Verification

Implementation meets specification

Mo Zou

Software is error-prone

- What is the best comment in source code you have ever encountered?
  - Top answers from *stackoverflow*

```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows
```

```
// drunk, fix later
```

```
// Magic. Do not touch.
```

Bugs are difficult to detect and fix

- "Program testing can be used to show the presence of bugs, but never to show their absence! "---Edsger W. Dijkstra
- A Ph.D.'s friend cycle

两年前在开发CloudVisor-D时遇到过一个非常难以解决的bug，当时委托一位同学专门研究了这个问题，可是最后花了两三个月也没有太多进展。此bug困难处在于它的随机性，它会造成多VCPU虚拟机的卡死，而且每次卡死的位置都不同。限于当时的调试手段，我们只能加打印研究现象。然而一旦加了打印，这个bug就意外地转移到了其他地方并表现出不同的行为。由于时间紧张，当时只能遗憾地选择战略绕开这个bug。

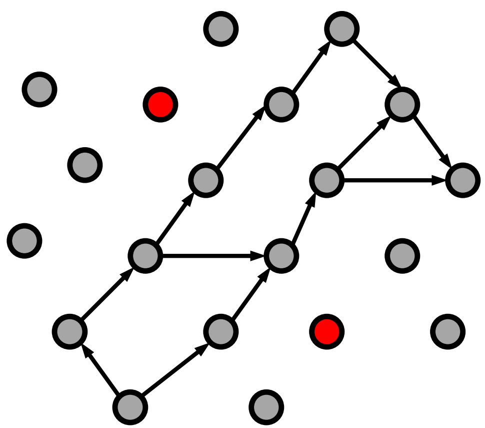
万万没想到的是，两年后我又遇到了同样的bug。不同的是，这次我们有了比较高级的调试工具，能够在不加打印的情况下观察软件和硬件的状态。所以我再次鼓起勇气，花了四五十个小时，一步步跟踪分析，从底层firmware到中间的hypervisor，再追到虚拟机内核，上上下下跟了好几个来回，终于定位到了问题所在（VCPU核间中断）。bug产生的位置距离最终虚拟机卡死的位置早已经过了不知道多少指令。

当时委托一位同学专门研究了这个问题，可是最后花了两三个月也没有太多进展。此bug的困难处在于它的随机性

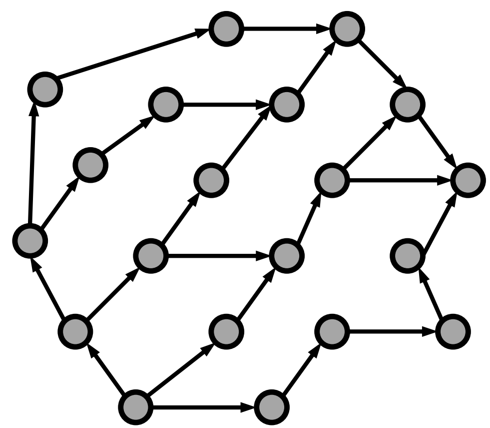
所以我再次鼓起勇气，花了四五十个小时，一步步跟踪分析*[...]*终于定位到了问题所在

Do we have a way to ensure the correctness?

- Formal verification
  - the only known way to guarantee that a software is free of programming errors



Software testing



Formal verification

Mathematical methods

Fully verify that impl meets spec

A beginner’s question/misunderstanding about verification

- 现有技术能够证明一个软件正确吗？
- 证明软件基于什么理论？
- 有实际软件是证明过的吗？
- 证明能保证软件完全正确。
- 证明需要很深的数学基础，我应该理解不了。
- 证明跟我没有关系，我以后也不会去做证明。

Outline

- What is verification
- A brief history of state-based verification
- Verification techniques
- Proof automation
- Verification at academia and industry

WHAT IS VERIFICATION

What to prove about a program

```
int x;
inc()
   x = x + 1;
```

Operating system kernel

- Safety property (functional correctness & invariant)---程序功能是否正确
- Liveness property---程序是否终止
- Security property---程序是否安全
- Resource usage property---资源使用情况
- Real-time property---时延情况
- Atomicity, crash-safety, etc.

Specification describes what is desired

- Specification is not unfamiliar to us
  - Comments in the code
  - Natural language document
- Informal specification cannot be used in proofs

open(2) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ERRORS | VERSIONS | CONFORMING TO | NOTES | BUGS | SEE ALSO | COLOPHON

```
int x;
// inc() increments the value of x by 1
inc()
  x = x + 1;
```

Formal specification

- A category of specification that have rigorous meaning
- Specification language
  - Math
  - Any language that is mathematically defined
- E.g., type system

```
int x;
// inc() increments the value of x by 1
inc()
   x = x + 1;
```

Formal specification

Precondition: x = N

Postcondition: x = N + 1

Implementation

- Programs implemented in a programming language
  - E.g., C program
- However, C has undefined behavior

```
i = i++  +  1;
```

```
int f() { }
```

- Programming language should also be mathematically defined---i.e., defining the semantics
  - Develop a mathematical model of C subset
  - Exclude undefined behavior

What does impl. meet spec. mean?

Implementation

Impl. meet spec.

Specification

假如前置条件满足，执行完程序，后置条件被满足

前置后置条件

假如x=N，执行完inc()，x=N+1

Pre: x = N

Post: x = N +1

Some C program

inc()

What does impl. meet spec. mean?

Correctness condition

Implementation	Impl. meet spec.	Specification
	假如前置条件满足，执行完程序，后置条件被满足	前置后置条件
Some C program	程序执行前满足不变式，程序每执行一步都满足不变式	不变式
	假如前置条件满足，程序能够执行完并且后置条件满足	前置后置条件+终止性
	程序每执行一步，假如发生崩溃，状态仍然一致	崩溃一致性

Correctness condition defines what impl. meets spec. means

Implementation	Correctness condition	Specification
I: A math model	Math definition of I meets S	S: A math model

- Different correctness condition for different specification
- Two most used specification & correctness condition for functional correctness---details later

Proof establishes correctness condition

Implementation | Correctness condition | Specification

I: A math model

Math definition of
I meets S

S: A math model

Proof

- Proof needs to be machine-checked
- Automatic proof technique

Summary: four parts to verification

- Specification: a precise description of the desired behavior of a system
- Implementation: an implementation of a system
- Correctness condition: what “impl meets spec” means
- Proof: mathematical proof showing implementation meets specification

A BRIEF HISTORY OF STATE-BASED VERIFICATION

Specifying sequential correctness

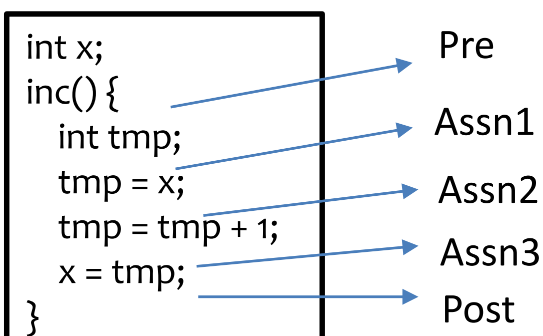
Correctness for sequential programs

1967

- First proposed by Robert W. Floyd in 1967
  - Receive Turing Award in 1978

Specifying sequential correctness

- Annotate each control point with an assertion



Proof decomposed into verification of each statement

E.g., Assn_i Command_i Assn_{i+1}

A logical framework for proving sequential program

Correctness for sequential programs

1967

1969

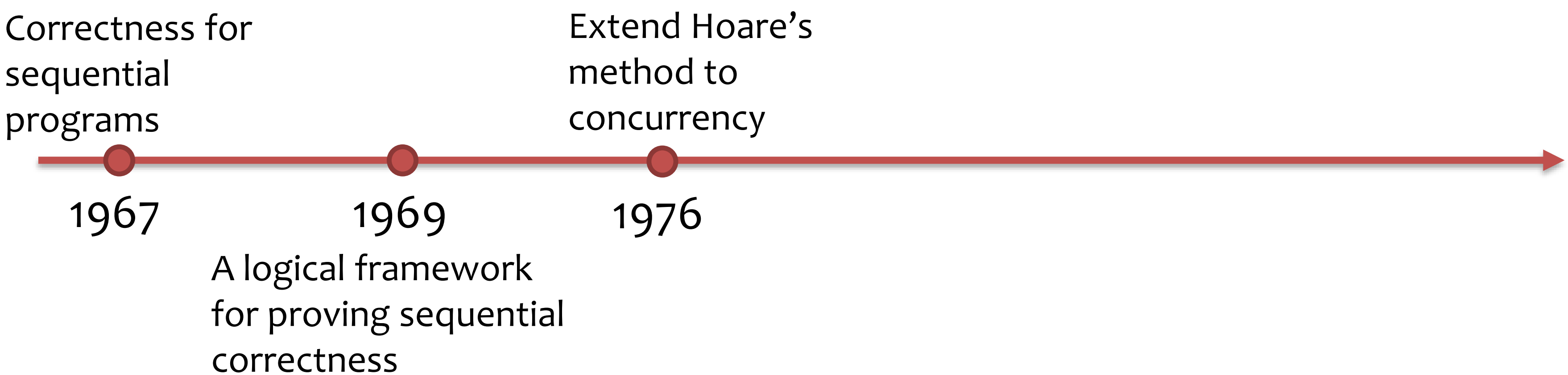
A logical framework for proving sequential correctness

- Tony Hoare recast Floyd's method into a logical framework, known as Hoare Logic (or Floyd-Hoare Logic)
  - Receive Turing Award in 1980
- Formula in Hoare Logic: P {S} Q
- Inference rules for proving

Floyd and Hoare changed the way to think of programs

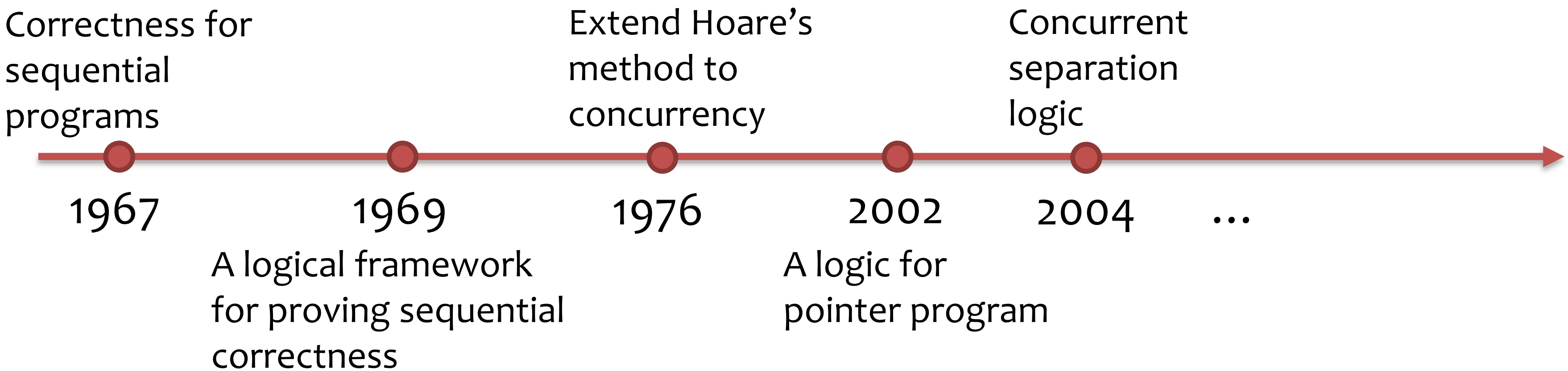
- Not as an event generator, but as a state transformer
- State are close to everyday mathematics
    - Described in terms of numbers, sequences, sets, functions and so on
        - E.g., C struct --> sequence
        - E.g., C map --> function
- Reduce program reasoning into mathematical reasoning

History of state-based verification



- A long period of time without much progress
- Because state model only has stores
  - Store: mapping of variable to values
  - Can't reason about pointer program, i.e., program with heap

History of state-based verification



- Separation logic (2002): logic for pointer program
- Concurrent separation logic (2004)
- After that, theories get more and more complete for practical systems

VERIFICATION TECHNIQUES

How would you prove that this program is functional correct

```
int x;
inc()
   x = x + 1;
```

1. Give a math model of the implementation
2. Give a mathematically defined specification
3. Define correctness condition
4. Prove correctness condition

Modelling programming language

- Syntactics (语法)
- State model (状态模型)
- Semantics (语义)

Expressions:

$$E ::= \; N \mid V \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times \; E_2 \mid \ldots$$

Boolean expressions:

$$B ::= \; \mathtt{T} \mid \mathtt{F} \mid E_1 {=} E_2 \mid E_1 \leq E_2 \mid \ldots$$

Commands:

$$\begin{array}{ll} C ::= & V \;\mathtt{:=}\; E \\ \quad \mid & C_1 \;\mathtt{;}\; C_2 \\ \quad \mid & \mathtt{IF}\; B \;\mathtt{THEN}\; C_1 \;\mathtt{ELSE}\; C_2 \\ \quad \mid & \mathtt{WHILE}\; B \;\mathtt{DO}\; C \end{array}$$

Syntactics for a toy language

Modelling programming language

- Syntactics (语法)
- State model (状态模型): state that the language operate
- Semantics (语义)

	A toy language	C language
State model	变量名到值的映射（E.g., X -> 1）	全局符号表，局部符号表，内存（地址到值的映射）

Modelling programming language

- Syntactics (语法)
- State model (状态模型)
- Semantics (语义): how program modifies the state

	A toy language	C language
Semantics of V := E (E.g., x = x +1)	st1 $\xrightarrow{V:=E}$ st2 is defined as st2 = st1[x -> eval(E, st1)] (E.g., st1 = x -> N, st2 = x -> N+1)	1. 计算右值E 2. 取左边变量的地址addr 3. 将E赋给addr

Using Hoare Triple as specification

- Formal reasoning about program correctness using pre- and postconditions
- Pre- and postconditions are assertions
  - E.g., $x = 1$, $\exists n.\ x = n$
- Semantics of assertions
  - Assertion holds on state
  - $x = 1$ actually means st $x = 1$

Using Hoare Triple as correctness condition

- Syntax: {P} S {Q}
  - P and Q are assertions
  - S is a program
- If we start in a state where P is satisfied and execute S, if S terminates in a state, then Q holds on this state
- Semantics of {P} S {Q}:
  - Correct(P, S, Q) = forall st1 st2, P st1 -> st1 $\xrightarrow{S}$ st2 -> Q st2.

How would you prove that this program is functional correct

```
int x;
inc()
   x = x + 1;
```

$\text{P}(st) \triangleq st\ x = n$

$\text{Q}(st) \triangleq st\ x = n + 1$

$\text{S} \triangleq x = x + 1;$

Goal: Correct(P, S, Q) = forall st1 st2, P st1 -> st1 $\xrightarrow{S}$ st2 -> Q st2

$$\text{Given} \quad \frac{\text{P st1} \qquad \text{st1} \xrightarrow{S} \text{st2}}{\text{Q st2}} \quad \text{Prove}$$

思路：

1.根据P st1以及st1 $\xrightarrow{S}$ st2得到st2

2.将st2带入Q st2进行证明

How would you prove that this program is functional correct

$$\text{Given} \quad \frac{P\ st1 \qquad st1 \xrightarrow{S} st2}{Q\ st2} \quad \text{Prove}$$

思路：

1.根据P st1以及$st1 \xrightarrow{S} st2$得到st2

2.将st2带入Q st2进行证明

$$\Downarrow$$

$$\text{Given} \quad \frac{st1\ x = n \qquad st2=st1[x\text{->}n+1]}{st2\ x=n+1} \quad \text{Prove}$$

$$\Downarrow$$

Prove st1[x->n+1] x=n+1

	A toy language
Semantics of V := E (E.g., x = x +1)	$st1 \xrightarrow{V:=E} st2$ is defined as st2 = st1[x -> eval(E, st1)] (E.g., st1 = x -> N, st2 = x -> N+1)

Hoare Logic: a logical framework for proving sequential program

- A set of inference rules for proving
- Syntactical proving instead of semantics proving

$$\frac{}{\{Q[E/x]\}\ x = E\ \{Q\}}\ (ASSN) \qquad \frac{\{P\}\ C_1\ \{R\} \quad \{R\}\ C_2\ \{Q\}}{\{P\}\ C_1;\ C_2\ \{Q\}}\ (SEQ)$$

$$\frac{P \Rightarrow P_1 \quad \{P_1\}\ C\ \{Q_1\} \quad Q_1 \Rightarrow Q}{\{P\}\ C\ \{Q\}}\ (CONSEQ)$$

Enrich program state with heap

- Heap: address points to value (addr $\mapsto$ value)
- Read from heap V := [X]
  - [X] is heap value from address X
- Write to heap [X] := E

```
int *x, y;
inc2()
   [x] = [x] + 1;
   [y] = [y] + 1;
```

Challenges with heap

- $\{\mathrm{x} \mapsto \mathrm{m} \land y \mapsto n\}\ [x] = 1\ \{\mathrm{x} \mapsto 1 \land y \mapsto n\}$ ✗
  - What if x and y refer to the same place
- $\{\mathrm{x} \mapsto \mathrm{m} \land y \mapsto n \land x \neq y\}\ [x] = 1\ \{\mathrm{x} \mapsto 1 \land y \mapsto n \land x \neq y\}$ ✓
  - Can we only mention program-related memory?
- Need to support pointer alias and local reasoning

Separation Logic

- Simplify the problem of specifying pre and post conditions by introducing new logical connectives
  - *: separating conjunction
  - * has $x \neq y$ built into it
  - now we can write $\{\mathrm{x} \mapsto m * y \mapsto n\}[\mathrm{x}] = 1\{\mathrm{x} \mapsto 1 * y \mapsto \mathrm{n}\}$

Separation conjunction allows local reasoning

- Frame rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \text{ where C doesn't modify free variables of R}$$

- Now to prove $\{\mathrm{x} \mapsto \mathrm{m} * y \mapsto n\}[\mathrm{x}] = 1\{\mathrm{x} \mapsto 1 * y \mapsto \mathrm{n}\}$
- We only need to prove $\{\mathrm{x} \mapsto \mathrm{m}\}[\mathrm{x}] = 1\{\mathrm{x} \mapsto 1\}$ and apply frame rule

Summary so far

- Hoare logic
  - Spec: {Pre} {Post}
  - Impl: S
  - Correctness condition: Correct(Pre, S, Post)
  - Proof: show Correct(Pre, S, Post) establishes
- Separation logic: extend Hoare logic with separation conjunction to reason about heaps

Limitation of Hoare Triple

- Not expressive enough
  - Concurrent interference makes assertion un-stable

```
int x;
inc-lf() {
   int done=0, tmp;
   while(!done){
      tmp = x;
      done = cas(&x, tmp, tmp+1);
   }
}
```

Wrong specification:

$\{x = N\}$ inc-lf() $\{x = N+1\}$

Weak specification:

$\{\exists N.\ x = N\}$ inc-lf() $\{\exists N.\ x = N+1\}$

Limitation of Hoare Triple

- Not expressive enough
  - Concurrent interference makes assertion un-stable
  - Some server program never ends

```
Precondition: P

while(true){
    …
}

Postcondition: False
```

Limitation of Hoare Triple

- Not expressive enough
  - Concurrent interference makes assertion un-stable
  - Some server program never ends
- Hoare Triple describes programs intentionally
- 不识庐山真面目，只缘身在此山中

Can we describe programs extensionally?

Observable behavior

- Character program from the **observable** behavior
  - Internal observer: program state
  - External observer: program output or abort ✓
- E.g., externally observable behavior of main is {(0,1)}

```
int x=0;
main()
   print(x);
   inc-lf();
   print(x);
```

Observable behavior (Cont)

- A trickier case: another thread running inc-lf()
  - Observable behavior is {(0,1), (0,2), (1,2)}
  - If also count prefix: {(0,1), (0,2), (1,2), (0), (1), Ø}

Given x = 0

```
thread1:
print(x);
inc-lf();
print(x);
```

```
thread2:
inc-lf();
```

What if another program produce the same behavior?

Another program with same observable behavior

Given x = 0

```
thread1:
print(x);
inc-lf();
print(x);
```

```
thread2:
inc-lf();
```

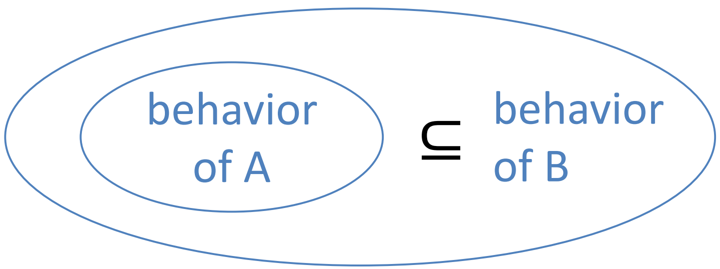
replace inc-lf() with

```
inc-lock()
    lock;
    x=x+1;
    unlock;
```

The new program can serve as the specification for the old program, and vice versa.

Refinement relation

- If program A has no more behavior than program B, than A refines B (A是B的一个精化)




```
int x=0;
main()
   print(x);
   inc-lf();
   print(x);
```

and

```
int x=0;
main()
   print(x);
   inc-lock();
   print(x);
```

refines each other

Refinement relation (Cont)

- If program A has no more behavior than program B, than A refines B (A是B的一个精化)
- Why no more---spec is more loose than impl
  - Spec: you can return x or y
  - Impl: I choose to return x

Refinement relation examples (1)

```
int x=0;
main()
    print(x);
    x = x+1;
    print(x);
```

```
int x=0;
main()
    print(x);
    x = x+1;
    print(x);
    print(x);
```

full trace	{(0,1)}	{(0,1,1)}
prefix trace	{(0,1), (0), Ø}	{(0,1), (0), Ø, (0,1,1)}

Depends on definition of behaviors

Refinement relation examples (2)

```
int x=0;
main()
    print(x);
    x = x+1;
    print(x);
```

```
int y=0;
main()
    print(y);
    y = y+1;
    print(y);
```

Implementation doesn't matter. Program A and B can use even different language

Refinement relation examples (3)

```
int x=0;
main()
    print(x);
    x = x+1;
    print(x);
```

```
int y=1;
main()
    print(y);
    y = y+1;
    print(y);
```

case (1)

```
int x=0;
main()
    print(x);
    x = x+1;
    print(x);
```

```
int y=2;
main()
    print(y-2);
    y = y-1;
    print(y);
```

case (2)

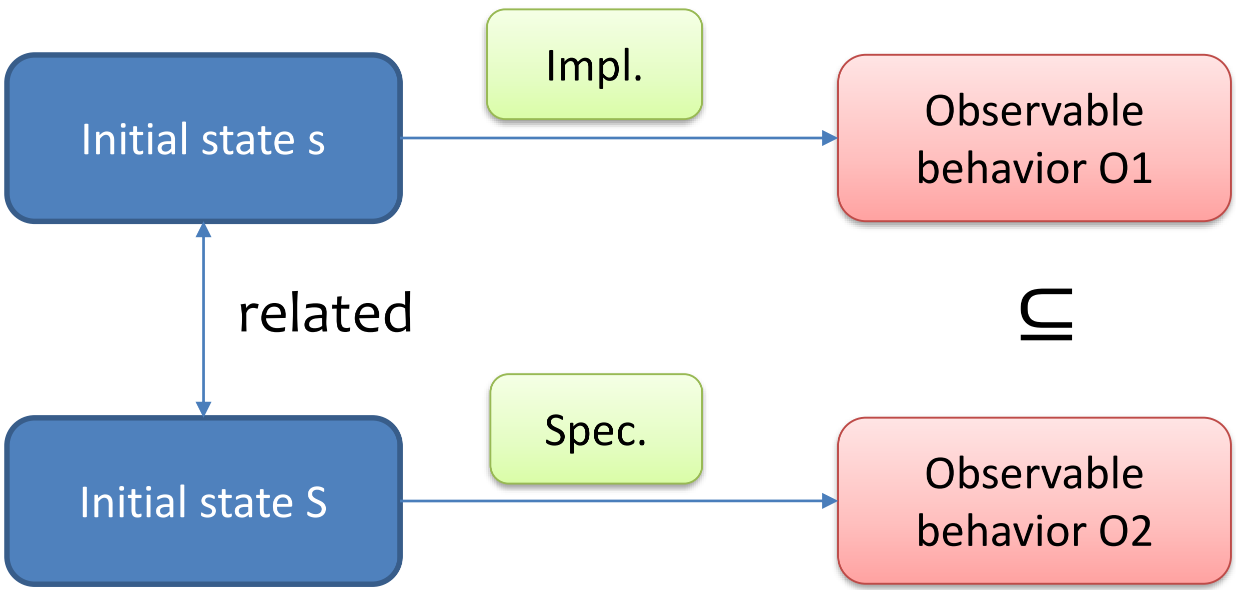
For two un-related program, the refinement is not useful

We usually request the initial state of two program map

E.g., in case (1), we require x = y

Refinement definition

Impl. refines spec. is defined as



Context

Impl. and spec. are full program (E.g., impl1 and impl2)
We only care about the inc implementation

Impl1

```
main()
    print(x);
    inc-lf();
    print(x);
```

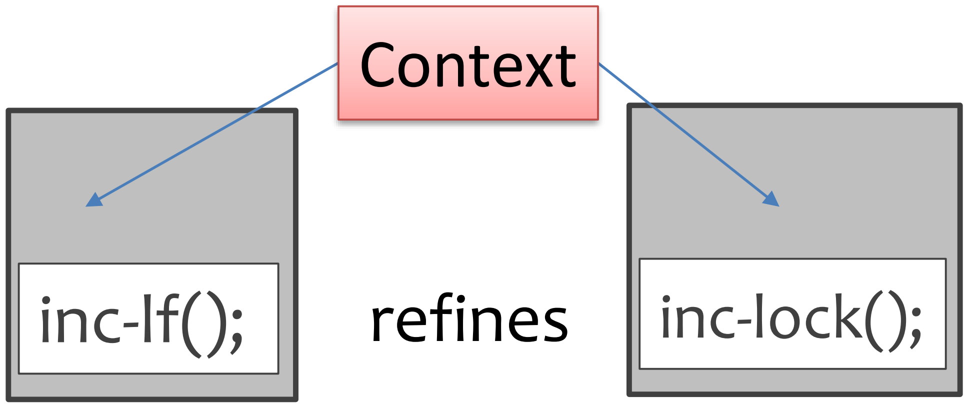
Impl2

```
thread1:
print(x);
inc-lf();
print(x);
```

```
thread2:
inc-lf();
```

Need a correctness condition that generalizes context

Contextual refinement



For any context, invoking impl. refines invoking spec, then impl. is the contextual refinement of spec.

What is unsaid?

- How to prove contextual refinement?
  - Consider all possible context?
  - Ans: simulation
- How to prove concurrent context?
  - Consider all possible interleavings?
  - Ans: CSL and RG
- How to prove other properties?
  - Many open problems!

Summary so far

- Introduce some widely used verification techniques
  - Hoare Logic and Separation Logic
  - Refinement
- Foundation for verifying practical systems

How do these theories apply to practical systems?
Are there new challenges?

PROOF AUTOMATION

Theorem proving is costly

- seL4: the first verified microkernel [SOSP'09]
  - A milestone in verification history
  - 8700 lines of C and 600 lines of assembler
- Total proof effort is 25 py (person year)
  - Redoing a similar proof requires 8 py

Do we have better ways to do proofs?

Classification of proof methods

	Push-button verification	Auto-active verification	Interactive verification
中文	一键证明	半自动证明	交互式证明
自动化程度	高	中	低
证明能力	低	中	高
证明	SMT Solver	SMT Solver	手动
包含范围	符号执行、模型检验	最弱前置条件、Dafny	Coq、Isabelle/HOL
问题	无法处理无界循环	无法处理并发	成本高

SMT solver

- Automatically decide whether some first-order logic formula is solvable
  - Output unsat or give an example
- Mostly used SMT solver: Z3

```
context cxt;
expr a = cxt.int_const("a");
expr b = cxt.int_const("b");
solver s(cxt);

//判断是否存在整数 a 和 b 使得 a>=0 和 b<a 同时成立
s.add(a >= 0);
s.add(b < a);

//输出判断结果 sat
cout << s.check();
```

Use SMT solver in verification

- Encode correctness condition as C
- Check if ~C holds
  - “unsat” means C always holds
  - Otherwise give a counter-example when C doesn’t hold

Push-button verification---symbolic execution

- Symbolic execution: treat input as symbolic

```
int x;
inc()
   x = x + 1;
```

Given $\quad \dfrac{P\ st1 \qquad st1 \xrightarrow{S} st2}{Q\ st2}$

Prove

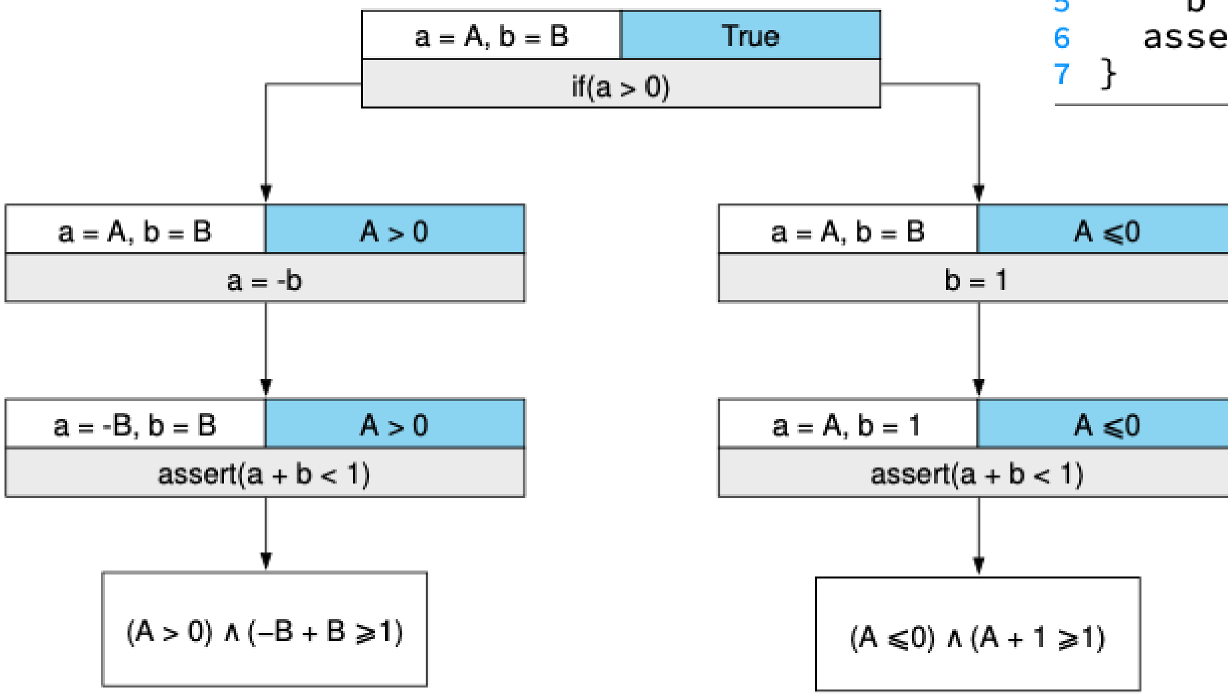
Interpret the program and compute st2

Leave Z3 to prove the goal

Push-button verification---symbolic execution

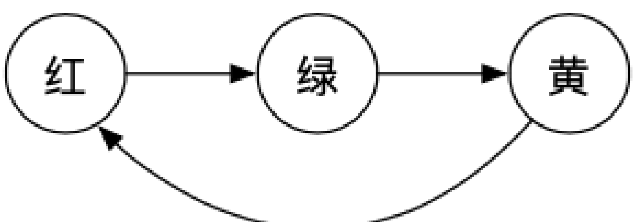
- A more complex case

```c
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 1;
  assert(a + b < 1);
}
```



Push-button verification---model checking

- Three parts to model checking
  - Model: a model of the system, represented as a state transition graph
  - Specification: usually written in temporal logic
  - Algorithm: exhaustive search



Spec: lights always go red

Limitation of push-button verification

- Exhaustive search cannot handle unbounded loop
  - SMT solver cannot return in a limited time
- To use push-button verification, rewrite program to remove unbounded loop

Auto-active verification---weakest precondition

- W is the weakest precondition of C and Q is defined as
  - forall P, if P => W, than {P} C {Q} holds
  - Represented as wp(C, Q)
- E.g., wp(x=x+1, x>2) is x>1
- To prove {P} C {Q}, we need to show P => wp(C, Q)
- E.g., to prove {x=2} x=x+1 {x>2}, we prove x=2 => x>1

Auto-active verification---Dafny

- Developed by Microsoft
- Users can add annotation to help the proof
  - Able to handle unbounded loop

```
method func(a : int, b : int) returns (c : int)
  requires a > 0
  ensures c < 1
{
  var tmp1 := a;
  var tmp2 := b;
  if a > 0
    { tmp1 := -b;}
  else
    { tmp2 := 1;}
  return tmp1 + tmp2;
}
```

Interactive verification---Coq

- Winner, 2013 ACM Software System Award
- “Coq is playing an essential role in our transition to a new era of formal assurance in mathematics, semantics and program verification”
- Have built-in domain specific automatic tactics
  - E.g., lia for Presburger arithmetic

```
Example silly_presburger_example : ∀ m n o p,
  m + n ≤ n + o ∧ o + 3 = p + 3 →
  m ≤ p.
Proof.
  intros. lia.
Qed.
```

Interactive verification---Coq

- Have a tactic language (Ltac) for developing automatic tactics
- We can invoke SMT solver in tactics

```
Theorem xor_zero_equal:
forall x,y,
xor x,y=zero→x=y.
Proof.
intros.apply same_bits_eq.
intros.assert(xorb(testbit x,i)(testbit y,i)=false).
rewrite←bits_xor; auto.
rewrite H.apply biths_zero.destruct(testbit x,i).destruct(testbit y,i).
reflexivity||discriminate.
Qed.
```

```
Theorem xor_zero_equal: forall x,y,
xor x,y=zero→x=y.
Proof.
smt4coq.
Qed.
```

VERIFICATION IN ACADEMIA AND INDUSTRY

Verification has gain popularity recently

- Formally verified microkernels
  - seL4(SOSP'09): first practical verified microkernel
  - CertiKOS(OSDI'16): first concurrent microkernel
- Formally verified compilers
  - CompCert (2008): a verified C compiler
  - Vellvm(2012): verified LLVM

Verification has gain popularity recently

- Formally verified file systems
  - FSCQ (SOSP’15): first verified crash-safe file system
  - AtomFS (SOSP’19): first verified concurrent file system
- Distributed systems, networks, cryptology, bitcoin …
- Next ten years will see a surge of verification efforts

Program analysis is widely deployed in industry

- Formal methods: mathematical method for developing and proving

形式化方法			
形式化证明			
	定理证明（半自动）	模型检验（自动）	程序分析（自动）
应用场景	核心系统软件（如OS、hypervisor）	软件/协议/算法验证	WCET/漏洞查找
包含范围	数学建模/程序逻辑/定理证明器	时序逻辑/模型检验工具	覆盖率分析/控制流分析/数据流分析

Automated verification techniques is more popular in industry

- Amazon: use model checking tools (TLA+) to help ensure correctness of products since 2011
  - "A big success at AWS"
- Microsoft research pioneers in automated verification
  - Verification tools: Z3, Dafny
  - Verification projects
    - Vale: verified low-level crypto (adopted by Linux)
    - Everest: verified HPPTS replacement
- Huawei makes huge investment in verification

Future of verification

- A uniform platform for doing verification
- More applications of verification to real-world systems
- Big challenges and opportunities

Conclusion

- What is verification?
- Verification techniques
- Now and future of verification

Thanks! Q&A