软件测试经典例子

来自苏黎世联邦理工苏振东老师的工作(FSE 2024 keynote)

钮鑫涛

Key mission of computer science

- 计算机科学的关键任务
  - 帮助人类把创造性的思想变为可以运行的系统
- 该任务的核心即为：
  - 软件！
- 由于被大量的依赖（甚至是一些性命、财产攸关的任务），基础性的软件的正确性尤为重要
  - 比如编译器、数据库、操作系统、OPENSSL、定理证明器

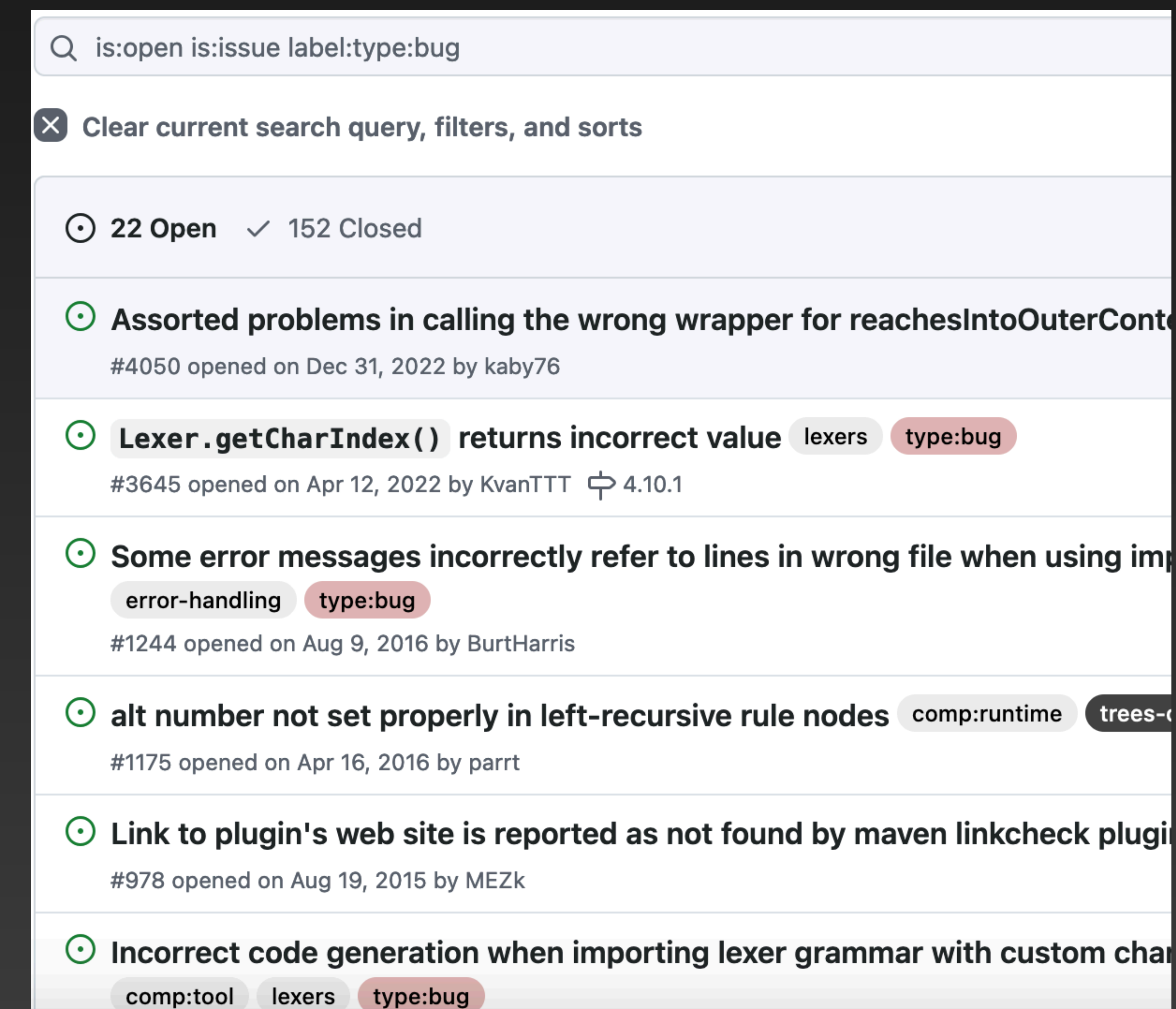
三个关键性软件

- 编译器
- 数据库引擎
- SMT solvers

编译器

编译器

- 编译器是重要的基础软件
- parse部分的理论基础（自动机）和实现（Antler）往往非常成熟
  - 额, Antler真的很成熟吗
    - 但确实相对成熟和简单
- 更多的错误和代码优化有关



一个样例

来自LLVM

```c
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

- “very, very concerning when I got to the root cause, and very annoying to fix…”
  - http://llvm.org/bugs/show_bug.cgi?id=14972

编译器的Oracle?

- 编译器的formal specification
  - 当然可以，尽管很难，但有，比如CompCert就是一个经过完全形式化证明的C编译器，（其前提就是将所有的规约都形式了，C11制定了这样的标准）
    - 然而，证明是非常昂贵，因此CompCert缺失很多有用的C特性（并且由于主流编译器开发者缺少形式化验证的能力，连添加新功能，如一些优化的pass，都很难）。此外，CompCert的有些没有证明的部分还是有很多bugs
  - 那么有其他吗?

回顾之前的Oracle章节

- Differential Testing
  - 对同一个specification的不同实现
    - 对于C编译器而言：GCC、CLang、CompCert
      - 随机生成一批C程序，然后观察这些C编译器编译后的的程序运行时行为是否一致（比如，对于相同输入，输出是否相同）
      - Csmith的工作，核心在于如何生成一批没有未定义行为的C程序

Yang X, Chen Y, Eide E, et al. Finding and understanding bugs in C compilers[C]//Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation (PLDI). 2011: 283-294.

回顾之前的Oracle章节

- Csmith的工作很好（其地位很高），因为大量的生成c program的能力是非常具有挑战的，其可以说是很多complier testing的根源
  - 但其能够生成的类型有限，但很多C的特性还没包括（同一批人的YarpGen、YarpGen V2后续增强了loop的生成）
  - 生成更多类型？很难
    - 从已有的修改？已有的测试用例太有限了！

Livinskii V, Babokin D, Regehr J. Random testing for C and C++ compilers with YARPGen[J]. Proceedings of the ACM on Programming Languages, 2020, 4(OOPSLA): 1-25.

Livinskii V, Babokin D, Regehr J. Fuzzing loop optimizations in compilers for C++ and data-parallel languages[J]. Proceedings of the ACM on Programming Languages, 2023, 7(PLDI): 1826-1847

回顾之前的Oracle章节

- 利用Metamorphic Testing！
  - 根据测试用例，可以生成后续的测试用例
    - 并且利用两者之间的关系，可以判定他们的结果是否符合预期（e.g., $x_1 = x_2 + 360 \implies \sin(x_1) = \sin(x_2)$, 给定一个测试用例$x_1$，我们可以生成很多相关的测试用例：如$x_2, x_2 + 360, x_2 + 720...$），当然不只是相等关系可以利用，其他性质也可以利用
    - 那么编译器有metamorphic relation吗？

回顾之前的Oracle章节

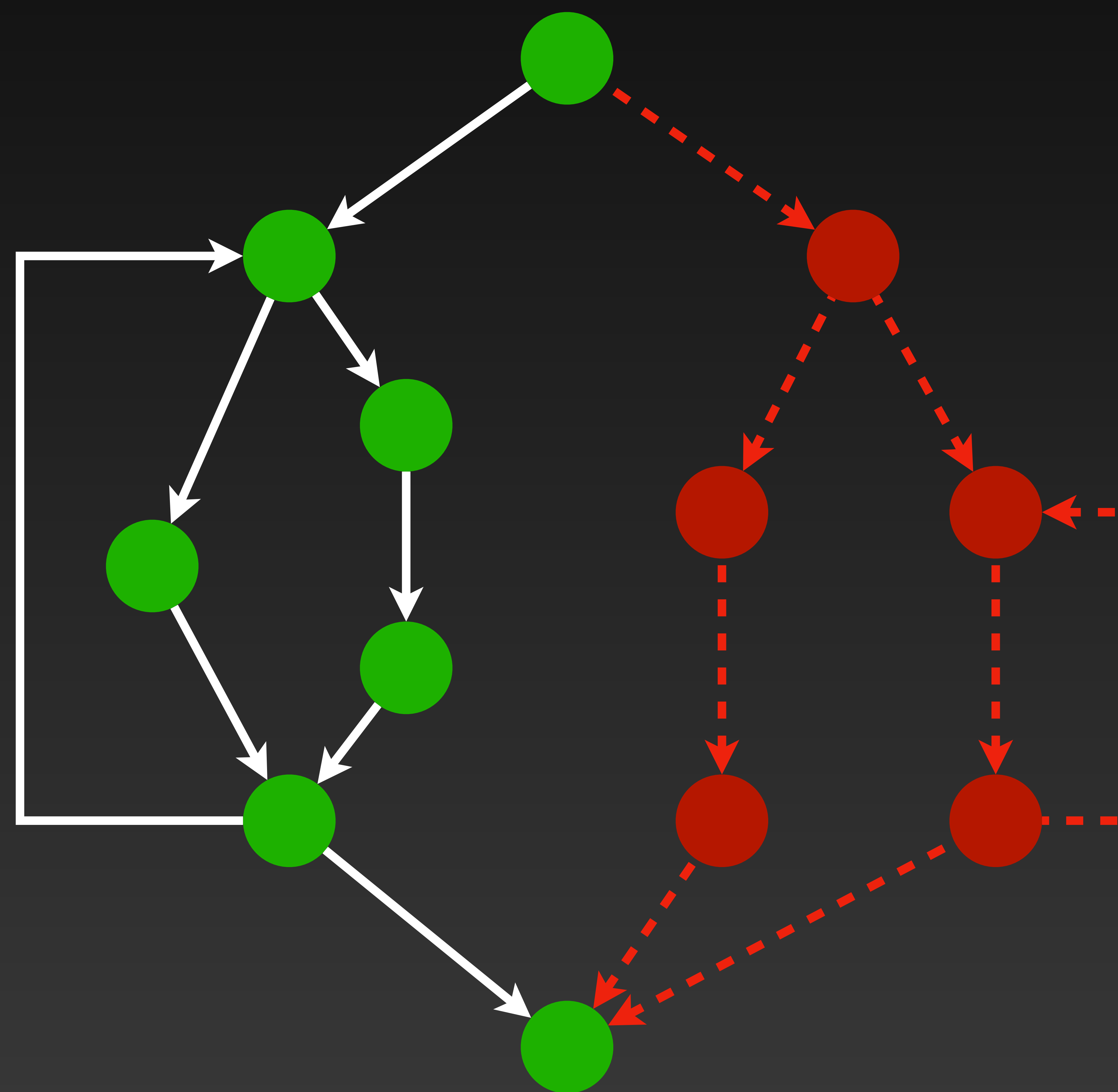
- 一个简单的想法，一个程序和插入死代码的程序在语义上是完全等价的
  - 但，死代码过于简单了，有没有更好的?
- Su带来了EMI的关系，其思想最震撼我的地方在于：其思考了一个程序的部分执行和完全执行之间的关系
  - 部分执行的正确性是包含在完全执行的正确性之内的，形式化的表达：
    - 传统的等价：$P \equiv Q \iff \forall i, P(i) = Q(i)$
    - Equiv. Modulo Inputs (EMI) : $P \equiv^{i} Q \iff P(i) = Q(i)$

Vu Le, Mehrdad Afshari, and Zhendong Su. 2014. Compiler validation via equivalence modulo inputs. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14).

EMI

- Profile

给定一次输入 I



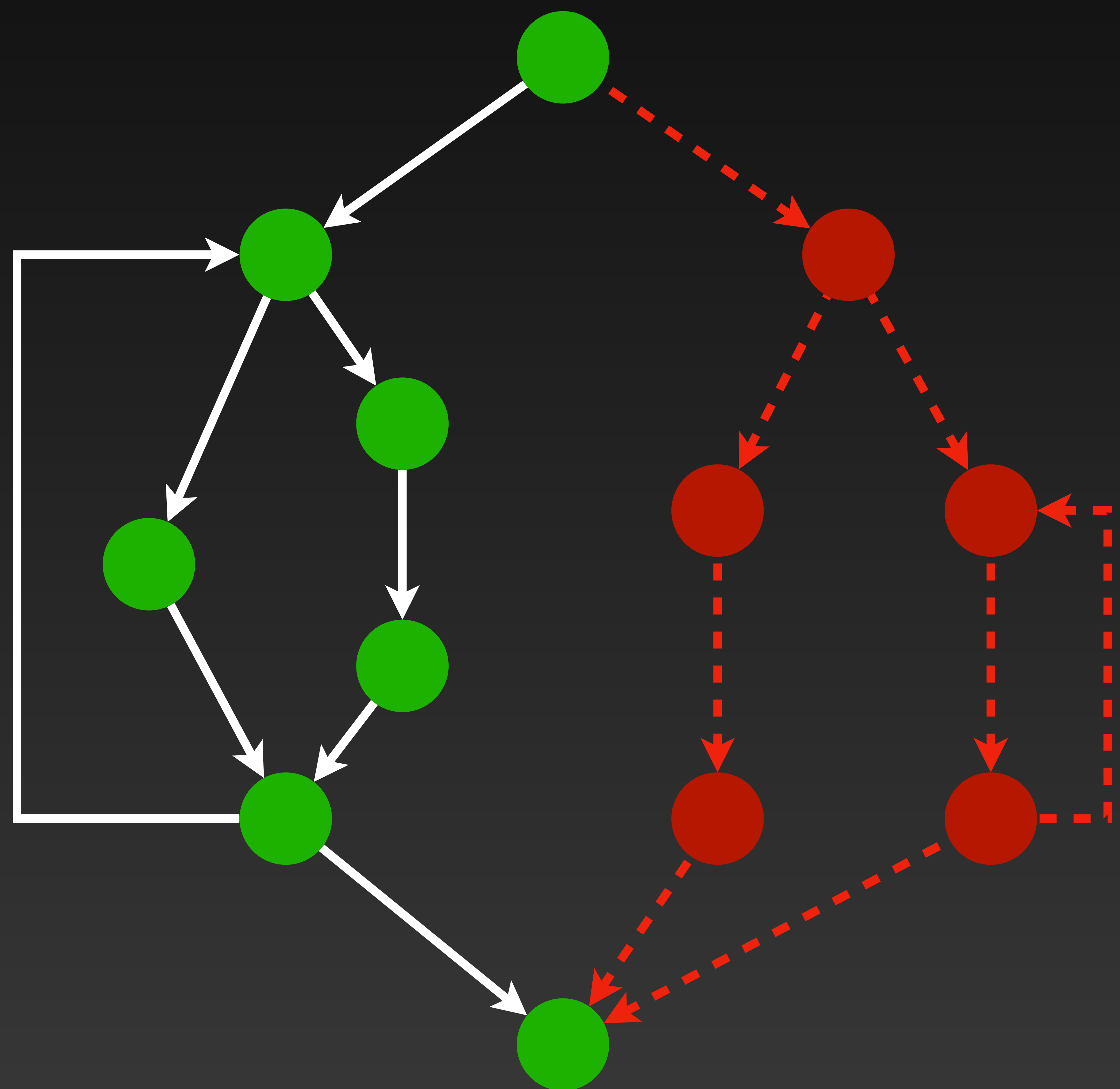
该输入下的“死”代码

概输入下输出 O

EMI

- Mutate

给定一次输入 I



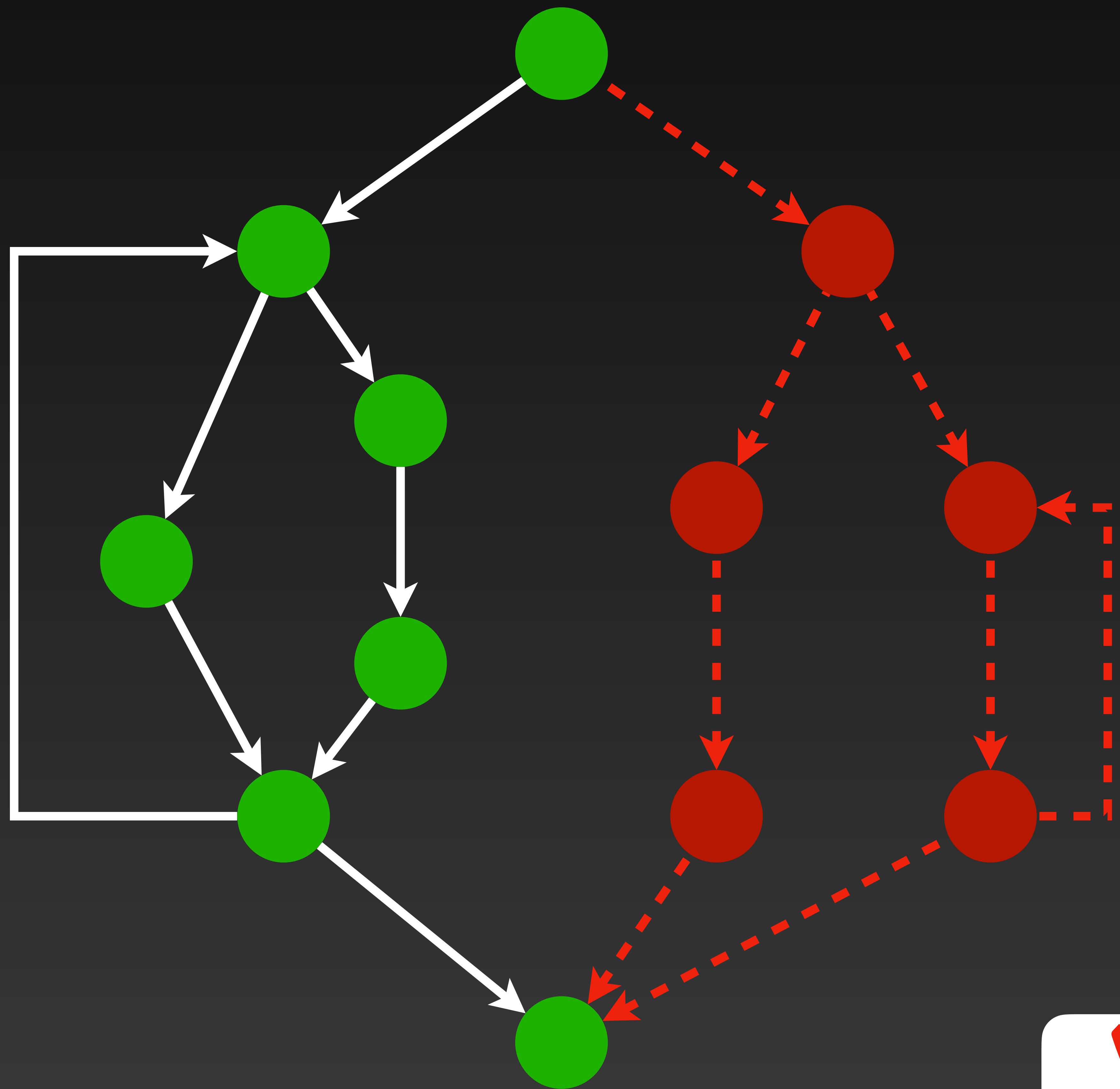
概输入下输出O

equivalent modulo I

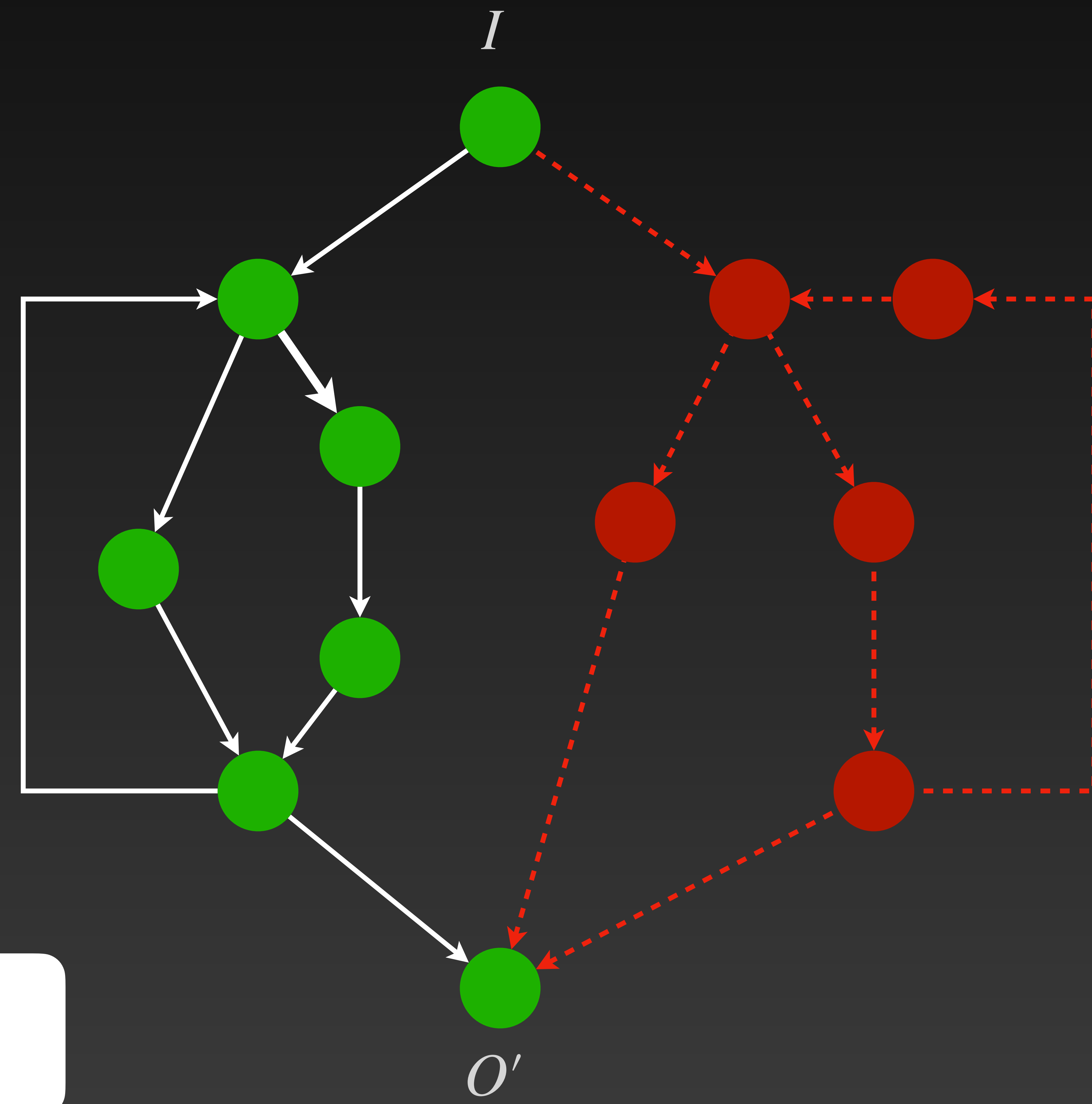
EMI

- Find bugs

equivalent modulo I



$O \neq O'$



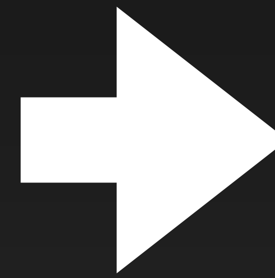
例子

LLVM bug

```c
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) abort();
    if (x.d != 20) abort();
    if (x.e != 30) abort();
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) abort();
    if (z.c != 12) abort();
    if (z.d != 22) abort();
    if (z.e != 32) abort();
    if (l != 123) abort();
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

Unexecuted

$ clang -m32 -O1 test.c ; ./a.out



```c
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) /* deleted */;
    if (x.d != 20) abort();
    if (x.e != 30) /* deleted */;
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) /* deleted */;
    if (z.c != 12) abort();
    if (z.d != 22) /* deleted */;
    if (z.e != 32) abort();
    if (l != 123) /* deleted */;
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

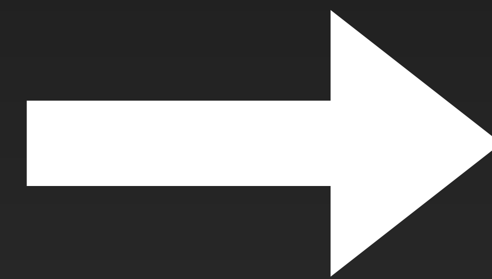
$ clang -m32 -O1 test.c ; ./a.out Aborted!

例子

- Reduced

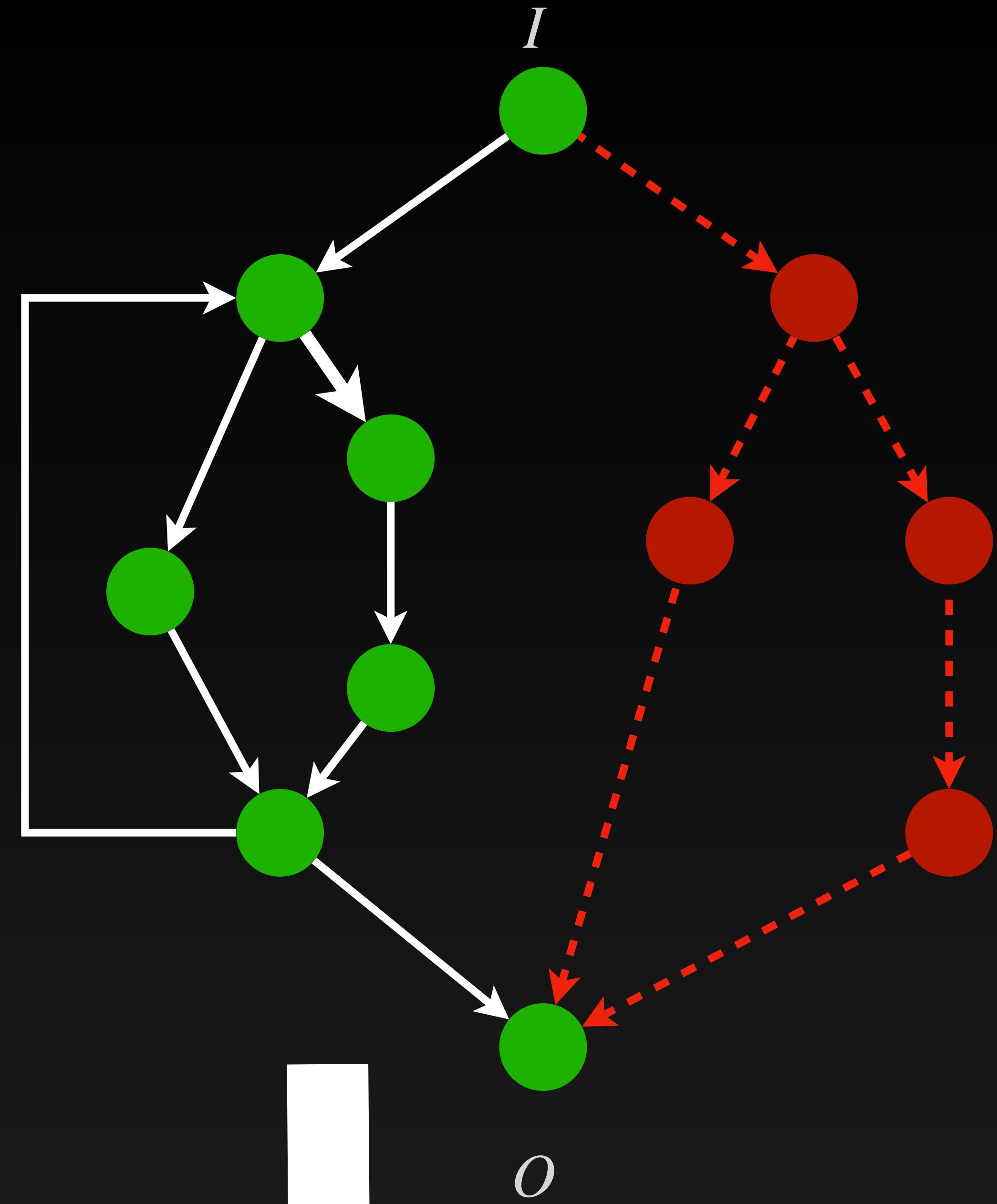
```c
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) /* deleted */;
    if (x.d != 20) abort();
    if (x.e != 30) /* deleted */;
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) /* deleted */;
    if (z.c != 12) abort();
    if (z.d != 22) /* deleted */;
    if (z.e != 32) abort();
    if (l != 123) /* deleted */;
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

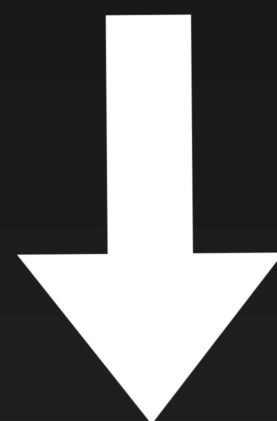
$ clang -m32 -O1 test.c ; ./a.out Aborted!

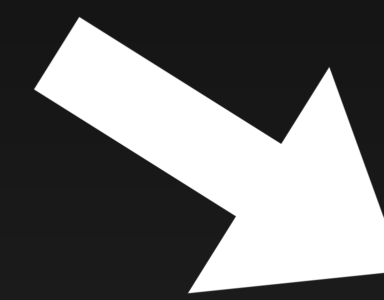


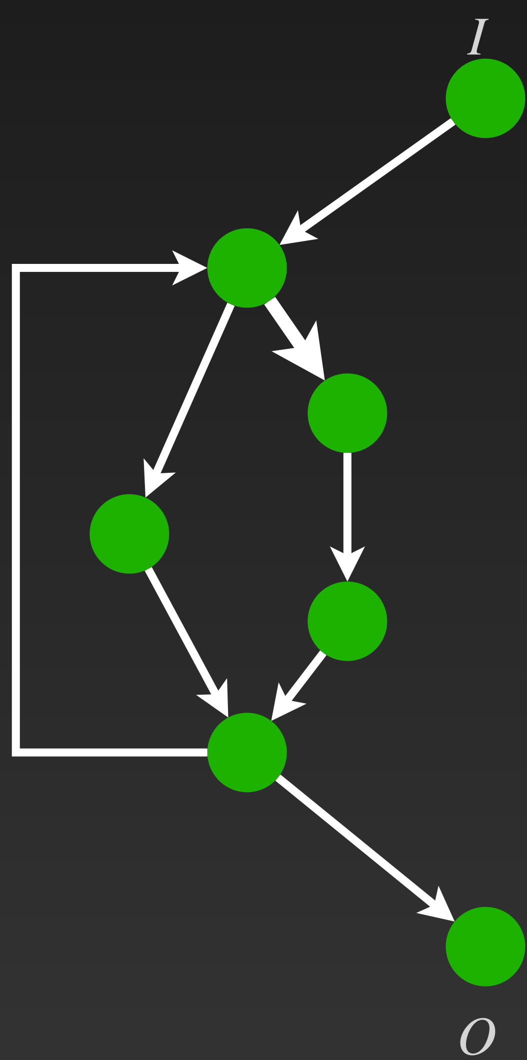
```c
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

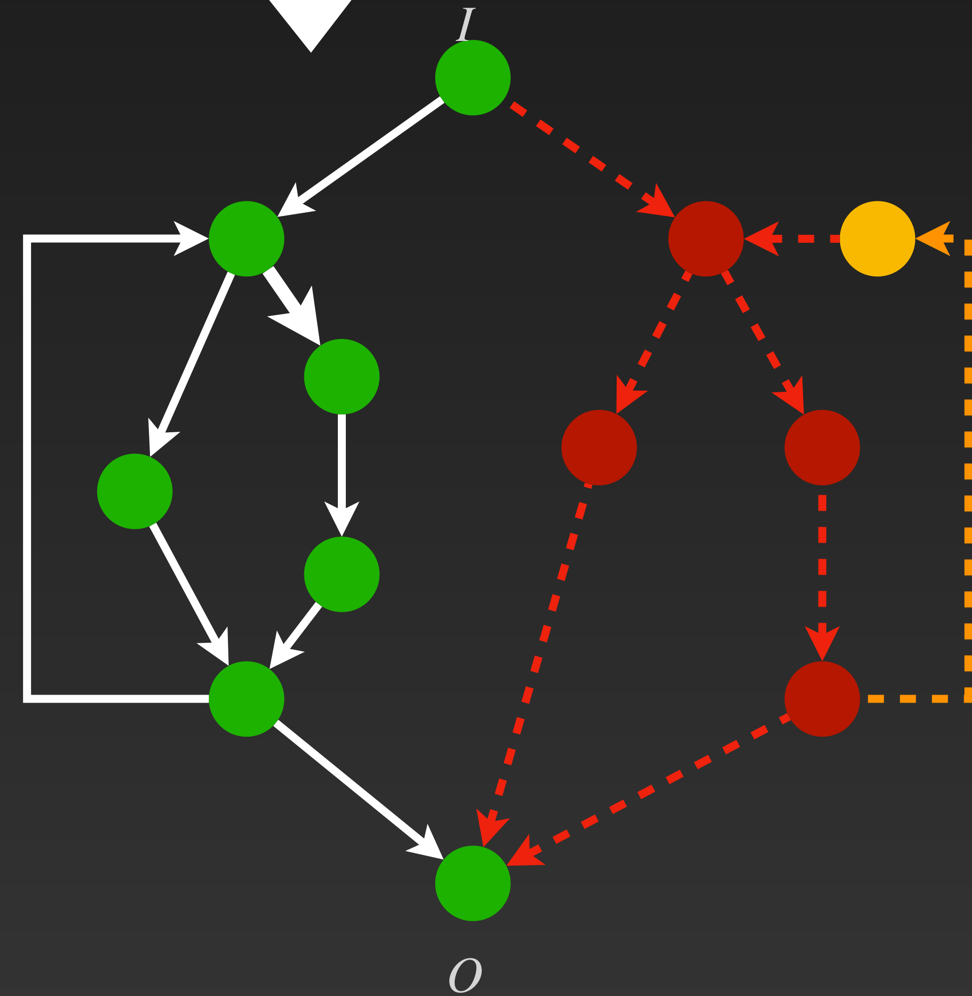
$ clang -m32 -O1 test.c ; ./a.out Aborted!

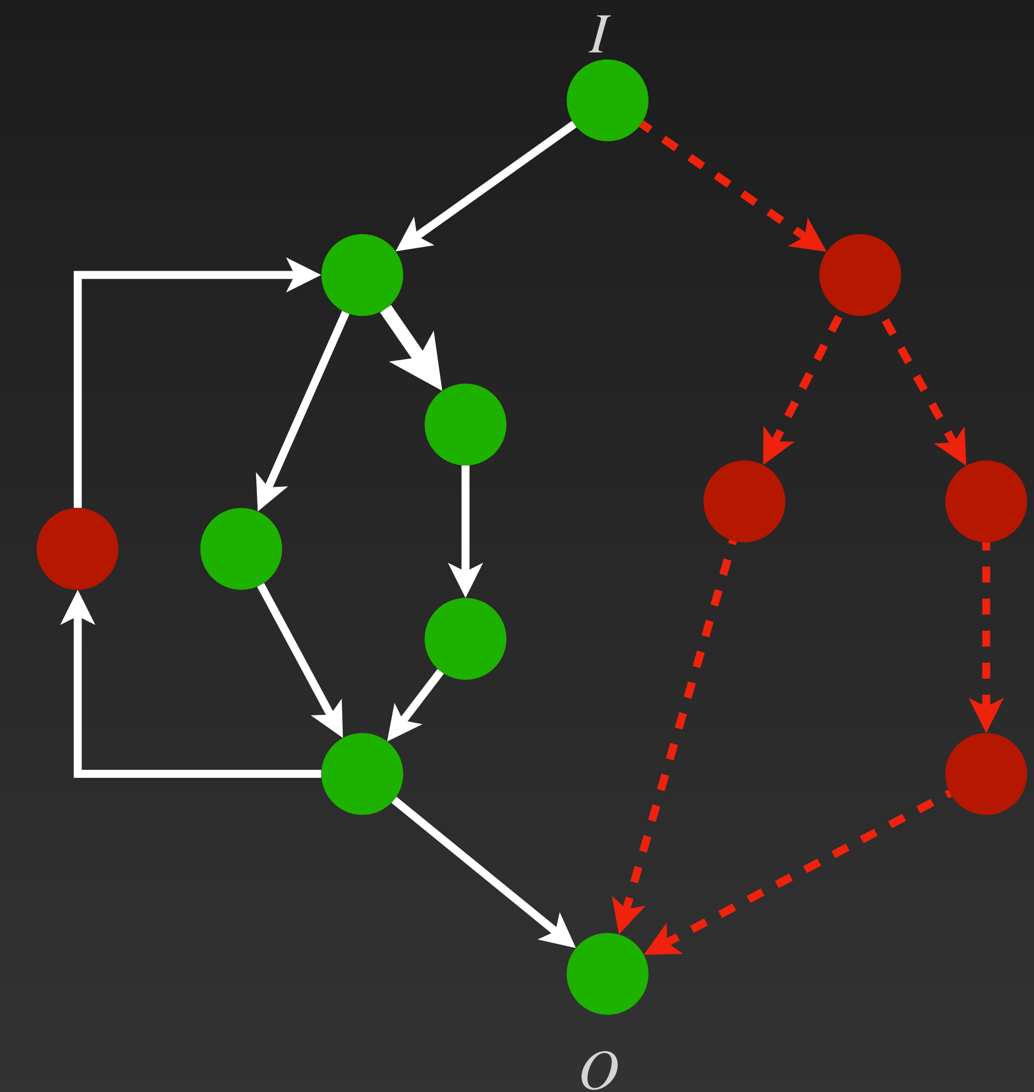












Orion (PLDI'14)
Prune dead code

Athena (OOPSLA'15)
Prune & inject dead code

Hermes (OOPSLA'16)
Mutate live code

为何影响力高?

- 首先，思想简单易懂（这个领域的人更加喜欢优雅、简单、实用的思路，对于那些看起来很高大上，很开门，但没什么用的想法不太感冒），而且具有很大的普适性!
- 第二，实验扎实，其每个工作都在服务器上连续跑了几个月
  - 平均每个工作可以为GCC、LLVM等编译器发现100多个bugs

LLVM 3.9 & 4.0 Release Notes

"… thanks to Zhendong Su and his team whose fuzz testing prevented many bugs going into the release …"

GCC's list of contributors

https://gcc.gnu.org/onlinedocs//gcc/Contributors.html

"Zhendong Su … for reporting numerous bugs"
"Chengnian Sun … for reporting numerous bugs"
"Qirun Zhang … for reporting numerous bugs"

当读这篇文章时，我发现

3.2.1 Extracting Coverage Information

Code coverage tools compute how frequently a program's statements execute during its profiled runs on some sample inputs. We can conveniently leverage such tools to track the executed (*i.e.* "covered") and unexecuted (*i.e.* "dead") statements of our test program *P* under input set *I*. Those statements marked "dead" are candidates for pruning in generating *P*'s EMI variants.

In particular, Orion uses `gcov` [7], a mature utility in the GNU Compiler Collection, to extract coverage information. We enable `gcov` by compiling the test program *P* with the following flag:

"`-O0 -coverage`"

which instruments *P* with additional code to collect coverage information at runtime. Orion then executes the instrumented executable on the provided input set *I* to obtain coverage files with information indicating how many times a source line has been executed.

Because `gcov` profiles coverage at the line level, it may produce imprecise results when multiple statements are on a single line. For example, in the example below,

```
if (false) { /* this could be removed */ }
```

`gcov` marks the entire line as executed. As a result, Orion cannot mutate it, although the statements within the curly braces could be safely removed. Note that we manually formatted the two test cases in Figure 1 for presentation. The actual code has every "`abort();`" on a separate line.

Occasionally, coverage information computed by `gcov` can also be ambiguous. For instance, in the sample snippet below (extracted from the source code of the Mathomatic[4] computer algebra system), `gcov` marks line 2613 as unexecuted (indicated by prepending the leading "#####"):

```
#####: 2613:  for (;; cp = skip_param(cp)) {
                  .....
    7: 2622:        break;
#####: 2623:  }
```

- profile工具gcov并不完美
  - 本质上source code和编译后的instructions存在gap
- 当时想着有没有可能对Gcov做点工作
  - 然而，是的，已经有这样的工作了😅

Uncovering Bugs in Code Coverage Profilers via Control Flow Constraint Solving Yang Wang, Peng Zhang, Maolin Sun, Zeyu Lu, Yibiao Yang, Yutian Tang, Junyan Qian, Zhi Li, Yuming Zhou IEEE Transactions on Software Engineering (TSE), 2023

Automatic Self-Validation for Code Coverage Profilers Yibiao Yang, Yanyan Jiang, Zhiqiang Zuo, Yang Wang, Hao Sun, Hongmin Lu, Yuming Zhou, Baowen Xu In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019

数据库

数据库的重要性

- 数据库的正确性的重要性不言而喻，关乎个人隐私数据、财产数据等等

"… seems likely that there are over one trillion (1e12) SQLite databases in active use …"

SQLite (~150,000 LOC) has 662 times as much test code as source code

SQLite's test cases achieve 100% branch test coverage

SQLite is extensively fuzzed (e.g., by Google's OS-Fuzz Project)

logic bugs

- 数据库系统之前的测试往往测试奔溃性错误
  - 这是最为直接和简单的Oracle
- 但是功能性的故障难以测试
  - 本质上还是缺少Oracle
  - 或者，本质上没有对待测软件进行深刻的剖析，从而得到其专有的性质

SQLite 3 bug

t0

c0
0
1
2
NULL

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

是一个"null-safe"的比较操作符

SQLite 3 bug

t0

c0
0
1
2
NULL

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

0 is not 1 $\Longrightarrow$ TRUE

SQLite 3 bug

t0

c0
0
1
2
NULL

```sql
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

1 is not 1 $\Longrightarrow$ False

SQLite 3 bug

t0

c0
0
1
2
NULL

```sql
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

2 is not 1 $\Longrightarrow$ TRUE

SQLite 3 bug

t0

c0
0
1
2
NULL

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

NULL is not 1 $\Longrightarrow$?

should be true,
but the result set has no NULL, a bug!

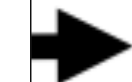
Differential Testing?

```
SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;
```

Check all DBMSs compute the **same result**: $RS_1 = RS_2 = RS_3$

PostgreSQL

RS_1



Syntax error

RS_2

Syntax error

RS_3

`{0, 2}`

每个数据库引擎有自己的独特的SQL语法，这些语法对于其他数据库而言相当于“未定义行为”

比如：**CREATE TABLE t0(c0);** 这一句对**SQLite**是合法的，单对于其他两个数据库非法，必须指定数据类型

A tool: SQLancer

- Generate a database
- Generate a query
- Validate the query's result

构造测试用例

构造Oracle

Pivoted Query Synthesis (PQS)

发现了数据库100多个bugs

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

Rigger M, Su Z. Testing database engines via pivoted query synthesis[C]//14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). 2020: 667-682.

Pivoted Query Synthesis (PQS)

随机生成一个数据库

t0
c0
0
1
2
NULL

随机挑选一行作为基准行

t0
c0
0
1
2
NULL

为其生成一个query

SELECT c0 FROM t0 WHERE ________________ ?

Pivoted Query Synthesis (PQS)

- 这个Query和这个Pivot Row产生关联：
  - 比如对于这个Row，这个Query里的谓词应该返回为True
    - 对于一个的Query，很容易可以实现一个表达式来判读该Row和这个Query是否一致（一般来说，验证是容易的）
  - 当然，如果返回为False，也没关系
    - 关键是，要提前获知这两者之间的关联

Pivoted Query Synthesis (PQS)

- 验证：
  - 对于一个提前验证为True的query
    - 检查其结果是否含有那个Pivot row，没有就是出错！
  - 对于一个提前验证为False的query
    - 检查其结果是否含有那个Pivot row，有就是出错！

t0

c0
0
1
2
NULL

```
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

对于pivot 应该是True
但结果集里没有这个，bug

更多后续工作

Detecting optimization bugs by rewriting the query so that it cannot be optimized.

Rigger M, Su Z. Detecting optimization bugs in database engines via non-optimizing reference engine construction[C]//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020: 1140-1152.

Partition the query into several **partitioning queries**, which are then composed.

Rigger M, Su Z. Finding bugs in database systems via query partitioning[J]. Proceedings of the ACM on Programming Languages, 2020, 4(OOPSLA): 1-30.

巨大的影响力

	#Bugs		
DBMS	**Logic**	**Error**	**Crash**
CockroachDB	16	46	5
DuckDB	29	13	31
H2	2	15	1
MariaDB	5	0	1
MySQL	21	9	1
PostgreSQL	1	11	5
SQLite	93	44	42
TiDB	30	27	4
Total	**197**	**165**	**90**

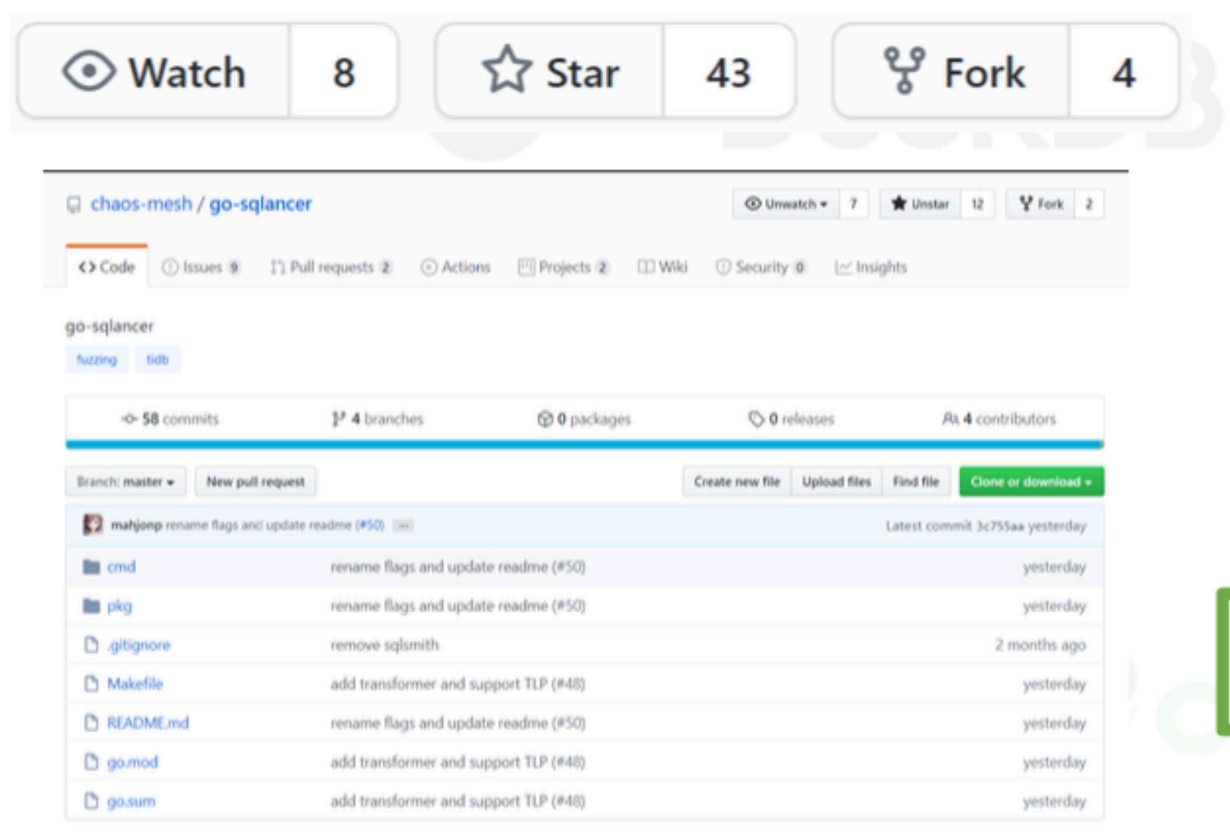
452 reported **379** fixed

巨大的影响力

✓	# 5677.15	AMD64	Bionic	SQLancer	4
✓	# 5677.16	AMD64	Bionic	SQLancer (with Address Sanitizer)	2

DuckDB runs SQLancer on every pull request

Watch 8 Star 43 Fork 4



https://github.com/chaos-mesh/go-sqlancer

PingCAP implemented go-sqlancer

"With the help of SQLancer … we have been able to identify **>100 potential problems** in corner cases of the SQL processor"

SMT solvers

可满足性模理论求解器

Satisfiability Modulo Theories (SMT)

部分内容来自熊英飞老师软件分析课程

- SAT（可满足性）问题回答某个命题逻辑公式的可满足性，如：
  - $A \wedge B \vee \neg C$，其中A, B, C都是bool变量
- 但实际应用中的公式更为复杂：
  - $a + b < c \wedge f(b) > c \vee c > 0$
- 从逻辑学角度，$a + b < c$ 或者 $f(b) > c$都是逻辑系统中不包含的符号，需要知道这些东西的意思

Satisfiability Modulo Theories (SMT)

- 在SMT中，理论(Theory)用于对上述这类符号赋予含义
  - 其包含一组公理和这组公理能推导出的结论
- 满足性模理论Satisfiability Modulo Theories
  - 给定一组理论，根据给定背景逻辑，求在该组理论解释下公式的可满足性

常见理论举例：EUP

- Equality with Uninterpreted Functions
- 公理：
  - $a_i = b_i \implies f(a_1, \ldots a_n) = f(b_1, \ldots, b_n)$
  - $a = b \iff \neg(a \neq b)$
- 如： $a * \big(f(b) + f(c)\big) = d \wedge b * \big(f(a) + f(c)\big) \neq d \wedge a = b$
  - $f, *, +$都可以看成是未定义的函数
  - 可以直接推出矛盾（无解）

常见理论举例

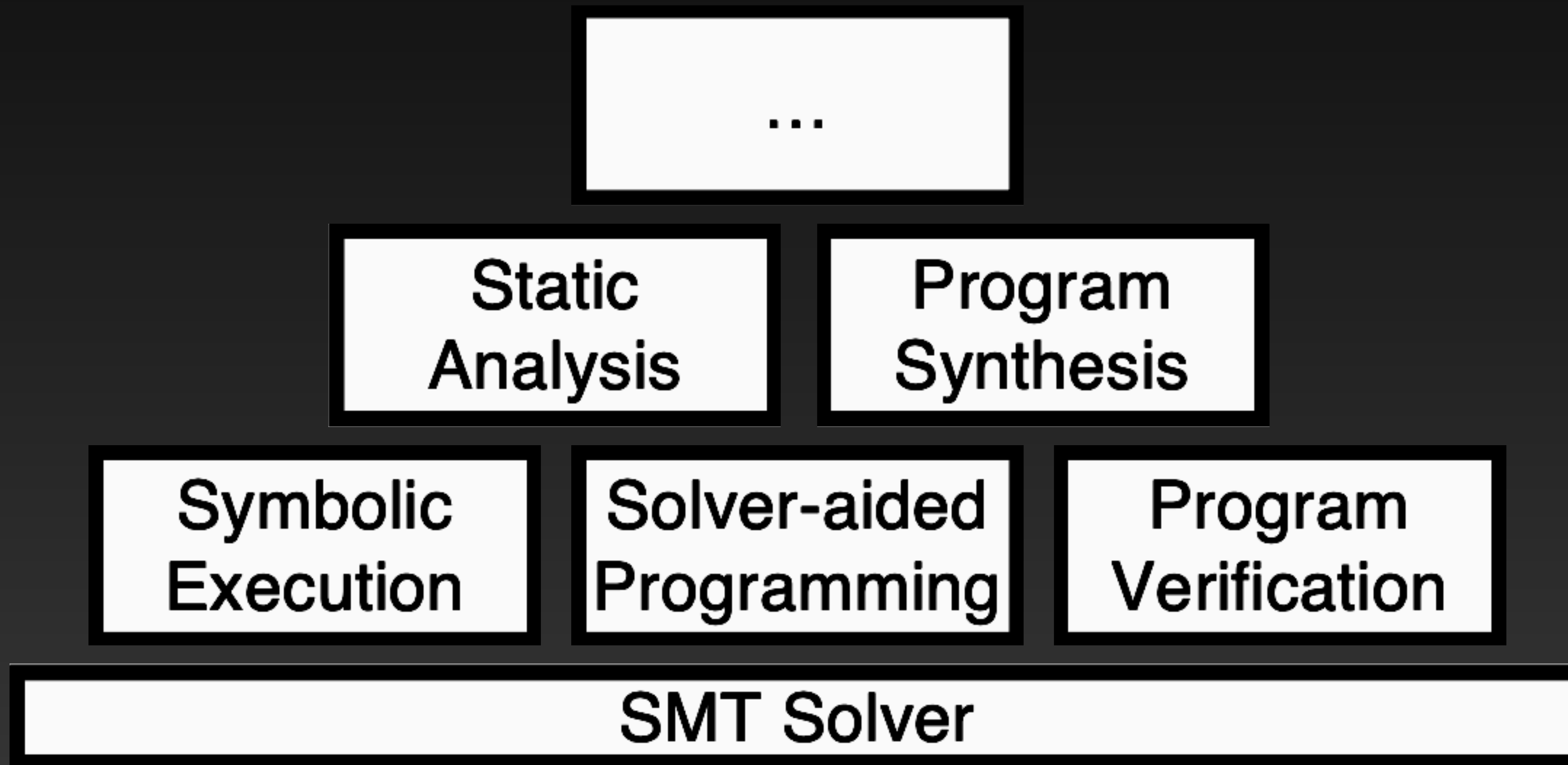
- 算术
  - $a + 10 < b$
  - $2x + 3y + 4z = 10$
- 数组
  - read(write(a, i, v), i) = v
- 位向量Bit Vectors
  - $a[0] = b[1] \wedge a = c \wedge b[1] \neq c[0]$

SMT的一些解法

- 70、80年代：出现了基本算法混合不同理论，但求解能力有限
- 2000年前后：SAT （NPC问题）速度大幅提升，很多算法渐渐将SMT转换为SAT问题求解
  - 比如$f(a) = c \wedge f(b) \neq c \wedge a \neq b$，引入符号代替函数：
    - $A = c \wedge B \neq c \wedge a \neq b$
    - $a = b \implies A = B$
  - 进一步，引入布尔变量$P_{A=c} \wedge \neg P_{B=C} \wedge P_{a \neq b}, P_{a=b} \implies P_{A=B}$
    - 同时需要一些传递性约束：$P_{A=C} \wedge P_{B=C} \implies P_{A=B}, P_{A=B} \wedge P_{B=C} \implies P_{A=C}, \ldots$
  - 然后利用算法DPLL/CDCL来进行求解
- SMT求解还有比赛https://smt-comp.github.io/

当然，直接转化为SAT难以规模化，而且难以利用很多已有算法（如线性规划等高效算法），所以现在往往采用lazy的算法（Lazy Satisfiability Modulo Theories）

SMT是很多CS任务的基础



地基不牢，地动山摇

回到SMT solver的测试

```
(declare-fun a () Real)
(declare-fun p () Real)
(declare-fun b () Real)
(declare-fun c () Real)
(declare-fun d () Real)
(declare-fun k () Real)
(declare-fun e () Real)
(declare-fun q () Real)
(assert (or
            (not (exists ((f Real))
                (=>
                (and
                    (>= c 0)
                    (> (/ b q) 2)
                    (>= (/ p q) 1)
                    (<= d 12)
                    (>= (/ p q) (- (* 1 k)))
                    (<= (/ p q) (+ 10 k)))
                (<= (+ (* (- 2) (- a e)) d) 12))))
            (exists ((o Real))
                (forall ((g Real))
                    (exists ((h Real))
                        (and
                            (or
                                (>= g (* (- 3) h) 57)
                                (and (> (* 79 o) 8 (+ g h) 0) (= h 0))
                                (< 0 (+ g h) 0))
                            (> (+ (* (- 97) o) g) 0)))))))
(assert (= a (+ c e)(* d q)(/ b q)))
(assert (= q (/ b k)))
(check-sat)
(get-model)
```

Z3 solver等现实的SMT solver有相应的语言，有相应的文法，就能fuzzing，但是除了crash faults之外，logic bugs怎么发现?

Semantic Fusion

语义融合

- Fusing formulas while preserving satisfiability
  - $\varphi_1 = x > 0 \wedge x > 1$ SAT
  - $\varphi_2 = y < 0 \wedge y < 1$ SAT
  - $\varphi_{concat} = \varphi_1 \wedge \varphi_2$?
- Finding bugs in state-of-the-art SMT solvers, 46 confirmed, 42 fixed bugs in Z3 and CVC4

Dominik Winterer, Chengyu Zhang, and Zhendong Su. 2020. Validating SMT solvers via semantic fusion. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020).

如何构造更加复杂的融合?

- $\varphi_1 = x > 0 \wedge x > 1, \varphi_2 = y < 0 \wedge y < 1$
- $\varphi_{concat} = x > 0 \wedge x > 1 \wedge y < 0 \wedge y < 1$ SAT

$x = 2, y = -2$

- $\varphi_{concat} = x > 0 \wedge x > 1 \wedge y < 0 \wedge y < 1$

z

引入一个新的变量，让其和x, y产生关联

$\varphi_{concat} = x > 0 \wedge x > 1 \wedge y < 0 \wedge y < 1$

我们可以令$z = x + y$，但这个式子（fusion function）不给solver获知，然后将其中一个x替换为$z - y$, 一个y替换为$z - x$

$\varphi_{concat} = x > 0 \wedge z - y > 1 \wedge z - x < 0 \wedge y < 1$

灵感来自：密码学?

- $\varphi_{concat} = x > 0 \wedge z - y > 1 \wedge z - x < 0 \wedge y < 1$
  - 显然有解，因为我们知道$x = 2, y = -2, z = x + y = 0$一定是一组解
  - 这里我的感觉是灵感来自密码学
    - 通过x, y可以很容易构造z, 联合的语义也必然有解
      - 但是对于solver而言从z解出其和x, y的关系却没那么简单，因为搜索空间变大了

Unsat也可以通过类似操作

- $\varphi_1 = x < 0 \land x > 1$ UNSAT
- $\varphi_2 = y < 0 \land y > 1$ UNSAT
- $\varphi_{concat} = \varphi_1 \lor \varphi_2$
  - 这里或操作比与操作更强
- 进一步进行融合操作：
  - $x < 0 \land z - y > 1 \lor y < 0 \land z - x > 1 \land z = x + y$ UNSAT

后续工作

- Differential + Operator mutation

```
$ cat > formula.smt2
(assert (forall ((a Int))
        (exist  ((b Int))
        (distinct (* 2 b) a))))
(check-sat)

$ cvc4 formula.smt2
sat

$ z3 formula.smt2
sat
```

```
$ cat > bug.smt2
(assert (forall ((a Int))
        (exist  ((b Int))
        (= (* 2 b) a))))
(check-sat)

$ cvc4 bug.smt2
unsat

$ z3 bug.smt2
sat    ✗
```

Winterer D, Zhang C, Su Z. On the unusual effectiveness of type-aware operator mutations for testing SMT solvers[J]. Proceedings of the ACM on Programming Languages, 2020, 4(OOPSLA): 1-25.

Q&A