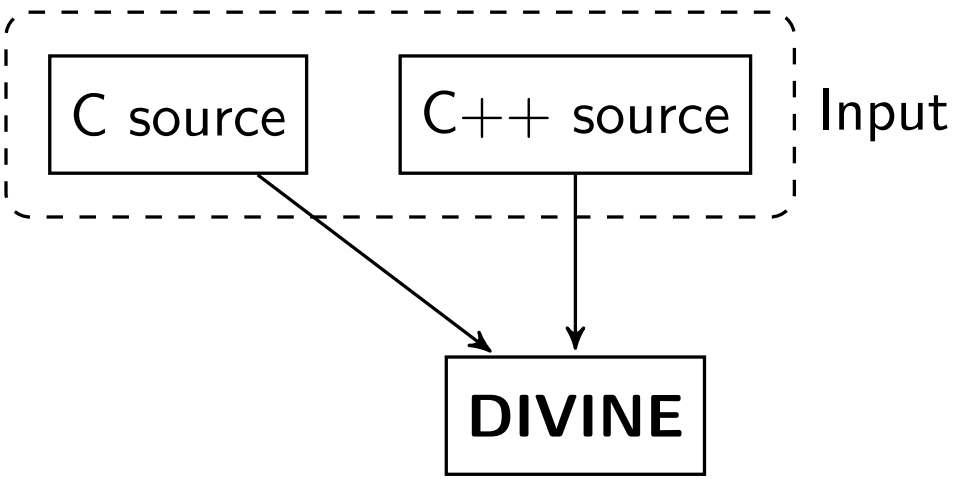
Vladimír Štill

Masaryk University

Brno, Czech Republic

23rd January 2020

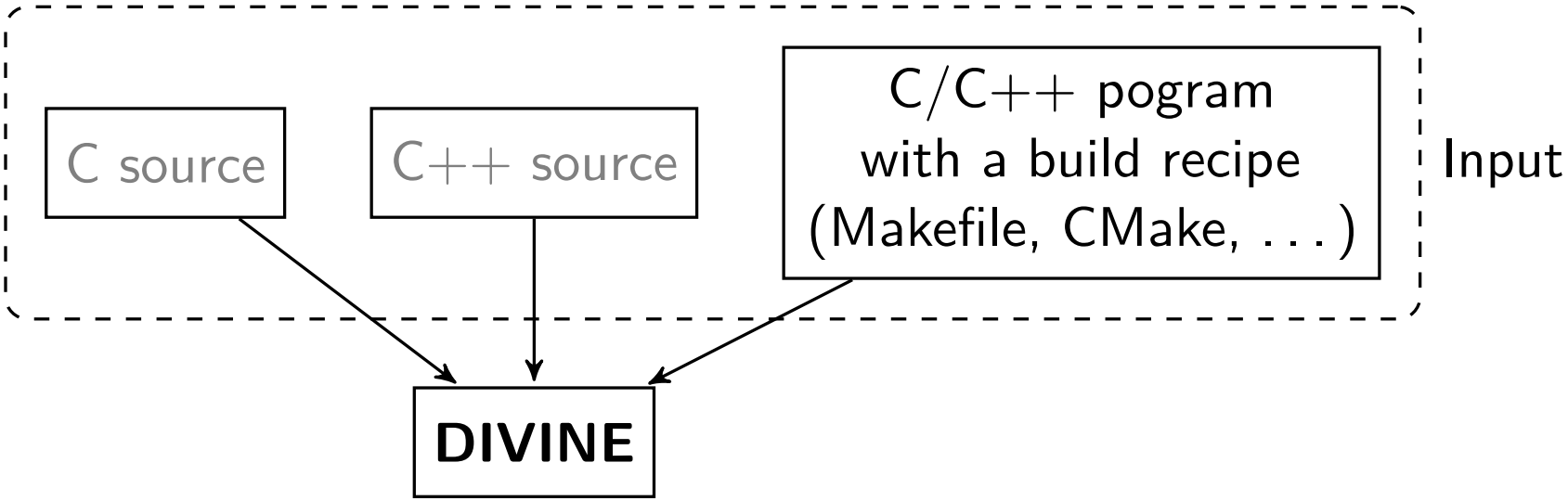
What is DIVINE?



DIVINE can find hard to discover problems in C and C++ programs.

```
$ divine check program.cpp
```

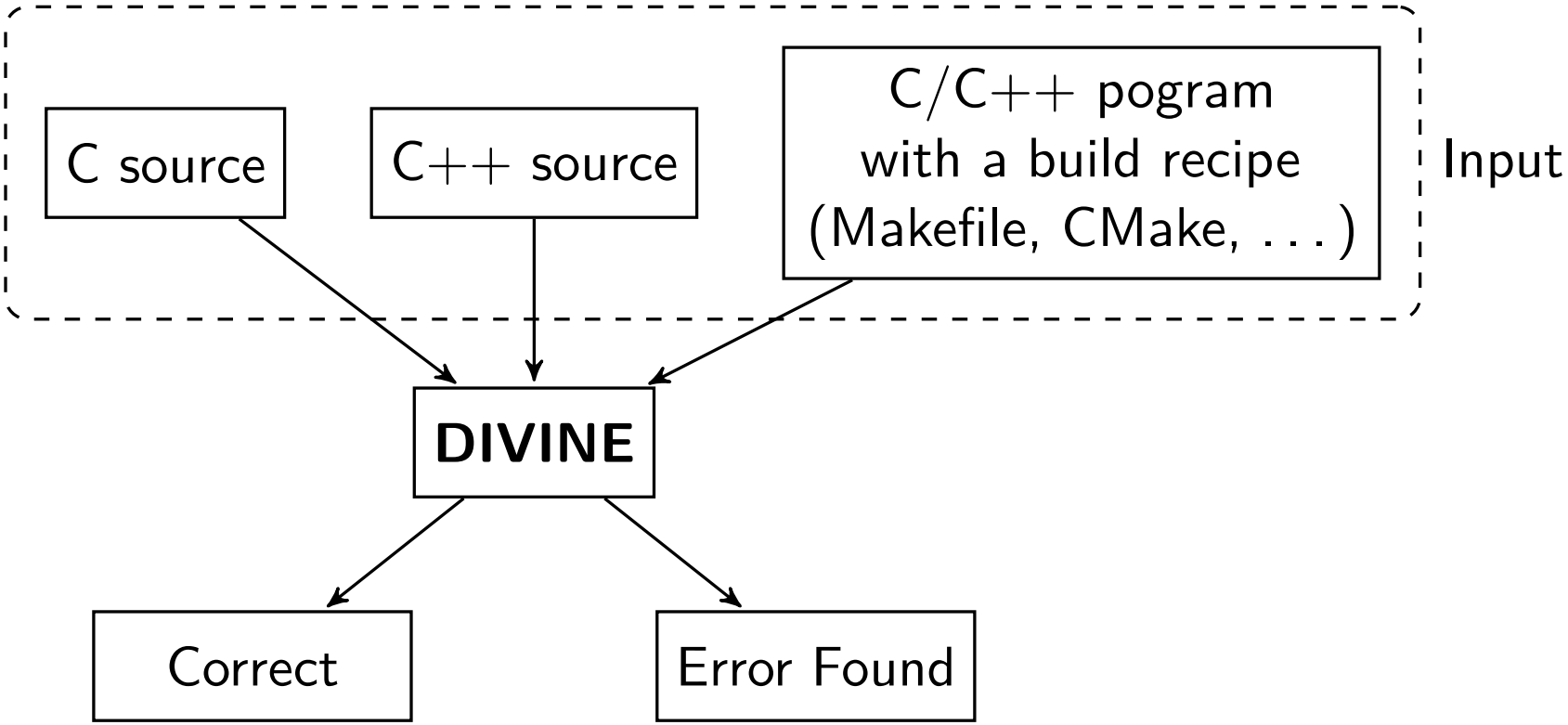
What is DIVINE?



It can process single files, or we can use a replacement compiler to compile larger programs so they can be processed by DIVINE.

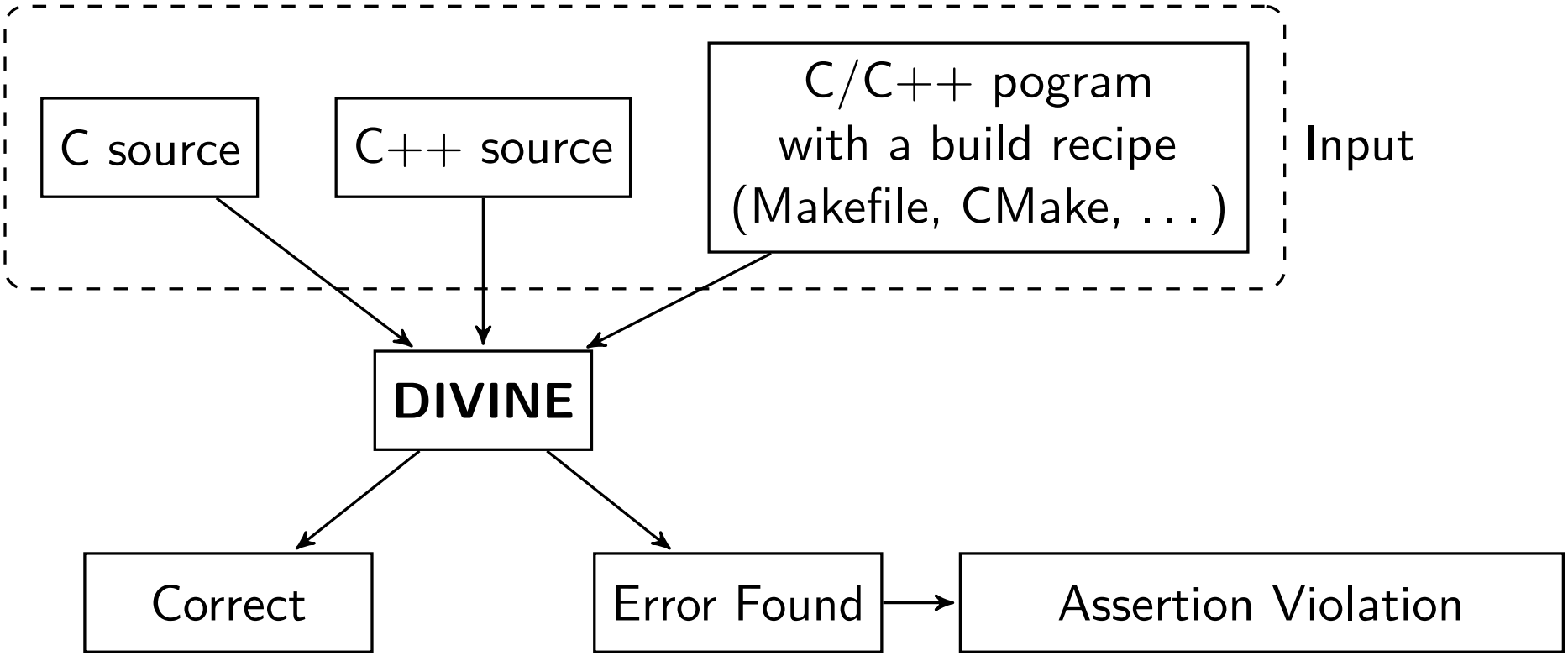
```
$ make CC=divcc
$ divine check build/program
```

What is DIVINE?



Given enough resources, DIVINE will produce result.
If an error is found, DIVINE will produce a report.

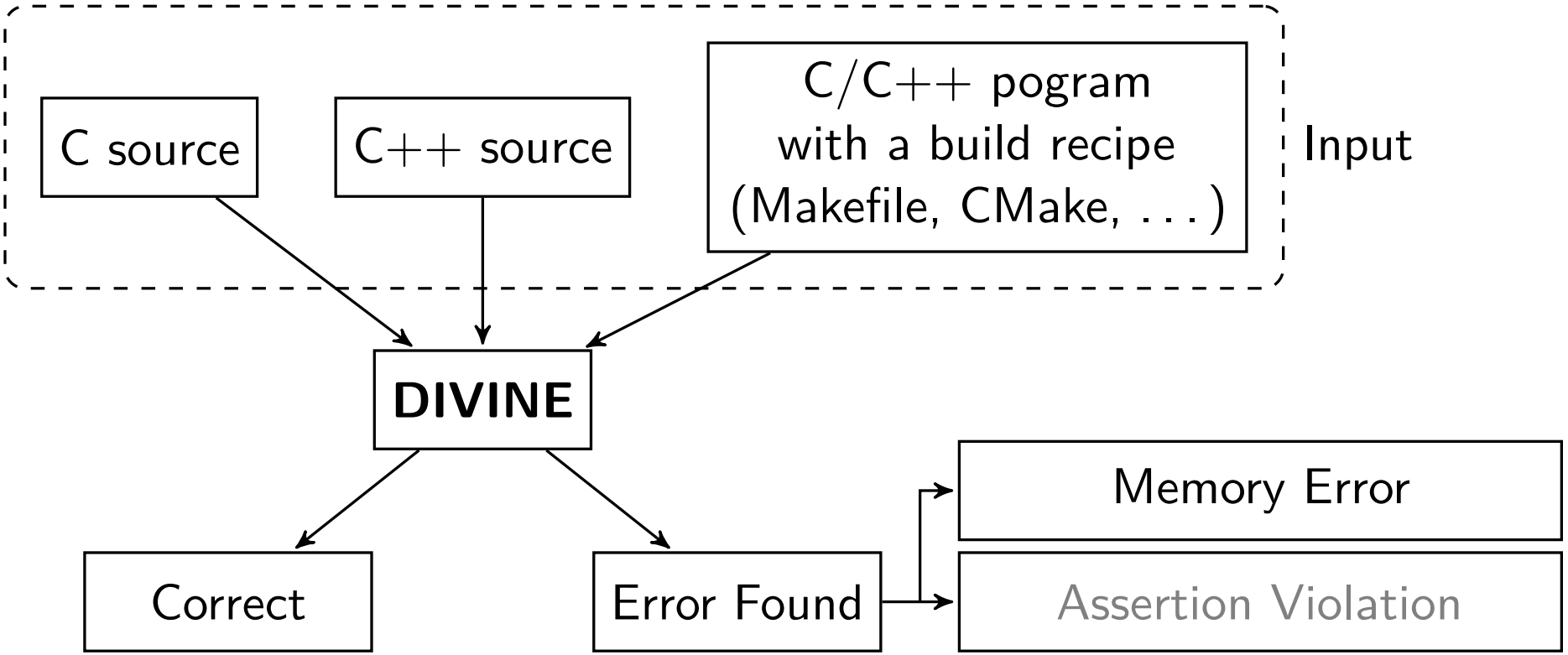
What is DIVINE?




```
int foo( int x ) { assert( x > 0 ); /* ...  */ }
int main() {
    int x = input();
    if ( x >= 0 ) { foo( x ); }
}
```

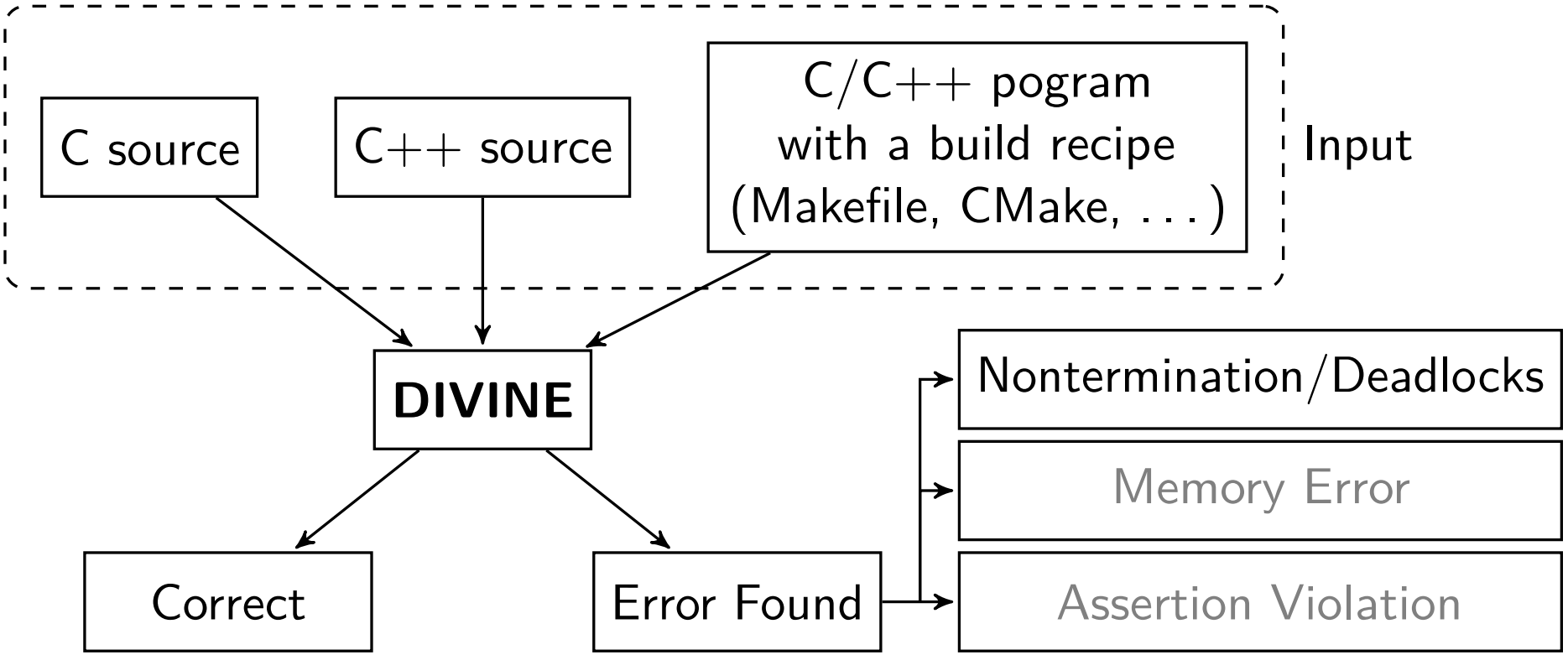
there can be *input* variables with arbitrary/unspecified values

What is DIVINE?




```
int main() {
    int x = 100;
    int array[100];
    array[x] = 42;
}
```

What is DIVINE?




```
void thread1() {
    while ( x != 0 ) { /* wait */ }
    /* ...  */
}
void thread2() { x = 42; }
```

DIVINE explores all possible runs

Before Verification I

1. obtain source code of the program to be verified

Before Verification I

1. obtain source code of the program to be verified
   - including dependencies (except for the basics – C and C++ standard libraries)

Before Verification I

1. obtain source code of the program to be verified
   - including dependencies (except for the basics – C and C++ standard libraries)
   - must be buildable by clang

Before Verification I

1. obtain source code of the program to be verified
   - including dependencies (except for the basics – C and C++ standard libraries)
   - must be buildable by clang
   - no assembly components or inline assembly

Before Verification II

2. obtain tests

Before Verification II

2. obtain tests
   - ideally, the program comes with unit tests usable for verification

Before Verification II

2. obtain tests
   - ideally, the program comes with unit tests usable for verification
   - not all tests work (concurrency stress tests, tests dependent on timing)

Before Verification II

2. obtain tests
   - ideally, the program comes with unit tests usable for verification
   - not all tests work (concurrency stress tests, tests dependent on timing)
   - DIVINE-specific tests can use input (unspecified) values to check a set of inputs at once

Before Verification II

2. obtain tests

- ideally, the program comes with unit tests usable for verification
- not all tests work (concurrency stress tests, tests dependent on timing)
- DIVINE-specific tests can use input (unspecified) values to check a set of inputs at once
- it is often easier to write concurrency tests for DIVINE than “normal” concurrency tests

Before Verification II

2. obtain tests

- ideally, the program comes with unit tests usable for verification
- not all tests work (concurrency stress tests, tests dependent on timing)
- DIVINE-specific tests can use input (unspecified) values to check a set of inputs at once
- it is often easier to write concurrency tests for DIVINE than "normal" concurrency tests
- concurrency tests in DIVINE are *deterministic* – they cannot sometimes fail and sometimes succeed
  - $\rightarrow$ they can be much smaller than stress tests

Before Verification II

2. obtain tests

- ideally, the program comes with unit tests usable for verification
- not all tests work (concurrency stress tests, tests dependent on timing)
- DIVINE-specific tests can use input (unspecified) values to check a set of inputs at once
- it is often easier to write concurrency tests for DIVINE than "normal" concurrency tests
- concurrency tests in DIVINE are *deterministic* – they cannot sometimes fail and sometimes succeed
  - $\rightarrow$ they can be much smaller than stress tests
  - DIVINE explores all possible ways the threads can interleave

Verification & After

- verification can be resource intensive
  - due to parallelism
  - due to inputs/unspecified values
  - due to memory tracking

Verification & After

- verification can be resource intensive
  - due to parallelism
  - due to inputs/unspecified values
  - due to memory tracking
- if an error is found, it can be analysed using an interactive simulator
  - a debugger-like tool integrated in DIVINE

Execution Environment

many programs communicate with other programs using file system or network

- DIVINE has support for a large part of the POSIX API

Execution Environment

many programs communicate with other programs using file system or network

- DIVINE has support for a large part of the POSIX API
- can be simulated/modelled (arbitrary input, mock clients/servers)

Execution Environment

many programs communicate with other programs using file system or network

- DIVINE has support for a large part of the POSIX API
- can be simulated/modelled (arbitrary input, mock clients/servers)
- or captured and replayed
  1. run the program in a way it can communicate and capture the communication
  2. use the capture in verification

Execution Environment

many programs communicate with other programs using file system or network

- DIVINE has support for a large part of the POSIX API
- can be simulated/modelled (arbitrary input, mock clients/servers)
- or captured and replayed
  1. run the program in a way it can communicate and capture the communication
  2. use the capture in verification
     - only works if the program does not encounter need for further communication during verification run

Execution Environment

many programs communicate with other programs using file system or network

- DIVINE has support for a large part of the POSIX API
- can be simulated/modelled (arbitrary input, mock clients/servers)
- or captured and replayed
  1. run the program in a way it can communicate and capture the communication
  2. use the capture in verification
     - only works if the program does not encounter need for further communication during verification run
- capture + replay can also make debugging these programs easier

Work in Progress

Verification of Binaries

- we are working on a way to lift compiled binaries so they can be analysed by DIVINE

Work in Progress

Verification of Binaries

- we are working on a way to lift compiled binaries so they can be analysed by DIVINE
- would allow verification of programs without available source code

Work in Progress

Verification of Binaries

- we are working on a way to lift compiled binaries so they can be analysed by DIVINE
- would allow verification of programs without available source code
- some information is lost in binaries – stack variable boundaries, variable names, ...

Work in Progress

Verification of Binaries

- we are working on a way to lift compiled binaries so they can be analysed by DIVINE
- would allow verification of programs without available source code
- some information is lost in binaries – stack variable boundaries, variable names, ...

Abstractions

- we are working on different ways to abstract data

Work in Progress

Verification of Binaries

- we are working on a way to lift compiled binaries so they can be analysed by DIVINE
- would allow verification of programs without available source code
- some information is lost in binaries – stack variable boundaries, variable names, ...

Abstractions

- we are working on different ways to abstract data
- including lossy/imprecise abstractions which are faster and can still provide some guarantees