# The Application of CDM

Zhaoguo Wang

# A Quick Recap

## 1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

## 1.2 First Order Logic

Step 1. Convert it into first order logic formula.

Step 2. Ask the computer to solve the formula.

## 1.3 Auto-active Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants

PySAT

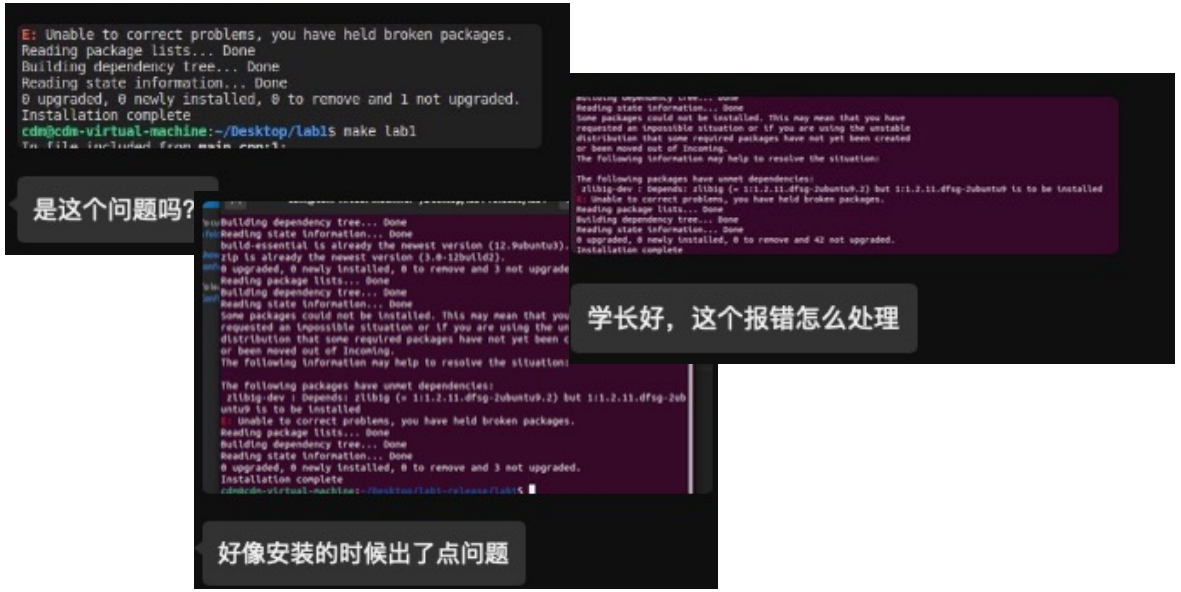
Z3

Dafny

# Outline – This Lecture

- The application of SAT solvers
- The application of SMT solvers
- All cases!
- Bonus lab

# Lab1: A Frequently Asked Question




Unsatisfiable

```
The following packages have unmet dependencies:
 zlib1g-dev : Depends: zlib1g (= 1:1.2.11.dfsg-2ubuntu9.2) but 1:1.2.11.dfsg-2ubuntu9 is to be install
ed
E: Unable to correct problems, you have held broken packages.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
Installation complete
```

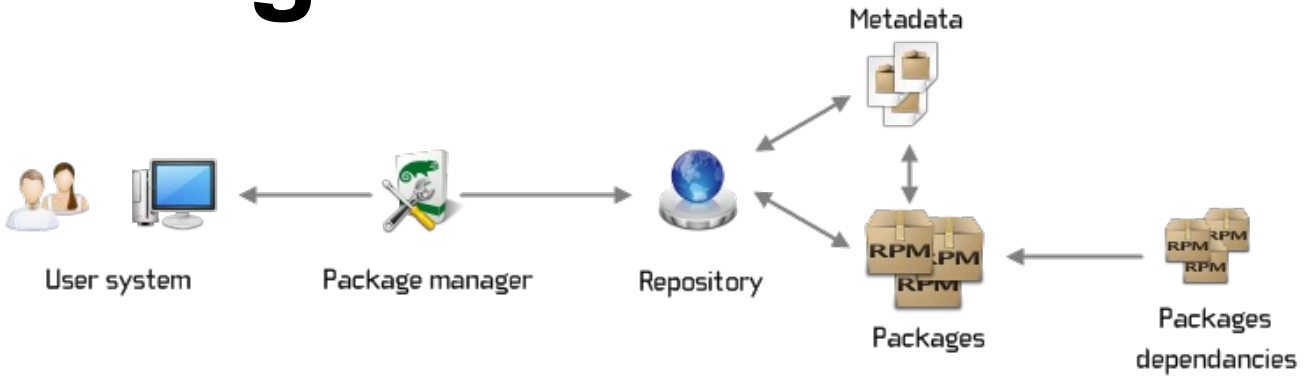
# An Example

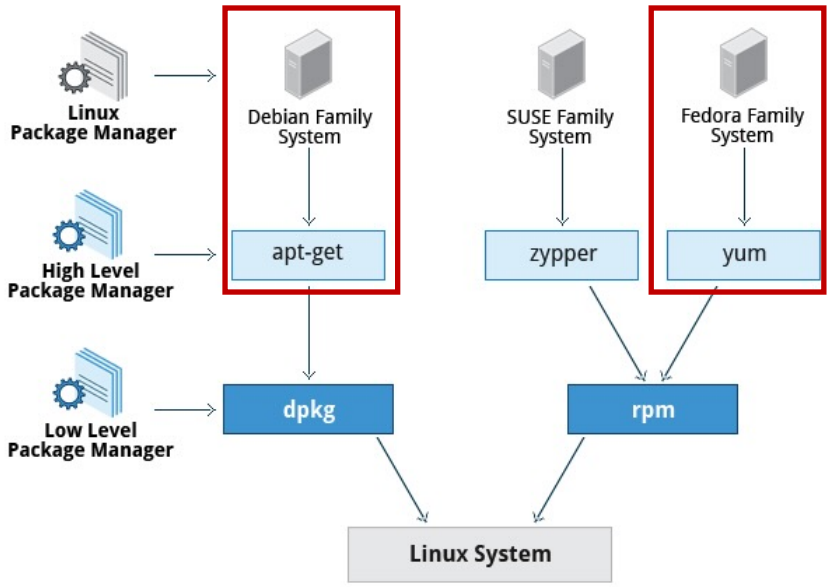
```
Package: firefox-3.0
Priority: optional
Section: web
Installed-Size: 3456
Maintainer: Alexander Sack <asac@ubuntu.com>
Architecture: i386
Version: 3.0.16+nobinonly-0ubuntu0.9.04.1
Replaces: firefox (<< 3), firefox-granparadiso, firefox-libth                x-trunk
Provides: firefox-libthai, www-browser
Depends: fontconfig, psmisc, debianutils (>= 1.16), xulrunner-1.9 (>= 1.9.0.1),
     libatk1.0-0 (>= 1.20.0), libc6 (>= 2.4), libcairo2 (>= 1.2.4),
     libfontconfig1 (>= 2.4.0), libfreetype6 (>= 2.2.1), libgcc1 (>= 1:4.1.1),
     libglib2.0-0 (>= 2.16.0), libgtk2.0-0 (>= 2.16.0),
     libnspr4-0d (>= 4.7.3-0ubuntu1~), libpango1.0-0 (>= 1.14.0),
     libstdc++6 (>= 4.1.1),
     firefox-3.0-branding (>= 3.0.3+nobinonly-0ubuntu1~) |
          abrowser-3.0-branding (>= 3.0.3+nobinonly-0ubuntu1~)
Suggests: ubufox,
     firefox-3.0-gnome-support (= 3.0.16+nobinonly-0ubuntu0.9.04.1),
     latex-xft-fonts, libthai0
Conflicts: firefox (<< 3), firefox-granparadiso (<< 3.0~alpha8-0),
     firefox-libthai, firefox-trunk (<< 3.0~a8~cvs20070914t1713-0)
Size: 887822
Description: safe and easy web browser from Mozilla
 Firefox delivers safe, easy web browsing. A familiar user interface,
 enhanced security features including protection from online identity
 and integrated search let you get the most out of the web.
 .
 Install this firefox package too, if you want to be automatically up
 new major firefox versions in the future.
```

Installing a browser requires to first install some other software in specific versions

There should be no software in these versions

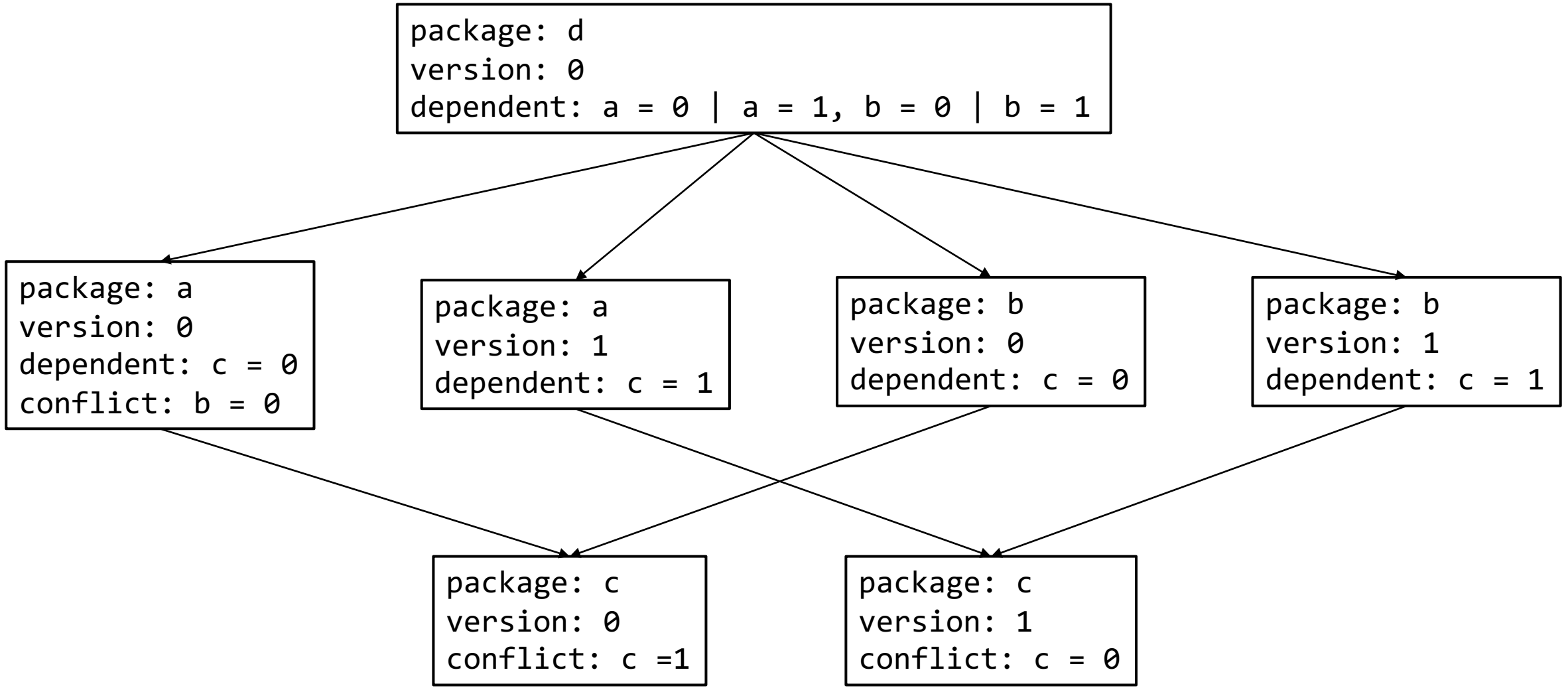
# Package Managers




| | | | |
|---|---|---|---|
| Go 883K Packages | npm 763K Packages | Packagist 215K Packages | Rubygems 146K Packages |
| Maven 138K Packages | PyPI 135K Packages | NuGet 119K Packages | Bower 68.2K Packages |
| WordPress 56.6K Packages | CocoaPods 44.3K Packages | CPAN 35.6K Packages | Clojars 17.7K Packages |
| Cargo 16.1K Packages | Meteor 13.4K Packages | CRAN 13.3K Packages | Hackage 12.7K Packages |
| Atom 11.1K Packages | Hex 6.4K Packages | Puppet 5.69K Packages | PlatformIO 5.14K Packages |
| Homebrew 4.7K Packages | Emacs 4.23K Packages | SwiftPM 4.15K Packages | Pub 3.48K Packages |
| Carthage 2.93K Packages | Julia 2.27K Packages | Sublime 1.97K Packages | Dub 1.45K Packages |
| Elm 1.4K Packages | Racket 1.24K Packages | Haxelib 1.19K Packages | Jam 772 Packages |
| Nimble 702 Packages | Alcatraz 465 Packages | PureScript 237 Packages | Inqlude 217 Packages |
| Shards 31 Packages | | | |

# How to Solve Dependencies



# Encode Dependencies and Conflict

## Dependencies can easily be translated into clauses :

```
package: d
version: 0
dependent: a = 0 | a = 1, b = 0 | b = 1
```

$$d_0 \rightarrow ((a_0 \vee a_1) \wedge (b_0 \vee b_1))$$

## Conflict can easily be translated into binary clauses :

```
package: c
version: 0
conflict: c =1
```

$$c_0 \rightarrow \neg c_1$$

# How to Solve Dependencies

Must install d0

```
package: d
version: 0
dependent: a = 0 | a = 1, b = 0 | b = 1
```

$$\boxed{d_0} \wedge (\neg d_0 \vee a_0 \vee a_1) \wedge (\neg d_0 \vee b_0 \vee b_1)$$

```
package: a
version: 0
dependent: c = 0
conflict: b = 0
```

$$(\neg a_0 \vee c_0) \wedge (\neg a_0 \vee \neg b_0)$$

```
package: a
version: 1
dependent: c = 1
```

$$(\neg a_1 \vee c_1)$$

```
package: b
version: 0
dependent: c = 0
```

$$(\neg b_0 \vee c_0)$$

```
package: b
version: 1
dependent: c = 1
```

$$(\neg b_1 \vee c_1)$$

```
package: c
version: 0
conflict: c =1
```

$$(\neg c_0 \vee \neg c_1)$$

```
package: c
version: 1
conflict: c = 0
```

$$(\neg c_0 \vee \neg c_1)$$

# How to Solve Dependencies

```
> ./apt_dependent.py
  [b_1, c_1, d_0, a_1]
```

Must install d0

```
package: d
version: 0
dependent: a = 0 | a = 1, b = 0 | b = 1
```

$$\boxed{d_0} \wedge (\neg d_0 \vee a_0 \vee a_1) \wedge (\neg d_0 \vee b_0 \vee b_1)$$

```
package: a
version: 0
dependent: c = 0
conflict: b = 0
```

$$(\neg a_0 \vee c_0) \wedge (\neg a_0 \vee \neg b_0)$$

```
package: a
version: 1
dependent: c = 1
```

$$(\neg a_1 \vee c_1)$$

```
package: b
version: 0
dependent: c = 0
```

$$(\neg b_0 \vee c_0)$$

```
package: b
version: 1
dependent: c = 1
```

$$(\neg b_1 \vee c_1)$$

```
package: c
version: 0
conflict: c =1
```

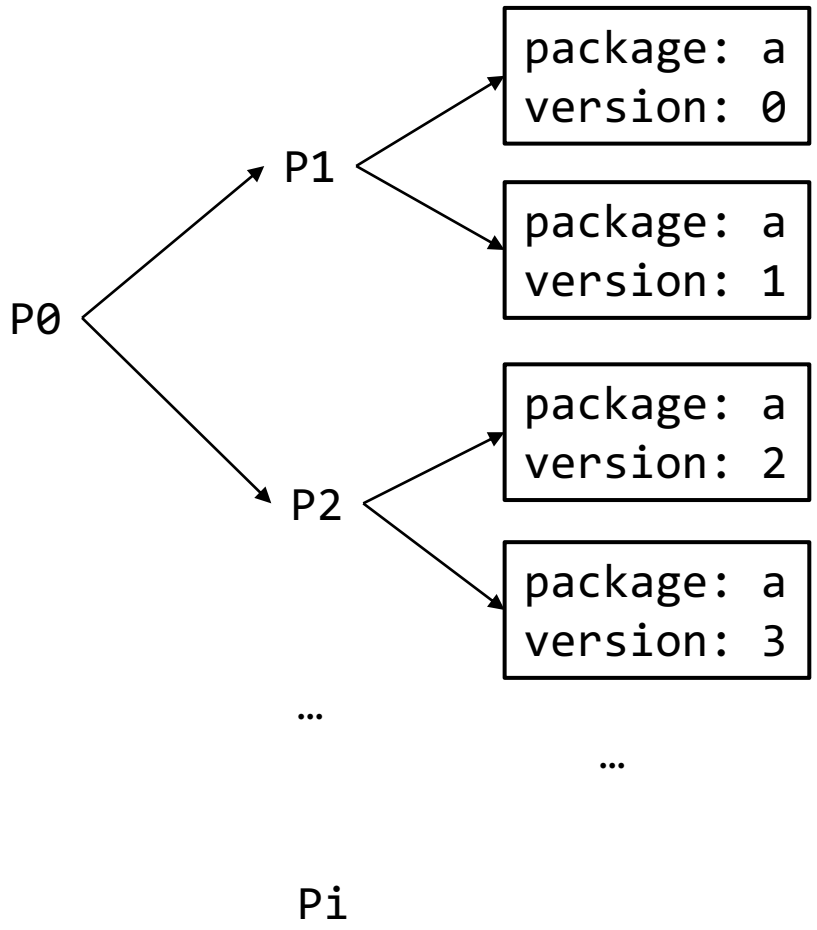
$$(\neg c_0 \vee \neg c_1)$$

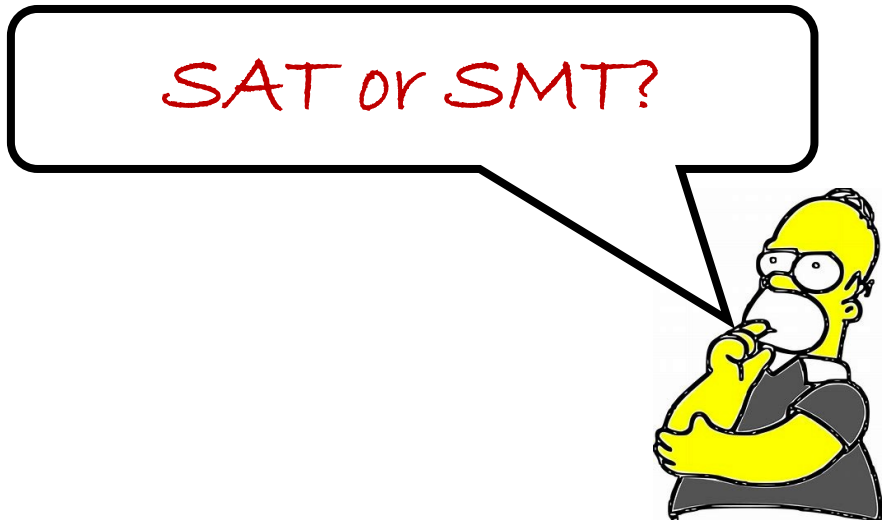
```
package: c
version: 1
conflict: c = 0
```

$$(\neg c_0 \vee \neg c_1)$$

# Express Multiple Versions

## Assume a package has n versions



# Sudoku

## DEFINITION

**Goal:** Fill in a 9×9 grid with numbers

**Rule:**

- Each cell contains a number with a value in {1, ..., 9}
- Each row contains a number exactly once
- Each column contains a number exactly once
- Each 3x3 square contains a number exactly once
- Some numbers have been filled in

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | | 6 | 3 | 4 | | 8 | 1 | |
| | 5 | 1 | 7 | | | 3 | | |
| 4 | 7 | | | 9 | 1 | | | 5 |
| | | | 9 | | 3 | | | 2 |
| | | 2 | | 8 | 7 | | | |
| 1 | | 7 | 2 | | | 6 | | |
| | 8 | 5 | | | 9 | 1 | | |
| | 3 | 4 | | 6 | | | | 9 |
| | 1 | | 5 | | 8 | 7 | | 6 |

# Sudoku Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | | 6 | 3 | 4 | | 8 | 1 | |
| | 5 | 1 | 7 | | | 3 | | |
| 4 | 7 | | | 9 | 1 | | | 5 |
| | | | 9 | | 3 | | | 2 |
| | | 2 | | 8 | 7 | | | |
| 1 | | 7 | 2 | | | 6 | | |
| | 8 | 5 | | | 9 | 1 | | |
| | 3 | 4 | | 6 | | | | 9 |
| | 1 | | 5 | | 8 | 7 | | 6 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 1 | 6 | 3 | 4 | | 8 | 1 | |
| | 5 | 1 | 7 | | | 3 | | |
| 4 | 7 | | | 9 | 1 | | | 5 |
| | | | 9 | | 3 | | | 2 |
| | | 2 | | 8 | 7 | | | |
| 1 | | 7 | 2 | | | 6 | | |
| | 8 | 5 | | | 9 | 1 | | |
| | 3 | 4 | | 6 | | | | 9 |
| | 1 | | 5 | | 8 | 7 | | 6 |

# Solve Sudoku with Z3

```
from z3 import *

# sudoku instance, we use '0' for empty cells
instance = ((9, 0, 6, 3, 4, 0, 8, 1, 0),
            (0, 5, 1, 7, 0, 0, 3, 0, 0),
            (4, 7, 0, 0, 9, 1, 0, 0, 5),
            (0, 0, 0, 9, 0, 3, 0, 0, 2),
            (0, 0, 2, 0, 8, 7, 0, 0, 0),
            (1, 0, 7, 2, 0, 0, 6, 0, 0),
            (0, 8, 5, 0, 0, 9, 1, 0, 0),
            (0, 3, 4, 0, 6, 0, 0, 0, 9),
            (0, 1, 0, 5, 0, 8, 7, 0, 6))

# 9x9 matrix of integer variables
X = [[Int("x_%s_%s" % (i+1, j+1)) for j in range(9)]
                                  for i in range(9)]
```

# Solve Sudoku with Z3

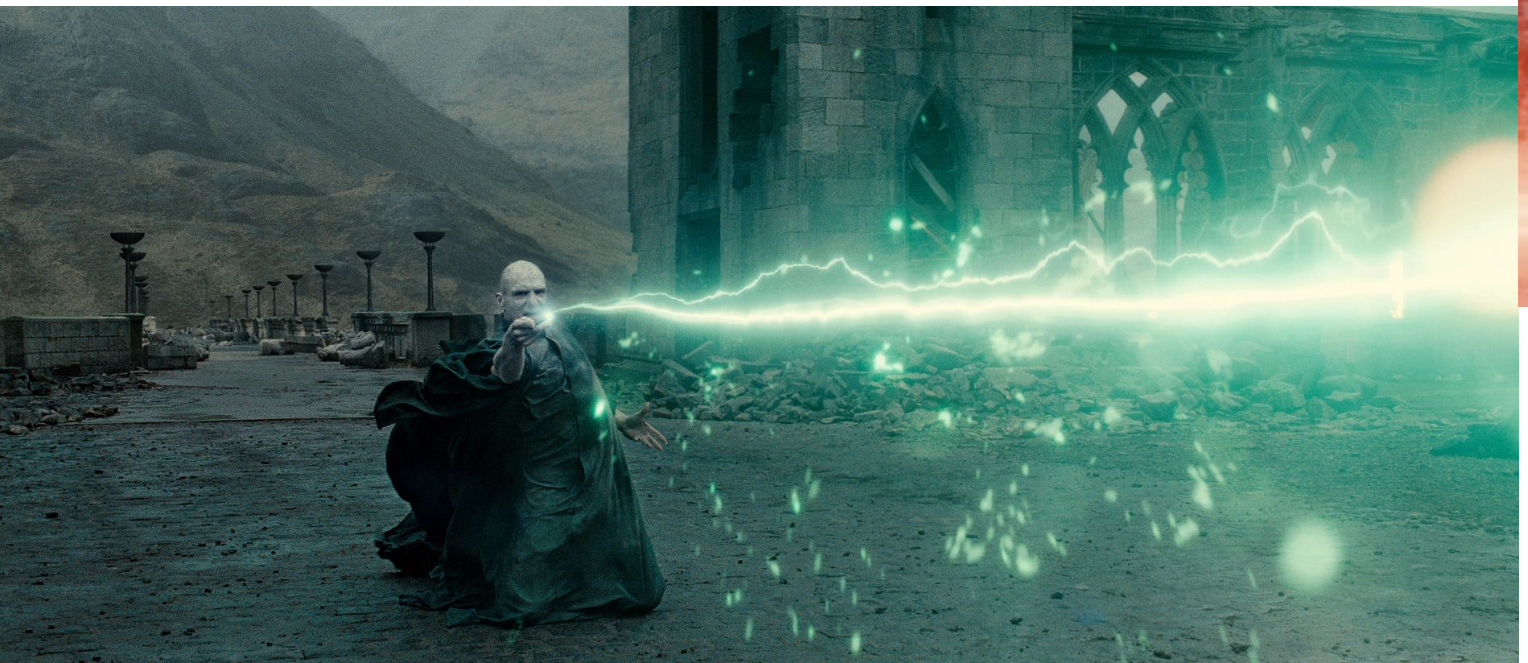
```
# Each cell contains a number with a value in {1, ..., 9}
  cells_c = [And(1 <= X[i][j], X[i][j] <= 9) for i in range(9) for j in range(9)]

# Each row contains a number exactly once
  rows_c = [Distinct(X[i]) for i in range(9)]

# Each column contains a number exactly once
  cols_c = [Distinct([X[i][j] for i in range(9)]) for j in range(9)]

# Each 3x3 square contains a number exactly once
  sq_c = [Distinct([X[3*i0 + i][3*j0 + j] for i in range(3) for j in range(3)])
                                          for i0 in range(3) for j0 in range(3)]

# Set filled numbers in the instance
  instance_c = [If(instance[i][j] == 0, True, X[i][j] == instance[i][j])
                                          for i in range(9) for j in range(9)]
```

# Solve Sudoku with Z3

```
from z3 import *

# sudoku instance, we use '0' for empty cells
instance = ((9, 0, 6, 3, 4, 0, 8, 1, 0),
            (0, 5, 1, 7, 0, 0, 3, 0, 0),
            (4, 7, 0, 0, 9, 1, 0, 0, 5),
            (0, 0, 0, 9, 0, 3, 0, 0, 2),
            (0, 0, 2, 0, 8, 7, 0, 0, 0),
            (1, 0, 7, 2, 0, 0, 6, 0, 0),
            (0, 8, 5, 0, 0, 9, 1, 0, 0),
            (0, 3, 4, 0, 6, 0, 0, 0, 9),
            (0, 1, 0, 5, 0, 8, 7, 0, 6))

# 9x9 matrix of integer variables
X = [[Int("x_%s_%s" % (i+1, j+1)) for j in range(9)]
                                  for i in range(9)]
```

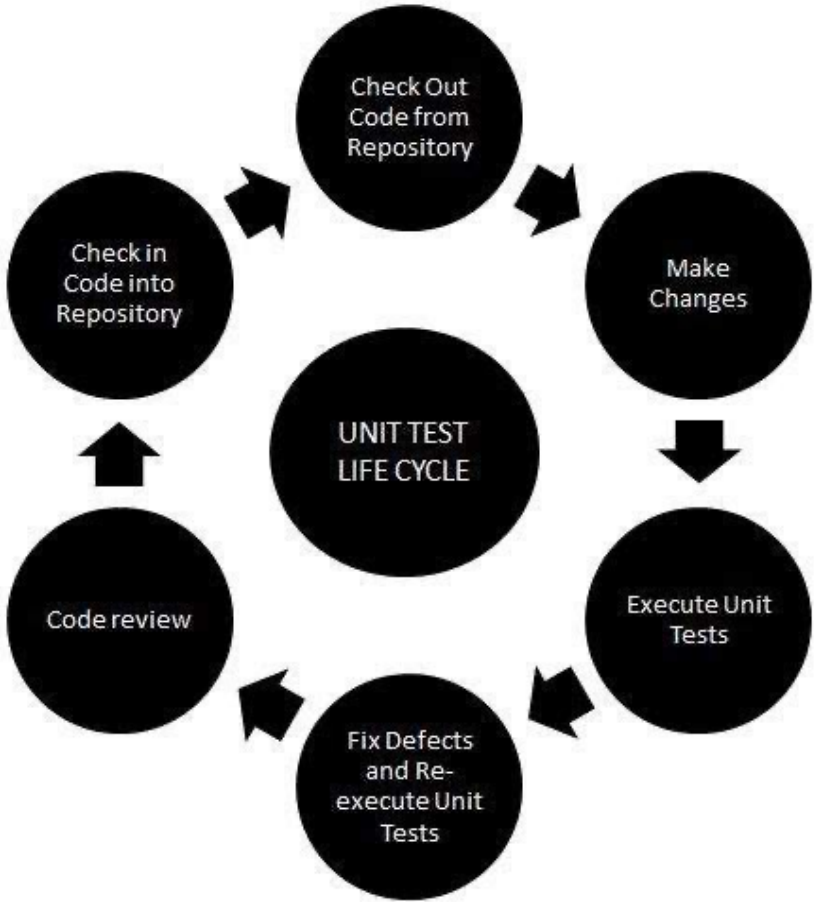
```
> ./sudoku.py
   [[9, 2, 6, 3, 4, 5, 8, 1, 7],
    [8, 5, 1, 7, 2, 6, 3, 9, 4],
    [4, 7, 3, 8, 9, 1, 2, 6, 5],
    [5, 6, 8, 9, 1, 3, 4, 7, 2],
    [3, 4, 2, 6, 8, 7, 9, 5, 1],
    [1, 9, 7, 2, 5, 4, 6, 3, 8],
    [6, 8, 5, 4, 7, 9, 1, 2, 3],
    [7, 3, 4, 1, 6, 2, 5, 8, 9],
    [2, 1, 9, 5, 3, 8, 7, 4, 6]]
```

# SMT solver is a magic wand



# Test Cases Generation

- After writing a program, we should manually write test cases



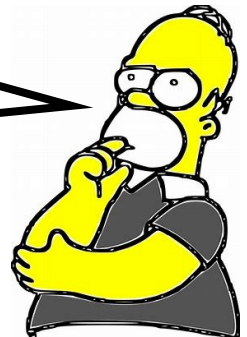
# Test Cases Generation

- **After writing a program, we should manually write test cases**
- **How to estimate the quality of test cases?**

DEFINITION

Code coverage(代码覆盖率) means the ratio of the LoC executed under test cases, which should be as higher as possible.

We can achieve higher code coverage when more execution paths are covered under testing



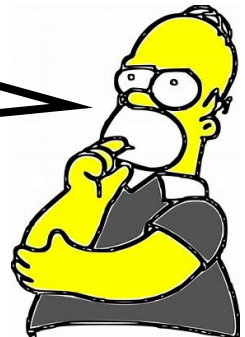
# Test Cases Generation

- **After writing a program, we should manually write test cases**
- **How to estimate the quality of test cases?**

DEFINITION

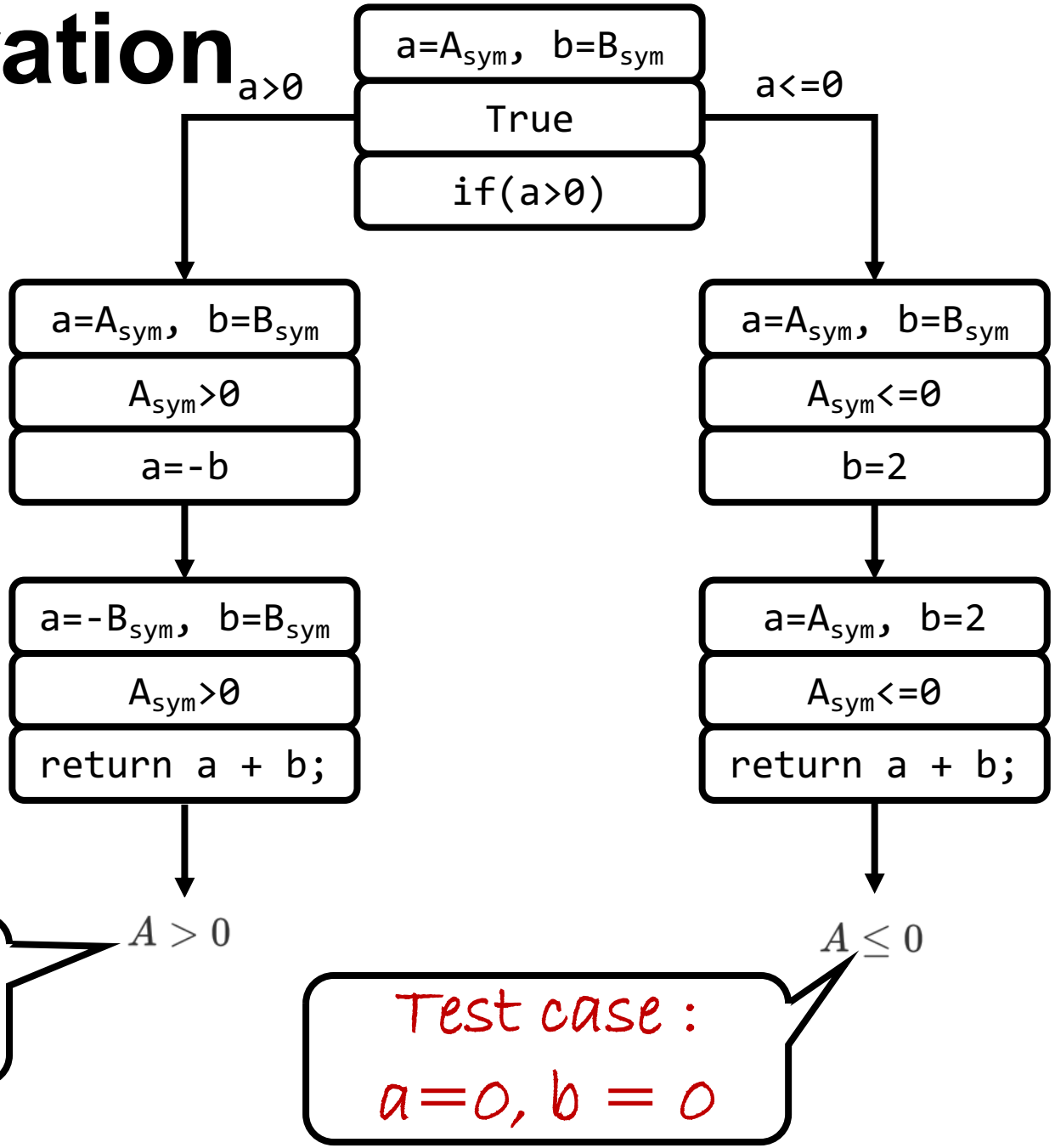
Code coverage(代码覆盖率) means the ratio of the LoC executed under test cases, which should be as higher as possible.

Could we automatically generate test cases with high code coverage?



# Test Cases Generation

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  return a + b;
}
```




**Code coverage : 100%**

Test case :
a=1, b = 0

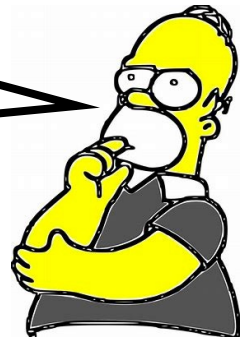
Test case :
a=0, b = 0

# Test Cases Generation

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  return a + b;
}
```

- **Test case 1: a = 1, b = 0; output: 0**
- **Test case 2: a = 0, b = 0; output: 2**

When code coverage is 100% and the code passes all test cases, can we ensure the absence of bugs?



# Test Cases Generation

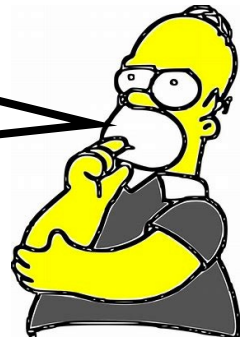
```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  return a + b;
}
```

- **Test case 1: a = 1, b = 0; output: 0**
- **Test case 2: a = 0, b = 0; output: 2**

**When we require that the return value should always be less than 10**

- **The program passes the testing**
- **But it is incorrect (a = 0, b = 100)**

Software testing cannot ensure the correctness, which is complete but not sound.



# Test Cases Generation

KLEE

Getting Started Documentation Tutorials Publications Projects Getting Involved Releases

## KLEE Symbolic Execution Engine

KLEE is a dynamic symbolic execution engine built on top of the LLVM compiler infrastructure, and available under the UIUC open source license. For more information on what KLEE is and what it can do, see the OSDI 2008 paper.

OSDI 2008 Best Paper

http://klee.github.io/

# Example

```
#include "klee/klee.h”

int get_sign(int x) {
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main() {
  int a;
  klee_make_symbolic(&a, sizeof(a), "a");
  return get_sign(a);
}
```

test cases

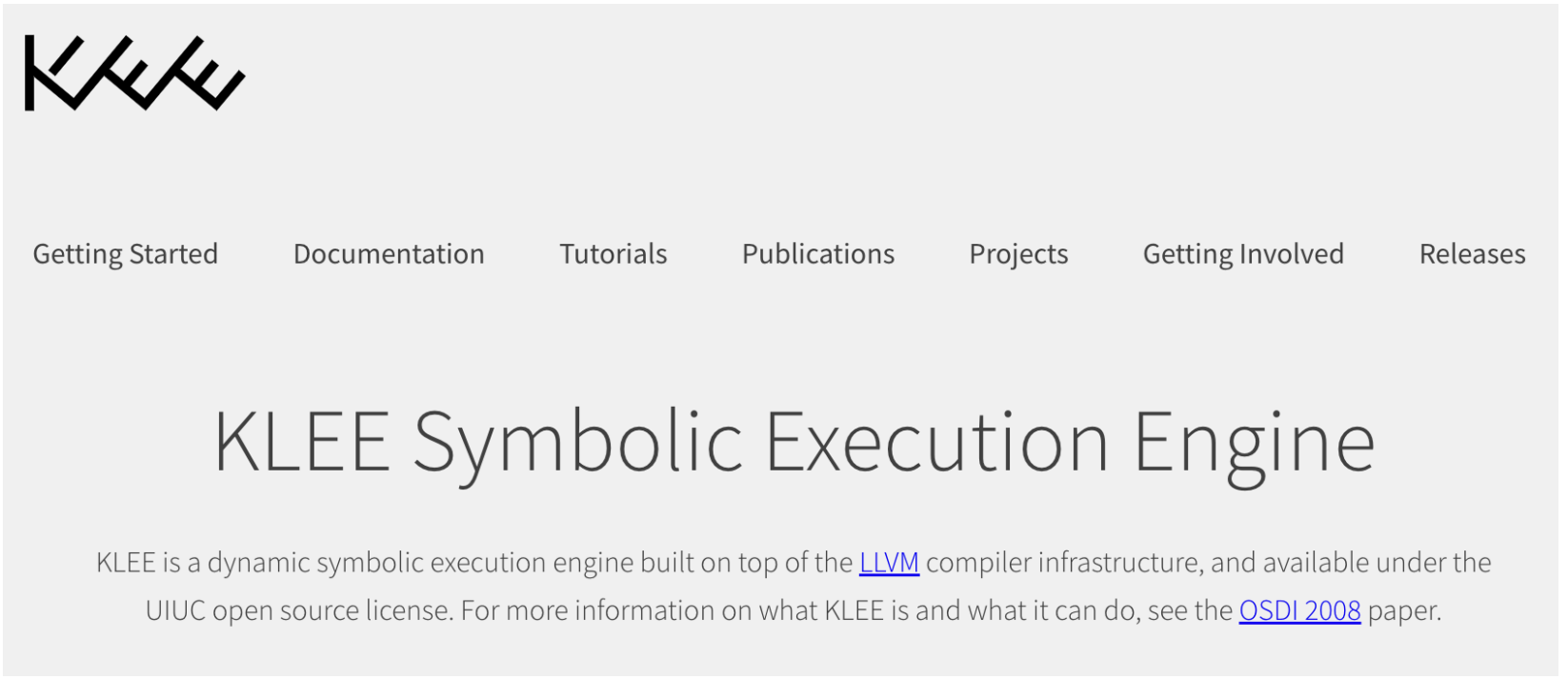
```
> ktest-tool test000001.ktest
  object 0: name: 'a’
  object 0: int : 0
> ktest-tool test000002.ktest
  object 0: name: 'a’
  object 0: int : 16843009
> ktest-tool test000003.ktest
  object 0: name: 'a’
  object 0: int : -2147483648
```

retest

```
> KTEST_FILE=test000001.ktest ./a.out
  0
> KTEST_FILE=test000002.ktest ./a.out
  1
> KTEST_FILE= test000003.ktest ./a.out
  -1
```

# Program Verification

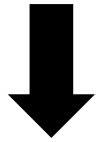
- We still need program verification
- KLEE can also be used to perform verification



# Example

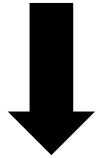
```c
#include "klee/klee.h"
// Reverse the order of bytes in x
uint64_t myhtobe64(uint64_t x){
  uint64_t addr = 0;
  uint8_t byte = 0;
  for (int i = 0; i < sizeof(x) * 8; i+=8) {
     byte = (x >> i) & 0xFF;
     addr |= byte;
     addr <<= 8;
  }
  assert(addr == htobe64(x));
  return addr;
}

int main(int argc, char** argv) {
   uint64_t a;
   klee_make_symbolic(&a, sizeof(a), "a");
   myhtobe64(a);
   return 0;
}
```

```
> ./run.sh
  KLEE: done: total instructions = 394
  KLEE: done: completed paths = 1
  KLEE: done: generated tests = 2
```



test

```
> ktest-tool ./klee-last/test000001.ktest
  Input: 0x56547d800980
  a.out: main.c:18: myhtobe64: Assertion
`addr == htobe64(x)' failed.
```



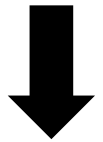
comment assert

```
> KTEST_FILE= ./klee-last/
test000001.ktest ./a.out
  Input: 0x56547d800980
  Myhtobe64: 0x09807d5456000000
  Htobe64:   0x8009807d54560000
```

# Example

```
#include "klee/klee.h"
// Reverse the order of bytes in x
uint64_t myhtobe64(uint64_t x){
  uint64_t addr = 0;
  uint8_t byte = 0;
  for (int i = 0; i < sizeof(x) * 8; i+=8) {
     byte = (x >> i) & 0xFF;
     addr |= byte;
     if(i < (sizeof(x)-1) * 8)
       addr <<= 8;
  }
  assert(addr == htobe64(x));
  return addr;
}

int main(int argc, char** argv) {
   uint64_t a;
   klee_make_symbolic(&a, sizeof(a), "a");
   myhtobe64(a);
   return 0;
}
```

```
> ./run.sh
  KLEE: done: total instructions = 429
  KLEE: done: completed paths = 1
  KLEE: done: generated tests = 1
```



retest

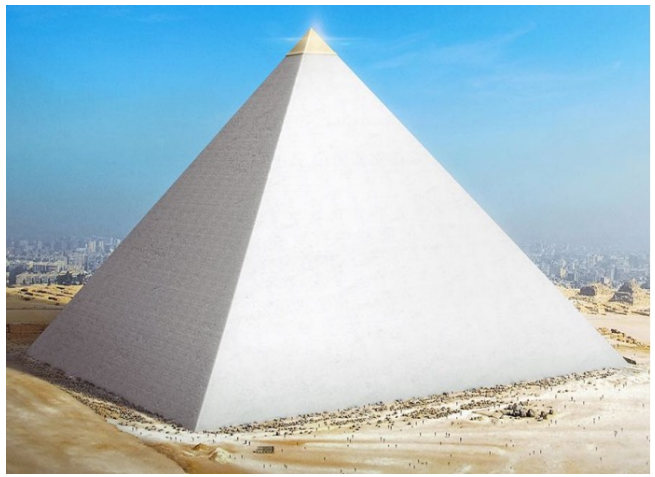
```
> KTEST_FILE= ./klee-
last/test000001.ktest ./a.out
  Input: 0x55de61e00950
  0
```

# An Interesting Problem

**Problem**

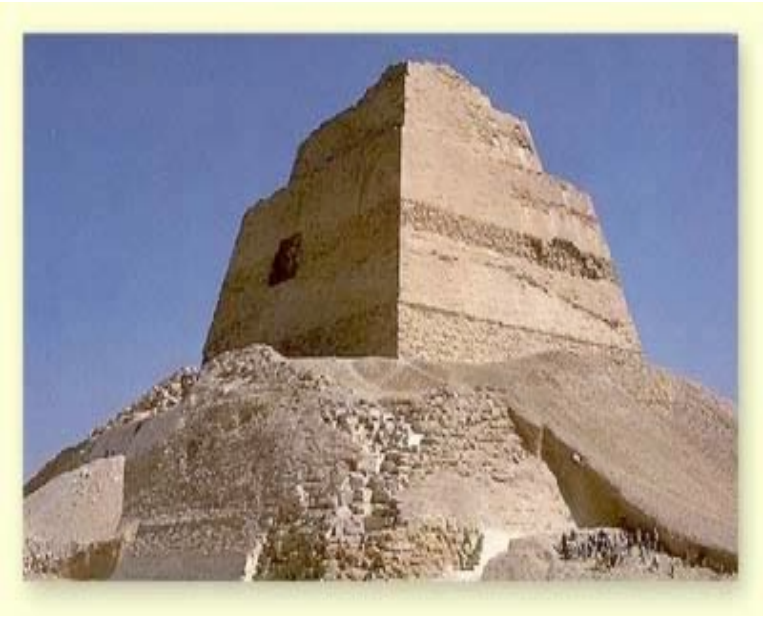
Can we discard software testing with program verification?

Program Testing



Program Verification

# Pharaoh Sneferu’s Pyramid Project



Try 1:
Meidum(52° angle)

Try 2:
Dashur/Bent(52° to 43.5° angle)

Try 3:
Red pyramid(43° angle)

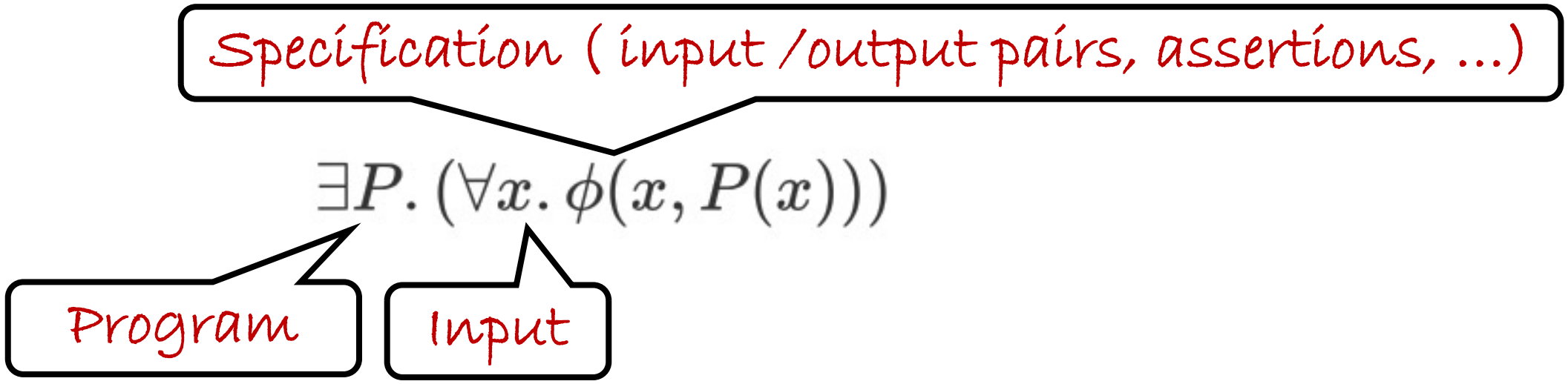
# Program Synthesis

- Program verification tells if the program is incorrect
- It does not tell why the code is incorrect
- Programmers need to manually
  - locate the codes introducing bugs
  - think how to fix the bug

# Program Synthesis

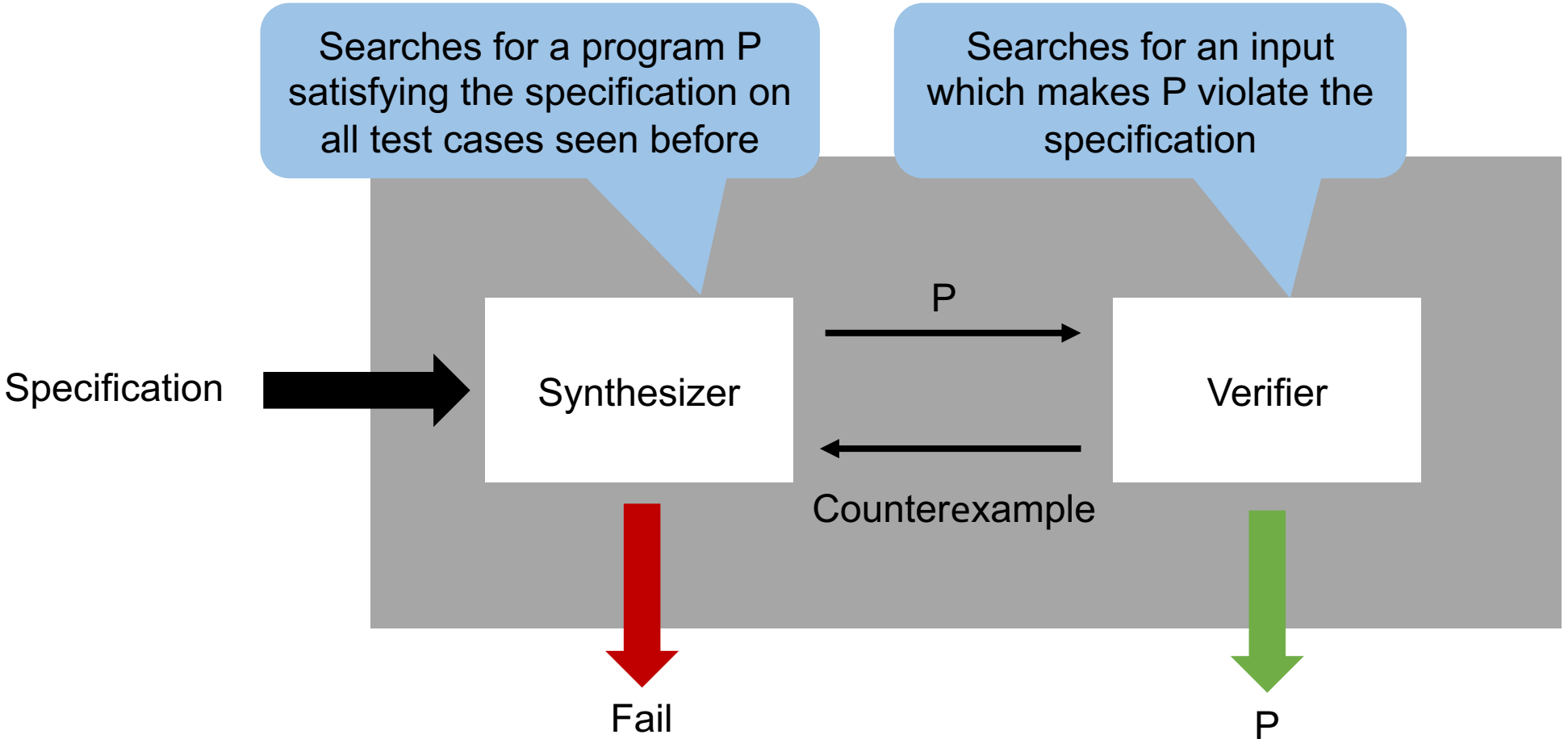
## DEFINITION

Program synthesis(程序合成) is automatically finding a program that satisfies the user intent expressed in the form of some specification.



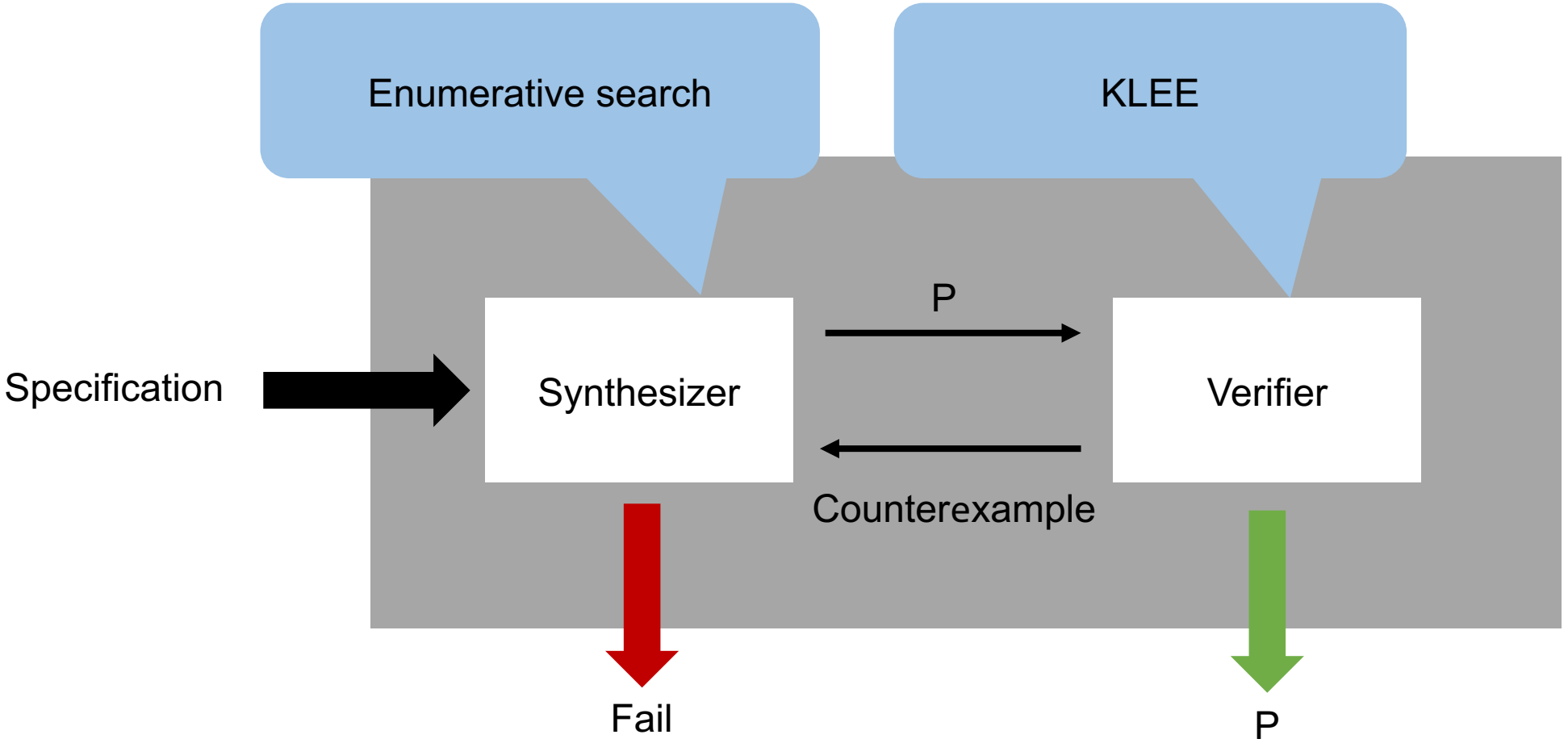

https://www.microsoft.com/en-us/research/publication/program-synthesis/

# Program Synthesis



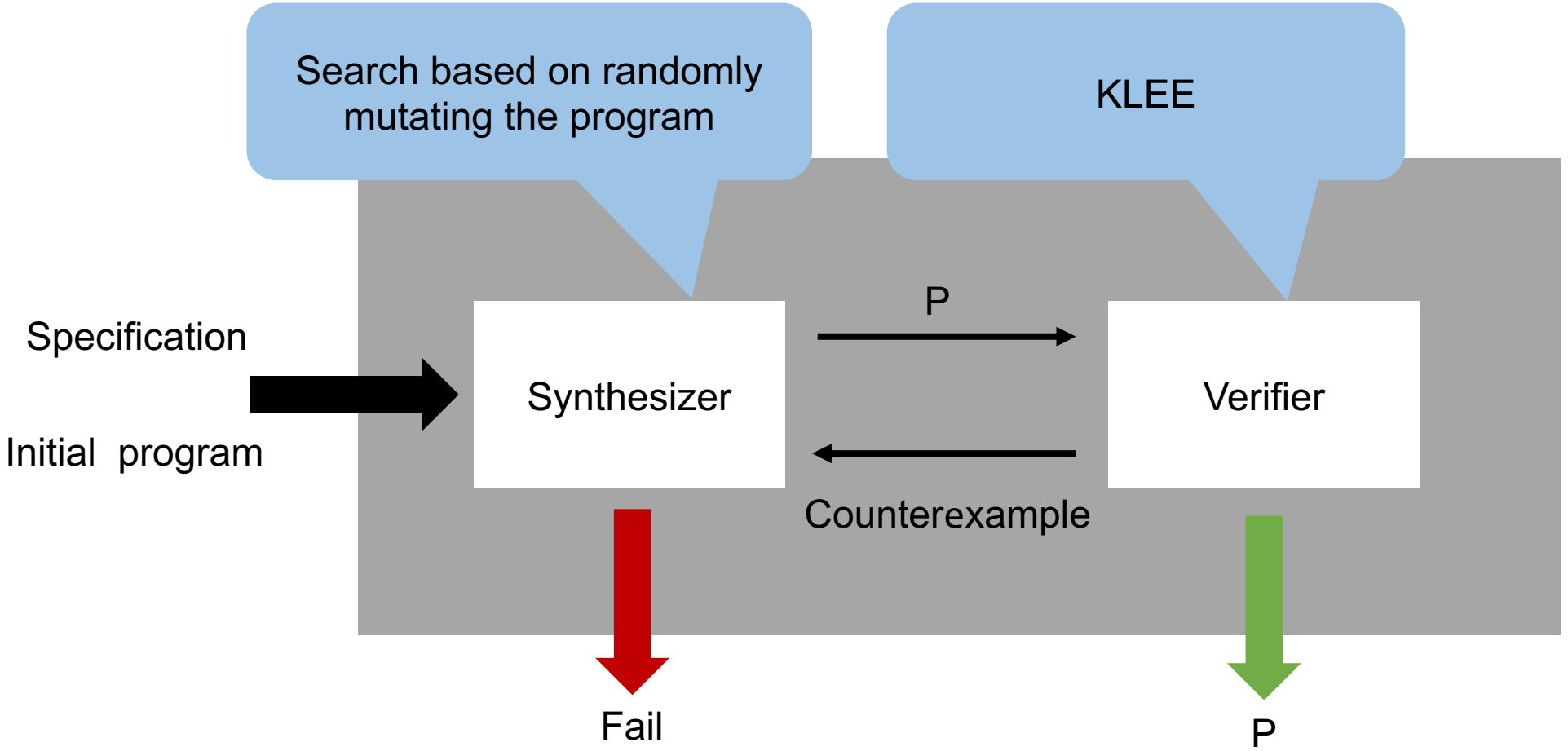

How to implement the synthesizer and the verifier?

# Program Synthesis




**It may be hard to find the program based on enumerative search**

# Program Synthesis




**Symbolic execution cannot handle unbounded loops**

# Program Synthesis

## Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis

Sergey Mechtaev Jooyong Yi Abhik Roychoudhury

School of Computing, National University of Singapore, Singapore
{mechtaev,jooyong,abhik}@comp.nus.edu.sg

ICSE 2016

https://github.com/jyi/fangelix

# Print Number from n-1 to 0

```
#define ANGELIX_OUTPUT(type, expr, id) expr

#include <stdio.h>
#ifndef ANGELIX_OUTPUT
#define ANGELIX_OUTPUT(type, expr, id) expr
#endif

int main(int argc, char *argv[]) {
  int n;
  n = atoi(argv[1]);
  // some complex logic...

  while (n > 1) { // n >= 1
    n--;
    printf("%d\n", ANGELIX_OUTPUT(int, n, "n"));
  }
  return 0;
}
```

oracle

```
Test cases:
"input": 2 "n": [1, 0]
"input": 3 "n": [2, 1, 0]
"input": 4 "n": [3, 2, 1, 0]
```

# Print Number from n-1 to 0

```
#define ANGELIX_OUTPUT(type, expr, id) expr

#include <stdio.h>
#ifndef ANGELIX_OUTPUT
#define ANGELIX_OUTPUT(type, expr, id) expr
#endif

int main(int argc, char *argv[]) {
  int n;
  n = atoi(argv[1]);
  // some complex logic...

  while (n > 1) { // n >= 1
    n--;
    printf("%d\n", ANGELIX_OUTPUT(int, n, "n"));
  }
  return 0;
}
```

oracle

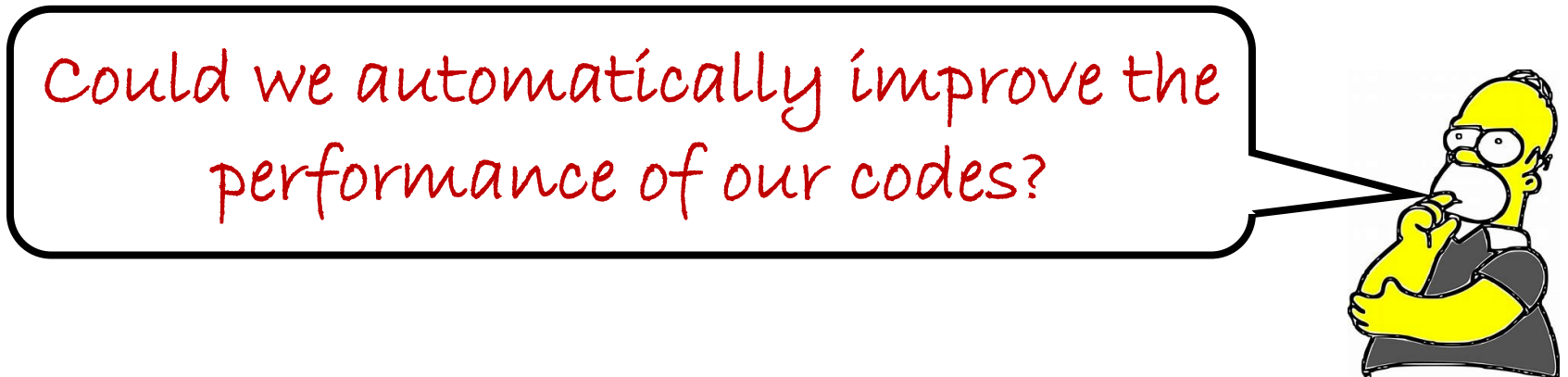
```
Test cases:
"input": 2 "n": [1, 0]
"input": 3 "n": [2, 1, 0]
"input": 4 "n": [3, 2, 1, 0]
```

patch

```
--- a/test.c
+++ b/test.c
@@ -27,7 +27,7 @@
     i--;
   }

-  while (n > 1) { // n >= 1
+  while ((n >= 1)) { // n >= 1
     n--;
}
```

# Superoptimization

- Correctness is not the only goal of programming
- Performance is also very important
- However, program synthesis does not guarantee the performance



# Superoptimization

"Given an instruction set, the superoptimizer finds the shortest program to compute a function"

## Superoptimizer -- A Look at the Smallest Program

*Henry Massalin*

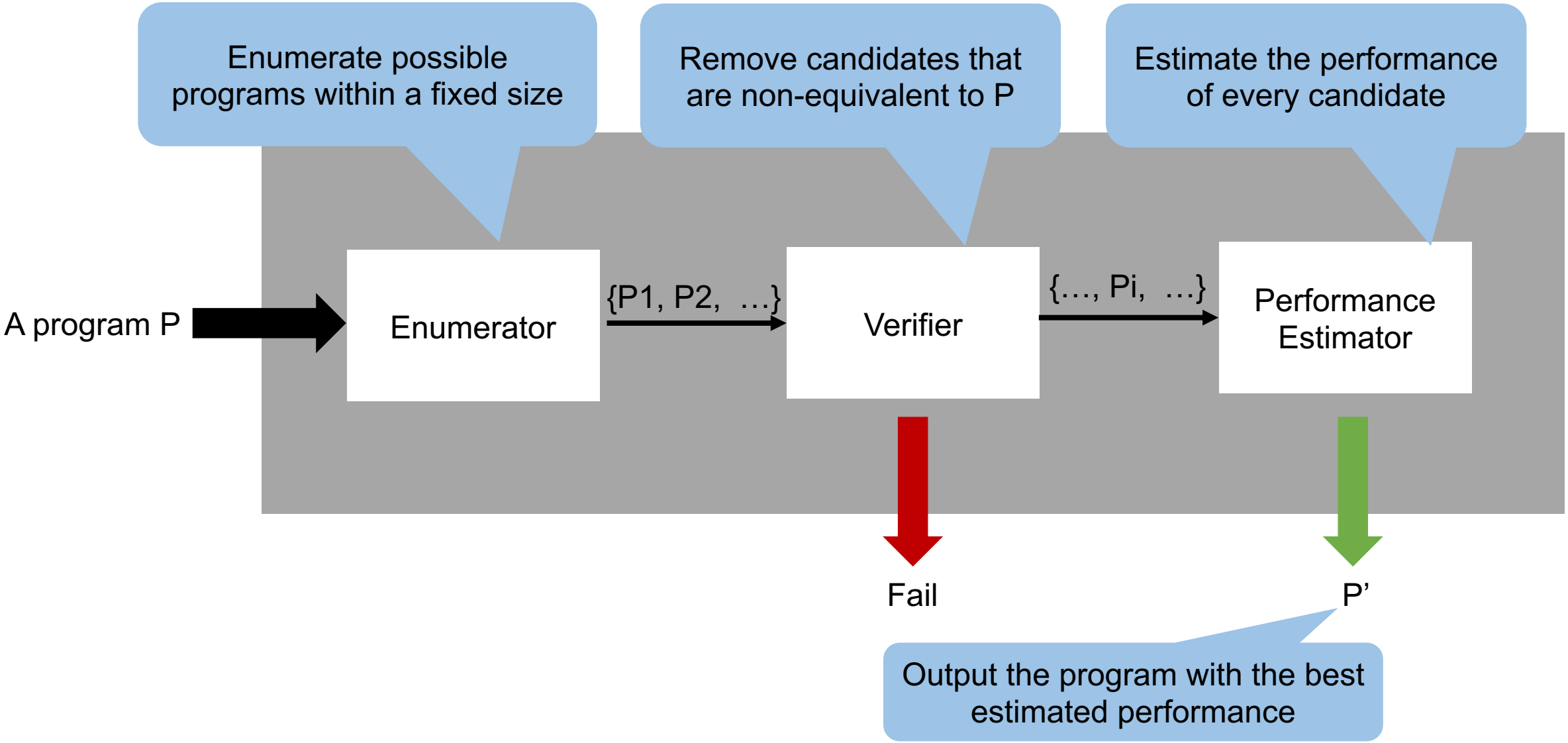
Department of Computer Science
Columbia University
New York, NY 10027

**Abstract**

Given an instruction set, the superoptimizer finds the shortest program to compute a function. Startling programs have been generated, many of them engaging in convoluted bit-fiddling bearing little resemblance to the source programs which defined the functions. The key idea in the superoptimizer is a probabilistic test that makes exhaustive searches practical for programs of useful size. The search space is defined by the processor's instruction set, which may include the whole set, but it is typically restricted to a subset. By constraining the instructions and observing the effect on the output program, one can gain insight into the design of instruction sets. In addition, superoptimized programs may be used by peephole optimizers to improve the quality of generated code, or by assembly language programmers to improve manually written code.

https://courses.cs.washington.edu/courses/cse501/15sp/papers/massalin.pdf

# Superoptimization

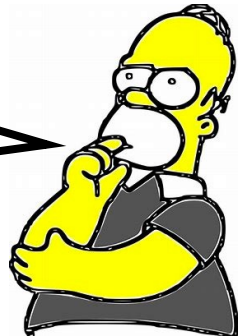


# Superoptimization

```
int func1(int a) {
  if(a > 0)
    return 0;
  else
    return 1;
}
```

```
int func2(int a) {
  a = a + a;
  if(a > 0)
    return 0;
  else
    return 1;
}
```

How to verify the equivalence between two programs?



# Superoptimization

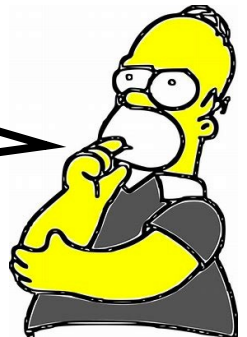
```
int func1(int a) {
  if(a > 0)
    return 0;
  else
    return 1;
}
```

```
int func2(int a) {
  a = a + a;
  if(a > 0)
    return 0;
  else
    return 1;
}
```

```
void func(int a) {
  int val1 = func1(a);
  int val2 = func2(a);
  assert(val1 == val2);
}
```

Given the same input, func1 and func2 always return the same value.

We can still use symbolic execution engines



# Superoptimization

```
int func1(int a) {
  if(a > 0)
    return 0;
  else
    return 1;
}
```

```
int func2(int a) {
  a = a + a;
  if(a > 0)
    return 0;
  else
    return 1;
}
```

**A simple method of estimating the performance**

- **Estimate the execution time of every instruction**
- **Compute the sum of the execution time of all instructions**

**The performance of func1 is better than func2**

How to estimate the performance?

# Superoptimization

## Stochastic Superoptimization

Eric Schkufza
Stanford University
eschkufz@cs.stanford.edu

Rahul Sharma
Stanford University
sharmar@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu

ASPLOS 2013

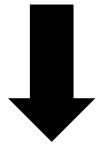
https://github.com/StanfordPL/stoke

# Example

```
/* The number of 1s in the binary
   representation */
size_t popcnt(uint64_t x) {
  int res = 0;
  for (; x > 0; x >>= 1) {
    res += x & 0x1;
  }
  return res;
}

int main(int argc, char** argv) {
  const auto itr = atoi(argv[1]);
  auto ret = 0; uint64_t j = 1;
  for (auto i = 0; i < itr; ++i) {
    j = (j*19 + 7 + (rand() % 5));
    ret += popcnt(j);
  }
  std::cout << ret << std::endl;
  return 0;
}
```

bins/_Z6popcntm.s

```
# Text                  #  Line  RIP       Bytes  Opcode
._Z6popcntm:            #        0x4009a0  0      OPC=<label>
  testq %rdi, %rdi      #  1     0x4009a0  3      OPC=testq_r64_r64
  je .L_4009bf          #  2     0x4009a3  2      OPC=je_label
  xorl %eax, %eax       #  3     0x4009a5  2      OPC=xorl_r32_r32
.L_4009b0:              #        0x4009b0  0      OPC=<label>
  movl %edi, %edx       #  13    0x4009b0  2      OPC=movl_r32_r32
  andl $0x1, %edx       #  14    0x4009b2  3      OPC=andl_r32_imm8
  addl %edx, %eax       #  15    0x4009b5  2      OPC=addl_r32_r32
  shrq $0x1, %rdi       #  16    0x4009b7  3      OPC=shrq_r64_one
  jne .L_4009b0         #  17    0x4009ba  2      OPC=jne_label
  cltq                  #  18    0x4009bc  2      OPC=cltq
  retq                  #  19    0x4009be  1      OPC=retq
.L_4009bf:              #        0x4009bf  0      OPC=<label>
  xorl %eax, %eax       #  20    0x4009bf  2      OPC=xorl_r32_r32
  retq                  #  21    0x4009c1  1      OPC=retq
```



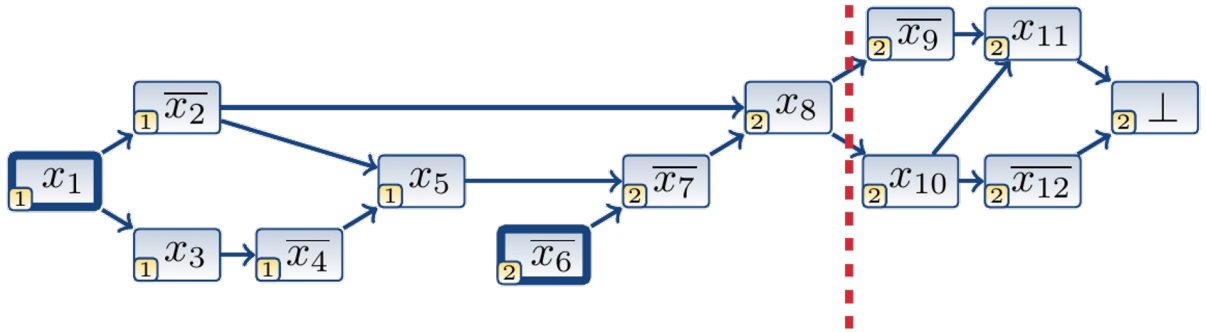
result.s

```
# Text                  #  Line  RIP       Bytes  Opcode
._Z6popcntm:            #        0x4009a0  0      OPC=<label>
  popcntq %rdi, %rax    #  1     0x4009a0  5      OPC=popcntq_r64_r64
  setl %ch              #  2     0x4009a5  3      OPC=setl_rh
  retq                  #  3     0x4009a8  1      OPC=retq
```

# Bonus Lab

CS2501 Discrete Mathematics  for Computer Science

# CDCL Solver




- Unit Propagation
- Build trail tracing reasons of assignments
- Find UIP (Unique Implication Point)
- Learn the cause of contradiction

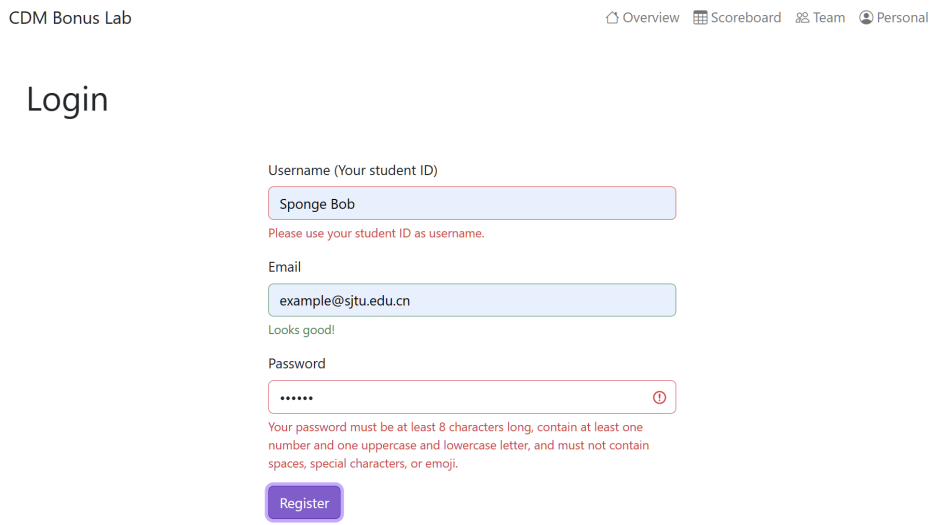
- Detailed explanation is in recitation class on Saturday

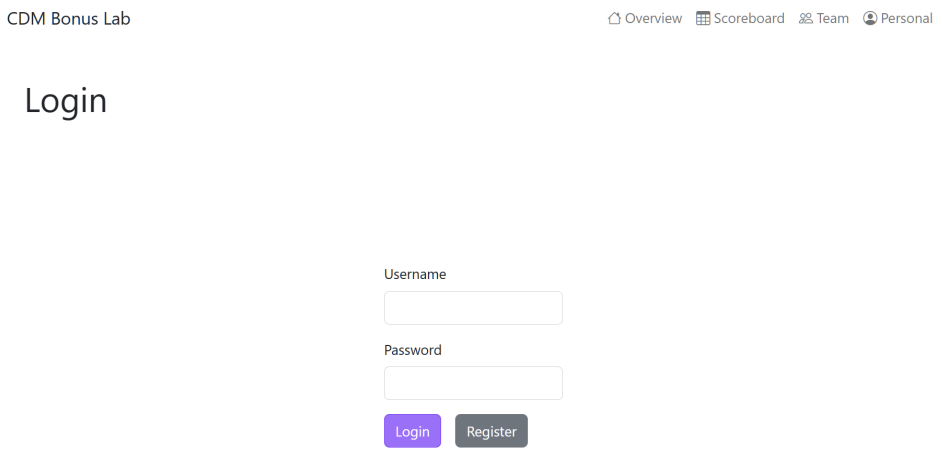
# Goal

- Phase 1: Basic optimization
  - Fix the problem of huge memory consumption
  - Caused by incorrect strategy of backtracking
  - Patiently and carefully read the code to find the problem!
- Phase 2: Compete!
  - Try to solve more CNF formulas in limited time
  - The awards are strongly related to your score.

# Register and Login

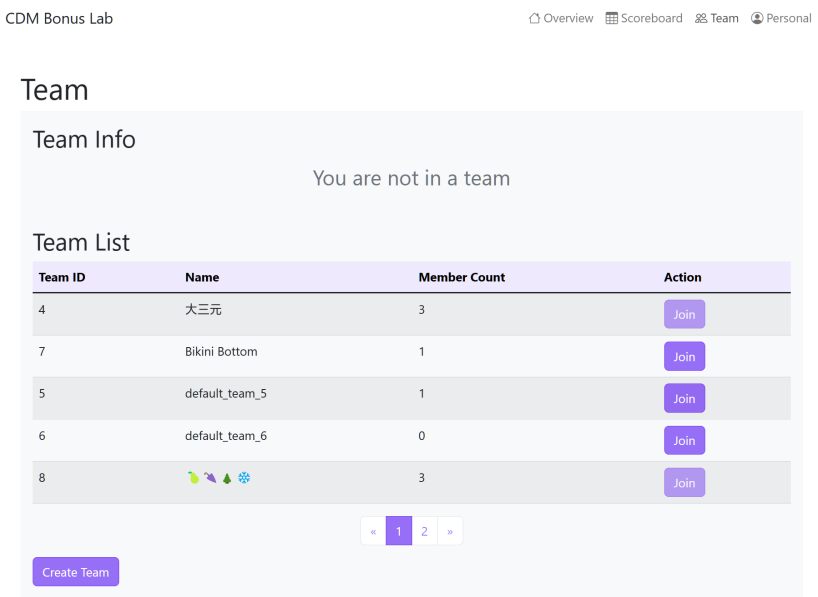
Use your Student ID as Username!

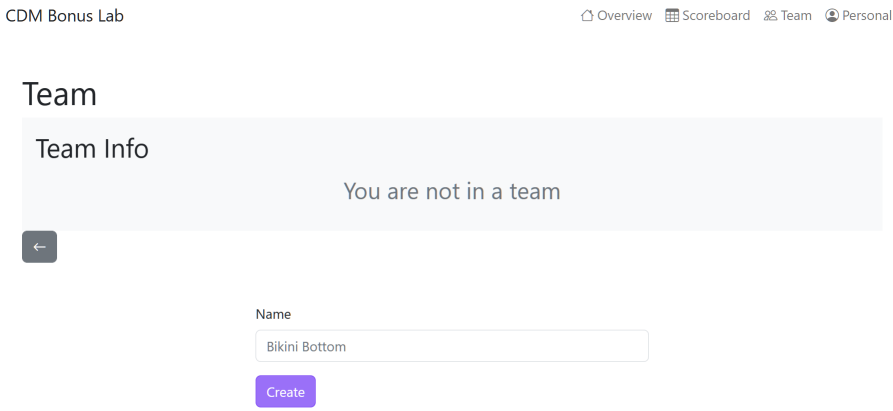


# Team

- Create or Join a team
- If a team of 3 members is formed, all members cannot join or quit anymore.



# Part 1: Basic optimization

- Check your answer by make part1_test

```
../test/cnf/add64.cnf
# 20 ...test/cnf/run.sh: running CNF test core 'add128'
../build/cadical \
../test/cnf/add128.cnf --check ../build/test-cnf-core-add128.prf
# 20 ...FAILED (actual exit code 134)
test/cnf/run.sh: running CNF test simp 'add128'
../build/../scripts/run-simplifier-and-extend-solution.sh \
../test/cnf/add128.cnf
# 20 ...FAILED (actual exit code 1)
test/cnf/run.sh: running CNF test core 'prime65537'
../build/cadical \
../test/cnf/prime65537.cnf --check ../build/test-cnf-core-prime65537.prf
# 20 ...FAILED (actual exit code 134)
test/cnf/run.sh: running CNF test simp 'prime65537'
../build/../scripts/run-simplifier-and-extend-solution.sh \
../test/cnf/prime65537.cnf
# 20 ...test/cnf/run.sh: CNF testing results: 129 ok, 13 failed
make: *** [makefile:19: part1_test] 错误 13
```

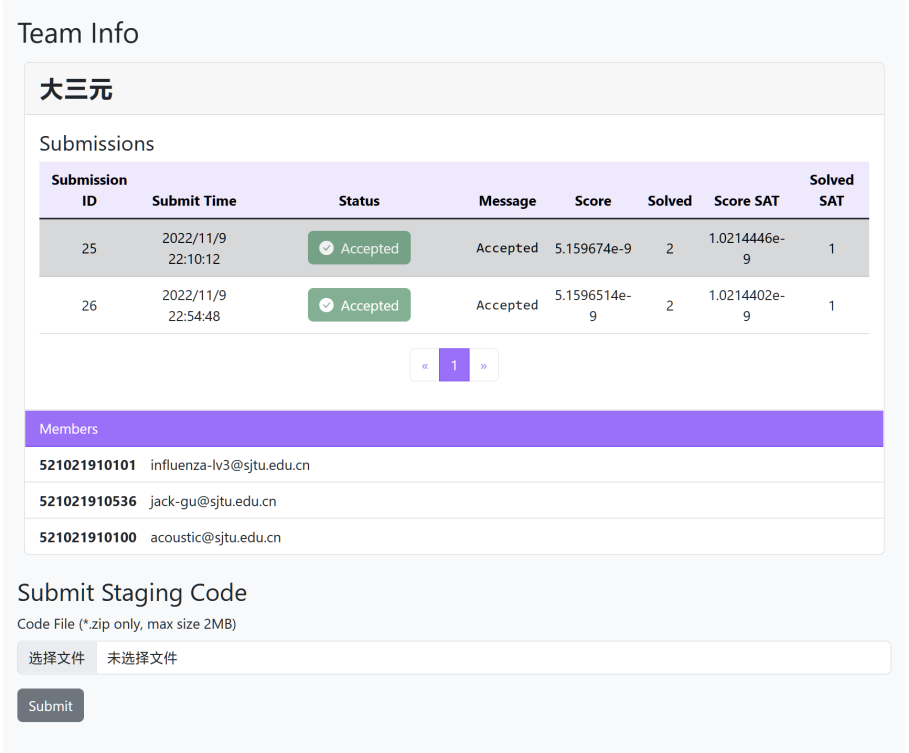
Failed output

```
../build/drat-trim \
../test/cnf/add128.cnf ../build/test-cnf-core-add128.prf
# 0 ...ok (proof checked)
test/cnf/run.sh: running CNF test simp 'add128'
../build/../scripts/run-simplifier-and-extend-solution.sh \
../test/cnf/add128.cnf
# 20 ...test/cnf/run.sh: running CNF test core 'prime65537'
../build/cadical \
../test/cnf/prime65537.cnf --check ../build/test-cnf-core-prime65537.prf
# 20 ...ok (exit code as expected)
../build/drat-trim \
../test/cnf/prime65537.cnf ../build/test-cnf-core-prime65537.prf
# 0 ...ok (proof checked)
test/cnf/run.sh: running CNF test simp 'prime65537'
../build/../scripts/run-simplifier-and-extend-solution.sh \
../test/cnf/prime65537.cnf
# 20 ...test/cnf/run.sh: CNF testing results: 141 ok, 0 failed
```

Success output

# Part 2: The Contest

- Submit your team’s solver to the website
- The website will immediately
  - Check if the solver passed part1
  - Let the solver solve for some CNF formulas ( 400 CNFs in total, try it!)
  - Rank the qualified results
- Scoreboard shows the best results for each team.



# Part 2: The Contest

CDM Bonus Lab

Overview Scoreboard Team Personal

## Scoreboard

| Team Id | Team Name | Score | Solved | Score SAT | Solved SAT | Submit Time |
|---|---|---|---|---|---|---|
| 4 | 大三元 | 5.159674e-9 | 2 | 1.0214446e-9 | 1 | 2022/11/9 22:10:12 |

Total score

Total CNFs Solved

Score for SAT CNFs

Total SAT CNFs Solved

# Enjoy it!

**TA**

Zhuohao Shen (申倬豪)

ao7777@sjtu.edu.cn

# Thanks & Questions