Incorrectness Logic

Peter W. O'Hearn

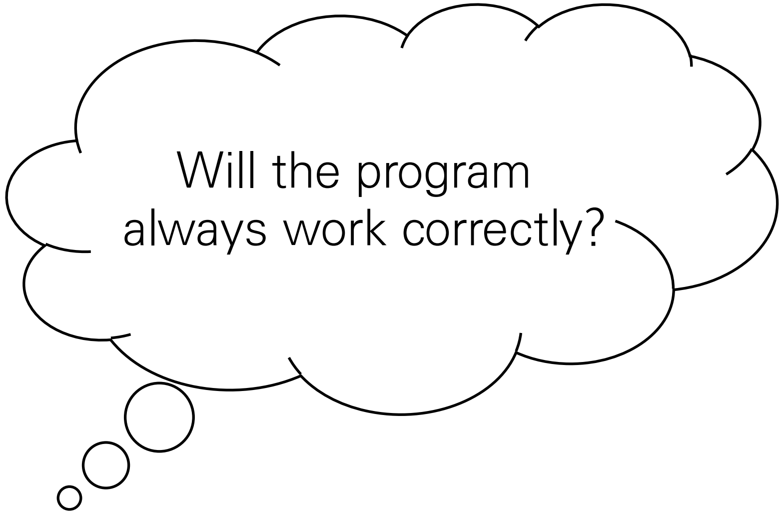
2024.05.02.

Motivation

- Even if you would like to have **correctness**,
  you might find yourself reasoning about **incorrectness**
- No **logical system** to reason about **the presence of bugs**

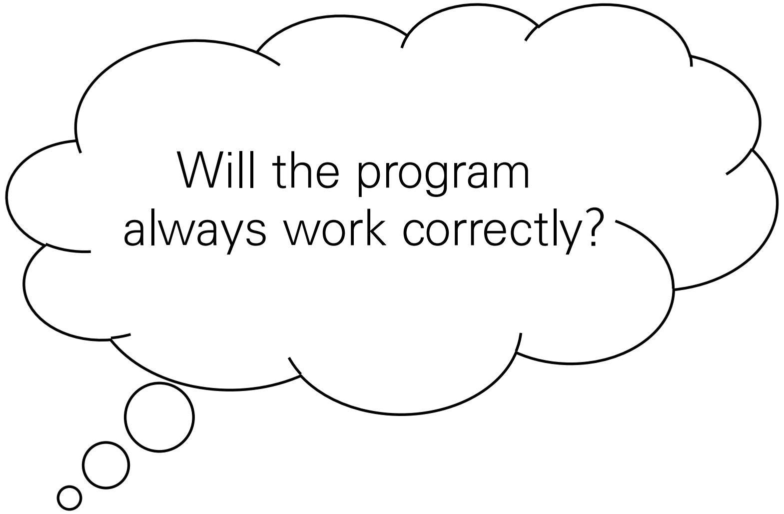
Motivation

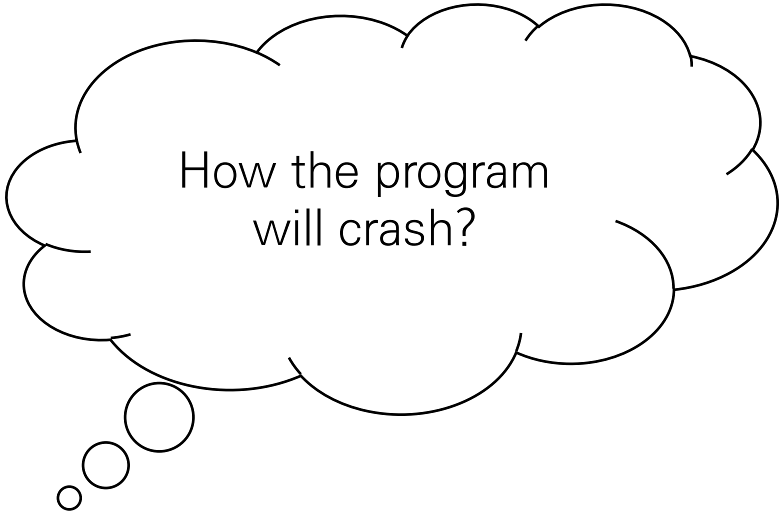
- Even if you would like to have **correctness**,
  you might find yourself reasoning about **incorrectness**
- No **logical system** to reason about **the presence of bugs**



Motivation

- Even if you would like to have **correctness**, you might find yourself reasoning about **incorrectness**
- No **logical system** to reason about **the presence of bugs**





Problems

1. Programmer's mind ↔ No logical system to reason about the presence of bugs

Problems

1. Programmer's mind ↔ No logical system to reason about the presence of bugs
2. Unable to reason abnormal termination of the program

Problems

1. Programmer's mind ↔ No logical system to reason about the presence of bugs
2. Unable to reason abnormal termination of the program
   - Hoare logic: reasons about the correctness of programs rigorously

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

Assertion prevents div0 → Formal description?

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

Assertion prevents div0 → Formal description?

$$\frac{\{\text{true}\}\ \text{assert}(X > 0)\ \{X > 0\} \qquad \{X > 0\}\ D \coloneqq 12/X\ \{X > 0 \land D > 0\}}{\{\text{true}\}\ \text{assert}(X > 0); D \coloneqq 12/X\ \{X > 0 \land D > 0\}}$$

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

Assertion prevents div0 → Formal description?

$$\frac{\{\text{true}\}\ \text{assert}(X > 0)\ \{X > 0\} \qquad \{X > 0\}\ D \coloneqq 12/X\ \{X > 0 \land D > 0\}}{\{\text{true}\}\ \text{assert}(X > 0); D \coloneqq 12/X\ \{X > 0 \land D > 0\}}$$

"Always satisfy this
(if normally terminated)"

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

```
X := input();
// assert(X > 0);
D := 12 / X;
```

No assertion

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

```
X := input();
// assert(X > 0);
D := 12 / X;
```

No assertion

$$\frac{[\text{true}]\ X \coloneqq \text{input}()\ [X = 0] \qquad [X = 0] D \coloneqq 12/X [\text{div0}: X = 0]}{[\text{true}] X \coloneqq \text{input}(); D \coloneqq 12/X [\text{div0}: X = 0]}$$

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

```
X := input();
// assert(X > 0);
D := 12 / X;
```

No assertion

$$\frac{[\mathrm{true}]\ X \coloneqq \mathrm{input}()\ [X = 0] \qquad [X = 0] D \coloneqq 12/X [\mathrm{div0}: X = 0]}{[\mathrm{true}] X \coloneqq \mathrm{input}(); D \coloneqq 12/X [\mathrm{div0}: X = 0]}$$

"Sometimes satisfy this"

Example

```
X := input();
assert(X > 0);
D := 12 / X;
```

```
X := input();
// assert(X > 0);
D := 12 / X;
```

No assertion

$$\frac{[\text{true}]\ X \coloneqq \text{input}()\ [X = 0] \qquad [X = 0] D \coloneqq 12/X[\text{div0}: X = 0]}{[\text{true}] X \coloneqq \text{input}(); D \coloneqq 12/X[\text{div0}: X = 0]}$$

"Sometimes satisfy this"

Orienting to prove the presence of bugs

Contribution

- Describe how under-approximate triple is relevant to proving the presence of bugs

Contribution

- Describe how under-approximate triple is relevant to proving the presence of bugs

- Designed specific logic system, **incorrectness logic**

Contribution

- Describe how under-approximate triple is relevant to proving the presence of bugs
- Designed specific logic system, **incorrectness logic**
- Explored reasoning idioms
  - Note: this paper doesn't delve into any specific analyses or tools

Key Idea

- Defining **incorrectness** logic = analogous to **correctness** logic
  - Hoare logic

Key Idea

- Defining **incorrectness** logic = analogous to **correctness** logic
  - Hoare logic
- **Under-approximate** the final state from starting state
  - vs. over-approximation

Key Idea

- Defining **incorrectness** logic = analogous to **correctness** logic
  - Hoare logic
- **Under-approximate** the final state from starting state
  - vs. over-approximation
- "From the presumption, the result can occur."

Example: under-approx.

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves : [z==2]
```

Example: under-approx.

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves : [z==2]
```

False under-approximate triple

Example: under-approx.

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves : [z==2]
```

False under-approximate triple

Also satisfied when z=2, x=1, y=2,
which cannot be achieved

Example: under-approx.

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves : [z==2]
```

False under-approximate triple

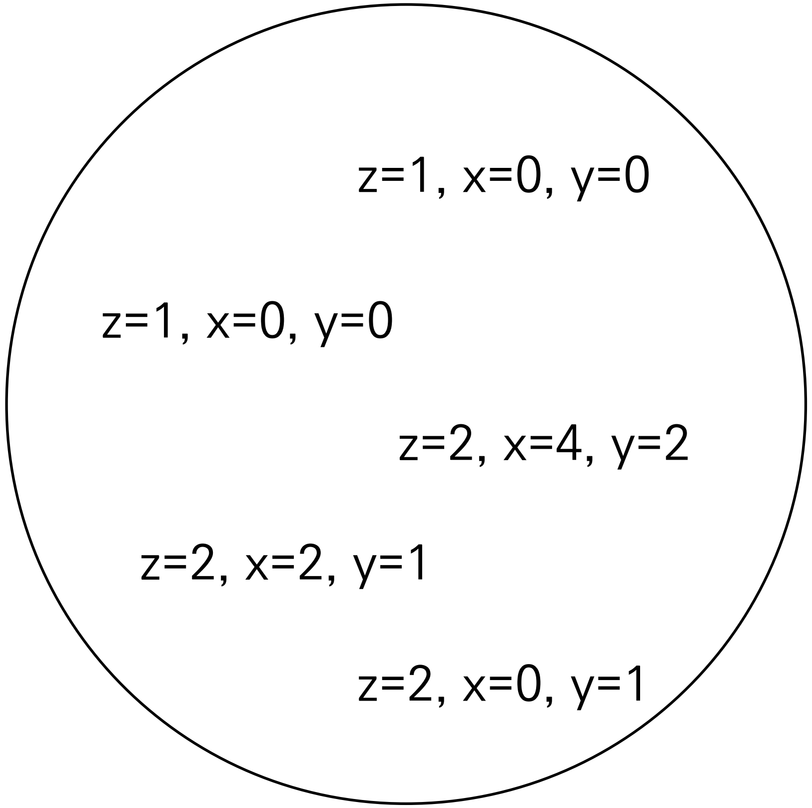
```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves :
// [z==2 and x==2 and y==1]
```

True under-approximate triple

Simple Diagram

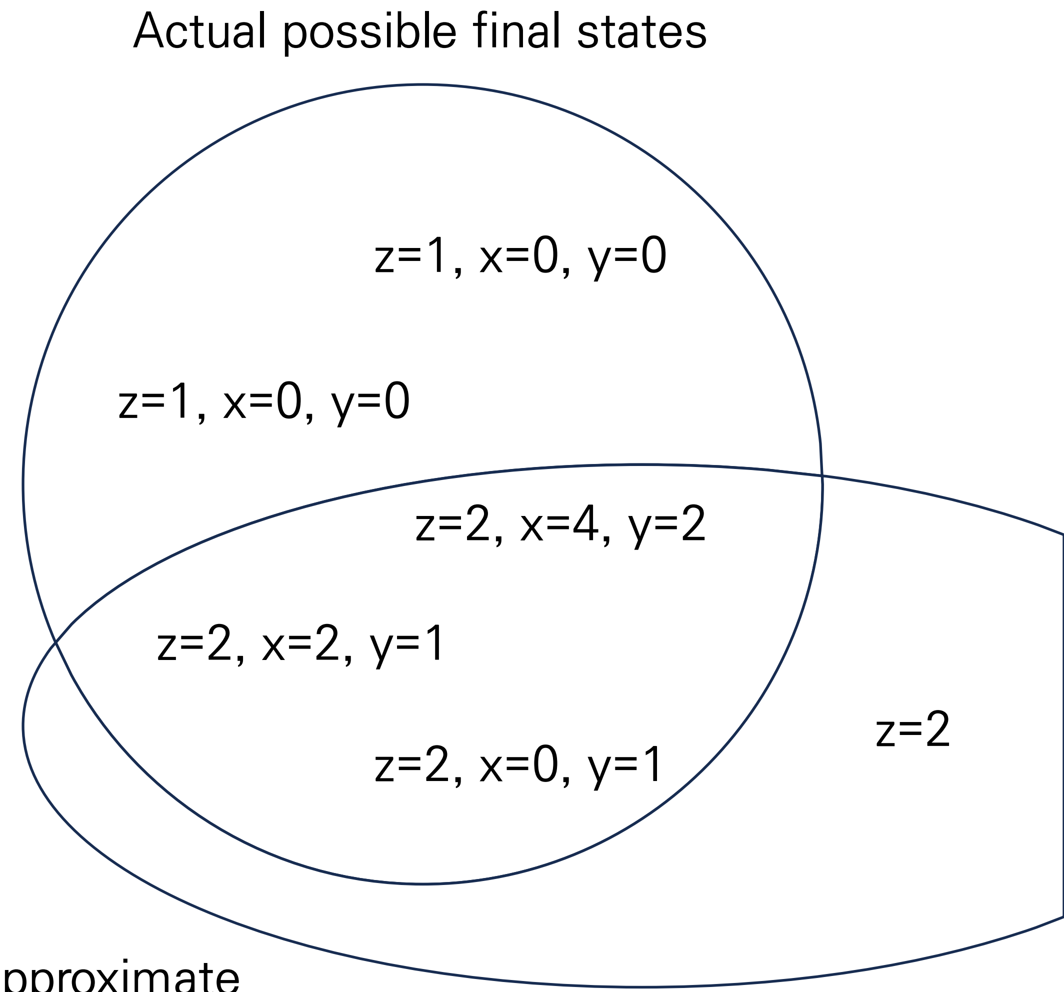
```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves :
// [z==2 and x==2 and y==1]
```

Actual possible final states



Simple Diagram

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves :
// [z==2 and x==2 and y==1]
```

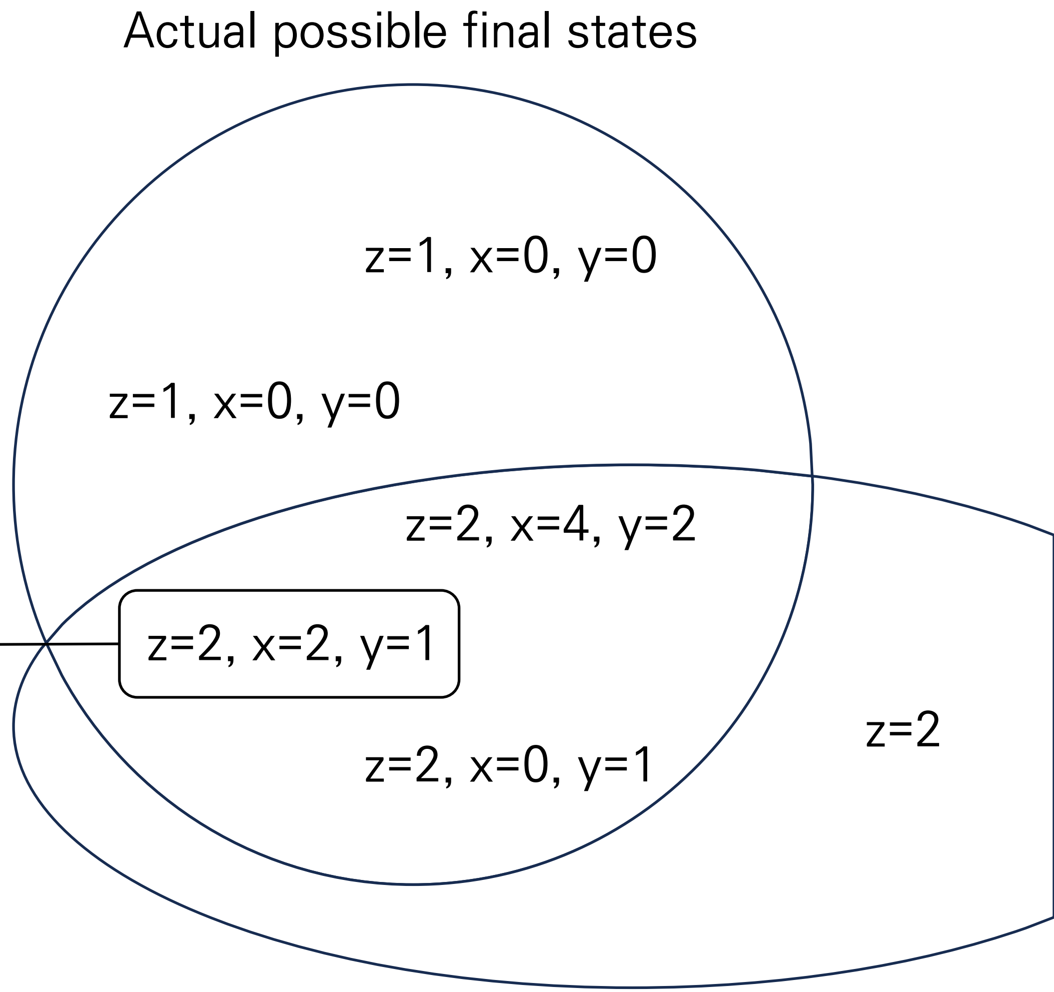


z=2 does not under-approximate

Simple Diagram

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
// achieves :
// [z==2 and x==2 and y==1]
```

“From the presumption, this result can occur.”



Vs. Hoare (Correctness) Logic

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
```

$$\frac{\{B \wedge P\}S\{Q\},\ \ \{\neg B \wedge P\}T\{Q\}}{\{P\}\ \text{if}\ B\ \text{then}\ S\ \text{else}\ T\ \{Q\}}$$

$$\frac{P_1 \rightarrow P_2,\ \{P_2\}S\{Q_2\},\ Q_2 \rightarrow Q_1}{\{P_1\}\ S\ \{Q_1\}}$$

Vs. Hoare (Correctness) Logic

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
```

$$\frac{\{B \wedge P\}S\{Q\},\ \ \{\neg B \wedge P\}T\{Q\}}{\{P\}\ \text{if}\ B\ \text{then}\ S\ \text{else}\ T\ \{Q\}}$$

$$\frac{P_1 \to P_2,\ \{P_2\}S\{Q_2\},\ Q_2 \to Q_1}{\{P_1\}\ S\ \{Q_1\}}$$

$$\frac{\cdots,\quad \overline{\{Z = 1 \wedge odd(X)\}\ \text{skip}\ \{Z = 1 \wedge odd(X)\}}}{\{Z = 1\}\ \text{if even}(X)\ \text{then (if odd}(Y)\ \text{then}\ Z \coloneqq 2\ \text{else skip) else skip}\ \{A\}}$$

Vs. Hoare (Correctness) Logic

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
```

$$\frac{\{B \land P\}S\{Q\},\ \ \{\neg B \land P\}T\{Q\}}{\{P\}\ \text{if}\ B\ \text{then}\ S\ \text{else}\ T\ \{Q\}}$$

$$\frac{P_1 \to P_2,\ \{P_2\}S\{Q_2\},\ Q_2 \to Q_1}{\{P_1\}\ S\ \{Q_1\}}$$

$$\frac{\cdots,\quad \overline{\{Z = 1 \land odd(X)\}\ \text{skip}\ \{Z = 1 \land odd(X)\}}}{\{Z = 1\}\ \text{if even}(X)\ \text{then (if odd}(Y)\ \text{then}\ Z \coloneqq 2\ \text{else skip) else skip}\ \{A\}}$$

where $A = \{Z = 1 \land \mathrm{odd}(X) \lor Z = 1 \land \mathrm{even}(X) \land \mathrm{even}(Y) \lor Z = 2 \land \mathrm{even}(X) \land \mathrm{odd}(Y)\}$

Vs. Hoare (Correctness) Logic

```
// presumes : [z==1]
if (x is even)
  if (y is odd)
    z = 2;
```

$$\frac{\{B \land P\}S\{Q\},\ \ \{\neg B \land P\}T\{Q\}}{\{P\}\ \text{if}\ B\ \text{then}\ S\ \text{else}\ T\ \{Q\}}$$

$$\frac{P_1 \to P_2,\ \{P_2\}S\{Q_2\},\ Q_2 \to Q_1}{\{P_1\}\ S\ \{Q_1\}}$$

$$\frac{\cdots,\quad \overline{\{Z = 1 \land odd(X)\}\ \text{skip}\ \{Z = 1 \land odd(X)\}}}{\{Z = 1\}\ \text{if even}(X)\ \text{then}\ (\text{if odd}(Y)\ \text{then}\ Z \coloneqq 2\ \text{else skip})\ \text{else skip}\ \{A\}}$$

where $A = \{Z = 1 \land \text{odd}(X) \lor Z = 1 \land \text{even}(X) \land \text{even}(Y) \lor Z = 2 \land \text{even}(X) \land \text{odd}(Y)\}$

"From the precondition, all final states satisfy postcondition."

Analogy Between Two Logics

- Correctness (Hoare) logic and Incorrectness logic

Analogy Between Two Logics

- Correctness (Hoare) logic and Incorrectness logic
  - Former: “From the precondition, **all final states satisfy** postcondition.”
  - Latter: “From the presumption, **this result can occur**.”

Analogy Between Two Logics

- Correctness (Hoare) logic and Incorrectness logic
  - Former: “From the precondition, all final states satisfy postcondition.”
  - Latter: “From the presumption, this result can occur.”

- Symmetric in many aspects

Analogy Between Two Logics

- Correctness (Hoare) logic and Incorrectness logic
  - Former: "From the precondition, all final states satisfy postcondition."
  - Latter: "From the presumption, this result can occur."

- Symmetric in many aspects

$$\frac{P_1 \rightarrow P_2,\ \ \{P_2\}\, S\, \{Q_2\},\ \ Q_1 \leftarrow Q_2}{\{P_1\}\, S\, \{Q_1\}} \qquad \frac{P_1 \leftarrow P_2,\ \ [P_2]\, S\, [Q_2],\ \ Q_1 \rightarrow Q_2}{[P_1]\, S\, [Q_1]}$$

Specifying Incorrectness

- Reasoning errors

Specifying Incorrectness

- Reasoning errors

```
X := input();
D := 12 / X;
```

Specifying Incorrectness

- Reasoning errors

```
X := input();
D := 12 / X;
```

- Correctness implicitly requires the successful termination of program

Specifying Incorrectness

- Reasoning errors

```
X := input();
D := 12 / X;
```

- Correctness implicitly requires the successful termination of program

- What about incorrectness logic?
  - “Div0” can be one end state of the program

Formal Description

- Incorrectness triple

$$[P]C[\epsilon{:}\,Q]$$

  - P: starting state (presumption)
  - C: code
  - ε: exit
    - For simplicity, we just discuss *ok* and *er* (which is manually raised by `error()`)
  - Q: ending state (result)

Defining Proof System

Empty under-approximates

$$[p]C[\epsilon{:}\,\mathit{false}]$$

Unit

$$[p]\texttt{skip}[ok{:}\,p][er{:}\,\mathit{false}]$$

Iterate zero

$$[p]C^\star[ok{:}\,p]$$

Choice (where $i = 1$ or 2)

$$\frac{[p]C_i[\epsilon{:}\,q]}{[p]C_1 + C_2[\epsilon{:}\,q]}$$

Consequence

$$\frac{p' \Leftarrow p \quad [p]C[\epsilon{:}\,q] \quad q \Leftarrow q'}{[p']C[\epsilon{:}\,q']}$$

Sequencing (short-circuit)

$$\frac{[p]C_1[er{:}\,r]}{[p]C_1; C_2[er{:}\,r]}$$

Iterate non-zero

$$\frac{[p]C^\star; C[\epsilon{:}\,q]}{[p]C^\star[\epsilon{:}\,q]}$$

Error

$$[p]\texttt{error()}[ok{:}\,\mathit{false}][er{:}\,p]$$

Disjunction

$$\frac{[p_1]C[\epsilon{:}\,q_1] \quad [p_2]C[\epsilon{:}\,q_2]}{[p_1 \vee p_2]C[\epsilon{:}\,q_1 \vee q_2]}$$

Sequencing (normal)

$$\frac{[p]C_1[ok{:}\,q] \quad [q]C_2[\epsilon{:}\,r]}{[p]C_1; C_2[\epsilon{:}\,r]}$$

Backwards Variant (where n fresh)

$$\frac{[p(n) \wedge \mathit{nat}(n)]C[ok{:}\,p(n+1) \wedge \mathit{nat}(n)]}{[p(0)]C^\star[ok{:}\,\exists n.p(n) \wedge \mathit{nat}(n)]}$$

Assume

$$[p]\texttt{assume}\ B[ok{:}\,p \wedge B][er{:}\,\mathit{false}]$$

Assignment

$$[p]x = e[ok{:}\,\exists x'.p[x'/x] \wedge x = e[x'/x]][er{:}\,\mathit{false}]$$

Constancy

$$\frac{[p]C[\epsilon{:}\,q]}{[p \wedge f]C[\epsilon{:}\,q \wedge f]} \quad \mathit{Mod}(C) \cap \mathit{Free}(f) = \emptyset$$

Substitution I

$$\frac{[p]C[\epsilon{:}\,q]}{([p]C[\epsilon{:}\,q])(e/x)} \quad \left(\mathit{Free}(e) \cup \{x\}\right) \cap \mathit{Free}(C) = \emptyset$$

Nondet Assignment

$$[p]x = \texttt{nondet()}[ok{:}\,\exists x' p][er{:}\,\mathit{false}]$$

Local Variable

$$\frac{[p]C(y/x)[\epsilon{:}\,q]}{[p]\texttt{local}\ x.C[\epsilon{:}\,\exists y.q]} \quad y \notin \mathit{Free}(p, C)$$

Substitution II

$$\frac{[p]C[\epsilon{:}\,q]}{([p]C[\epsilon{:}\,q])(y/x)} \quad y \notin \mathit{Free}(p, C, q)$$

$$\begin{aligned}
\texttt{while}\ B\ \texttt{do}\ C &=_{def} (\texttt{assume}(B); C)^\star; \texttt{assume}(\neg B) \\
\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C' &=_{def} (\texttt{assume}(B); C) + (\texttt{assume}(\neg B); C') \\
\texttt{assert}(B) &=_{def} \texttt{assume}(B) + (\texttt{assume}(\neg B); \texttt{error()})
\end{aligned}$$

Let's just see examples!

Examples – Proving Triple

```
// presumes : [true], achieves : [ ok : x>=0 ]
void f() {
  // skipped
}

// presumes : [true], achieves : [ er : x==10 ]
void client() {
  f();
  if (x==10) error();
}
```

Examples – Proving Triple

```
// presumes : [true], achieves : [ ok : x>=0 ]
void f() {
  // skipped
}

// presumes : [true], achieves : [ er : x==10 ]
void client() {
  f();
  if (x==10) error();
}
```

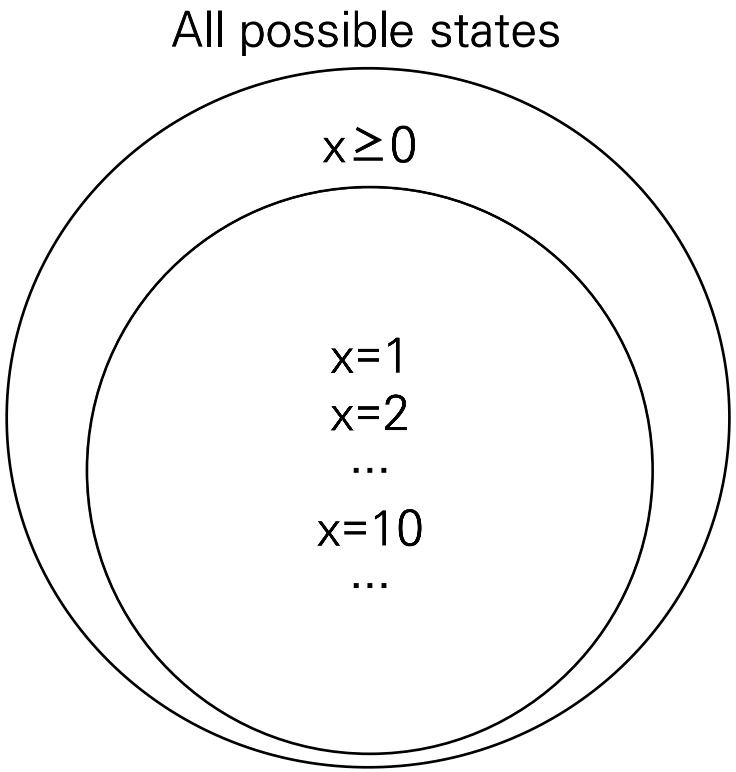
To-prove:

$$[\text{true}]\ \text{f}();\ \text{if}(x = 10)\ \text{then error}()\ \text{else skip}\ [\text{er}: x = 10]$$

Examples – Proving Triple

```
// presumes : [true], achieves : [ ok : x>=0 ]
void f() {
  // skipped
}

// presumes : [true], achieves : [ er : x==10 ]
void client() {
  f();
  if (x==10) error();
}
```



To-prove:

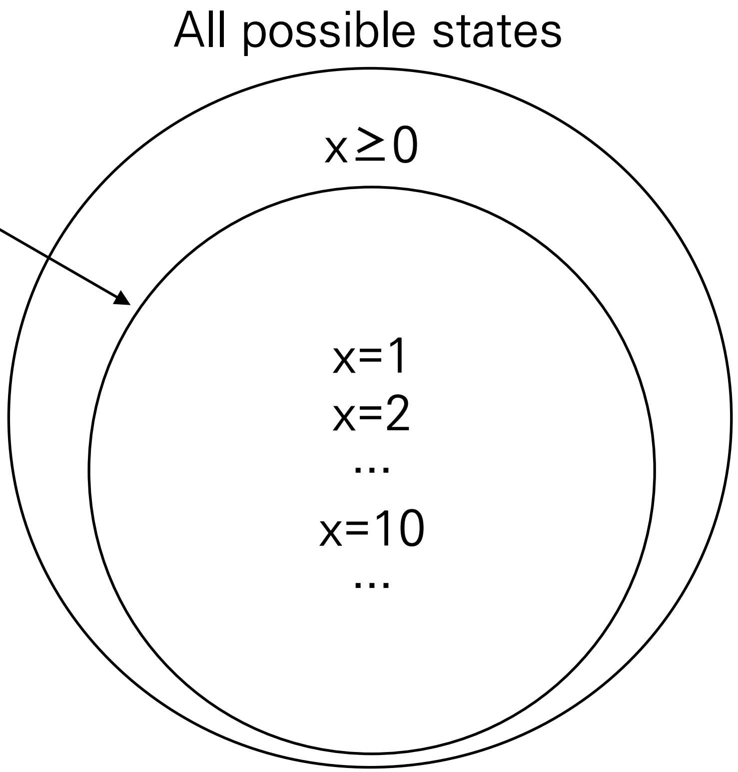
$[\text{true}]\ \text{f}();\ \text{if}(x = 10)\ \text{then error() else skip}\ [\text{er}: x = 10]$

Examples – Proving Triple

```
// presumes : [true], achieves : [ ok : x>=0 ]
void f() {
  // skipped
}

// presumes : [true], achieves : [ er : x==10 ]
void client() {
  f();
  if (x==10) error();
}
```

Not all output satisfy x≥0,
but all satisfy x≥0 is possible output



To-prove:

$[\text{true}]\ \text{f}();\ \text{if}(x = 10)\ \text{then error}()\ \text{else skip}\ [\text{er}: x = 10]$

Examples – Proving Triple

- Goal: to prove $[\text{true}]\ \text{f}();\ \text{if}(x = 10)\ \text{then error}()\ \text{else skip}\ [\text{er}: x = 10]$
  - Given that $[\text{true}]\ \text{f}()\ [\text{ok}: x \geq 0]$

Examples – Proving Triple

- Goal: to prove $[\mathrm{true}]\ \mathrm{f}();\ \mathrm{if}(x = 10)\ \mathrm{then\ error}()\ \mathrm{else\ skip}\ [\mathrm{er}{:}\, x = 10]$
  - Given that $[\mathrm{true}]\ \mathrm{f}()\ [\mathrm{ok}{:}\, x \geq 0]$

$$\frac{[\mathrm{true}]\ \mathrm{f}()[\mathrm{ok}{:}\, x \geq 0], \quad x \geq 0 \Leftarrow x = 10}{[\mathrm{true}]\ \mathrm{f}()\ [\mathrm{ok}{:}\, x = 10]}$$

Consequence rule

$$\frac{P_1 \leftarrow P_2,\ [P_2]\ S\ [\epsilon{:}\, Q_2],\ Q_2 \leftarrow Q_1}{[P_1]\ S\ [\epsilon{:}\, Q_1]}$$

Examples – Proving Triple

- Goal: to prove $[\text{true}]\ \text{f}();\ \text{if}(x = 10)\ \text{then error() else skip}\ [\text{er}: x = 10]$
  - Given that $[\text{true}]\ \text{f}()\ [\text{ok}: x \geq 0]$

$$\frac{[\text{true}]\ \text{f}()[\text{ok}: x \geq 0],\quad x \geq 0 \Leftarrow x = 10}{[\text{true}]\ \text{f}()\ [\text{ok}: x = 10]}$$

Consequence rule

$$\frac{P_1 \leftarrow P_2,\ \ [P_2]\ S\ [\epsilon: Q_2],\ \ Q_2 \leftarrow Q_1}{[P_1]\ S\ [\epsilon: Q_1]}$$

$$\frac{[x = 10]\ \text{error()}\ [\text{er}: x = 10]}{[x = 10]\ \text{if}\ x = 10\ \text{then error() else skip}\ [\text{er}: \text{x} = 10]}$$

Error rule (+ condition rule, skip rule)

$$\frac{}{[P]\ \text{error()}\ [\text{er}: P]}$$

Examples – Proving Triple

- Goal: to prove $[\text{true}]\ \text{f}();\ \text{if}(x = 10)\ \text{then error() else skip}\ [\text{er}: x = 10]$
  - Given that $[\text{true}]\ \text{f}()\ [\text{ok}: x \geq 0]$

$$\frac{[\text{true}]\ \text{f}()\ [\text{ok}: x = 10], \qquad [x = 10]\ \text{if}\ x = 10\ \text{then error() else skip}\ [\text{er}: \text{x} = 10]}{[\text{true}]\ \text{f}();\ \text{if}(x = 10)\ \text{then error() else skip}\ [\text{er}: \text{x} = 10]}$$

Normal sequencing rule

$$\frac{[P]\ C_1\ [\text{ok}: Q],\ \ [Q] C_2 [\epsilon: R]}{[P]\ C_1; C_2\ [\epsilon: R]}$$

Review

- Pros
  - Provided contexts of motivations and intuitions enough
  - A number of examples
  - Easy-to-understand notations
  - Precise definition

- Cons
  - Almost math paper, hard to understand
  - No actual applications, rely on readers
  - Bad readability

Review

- Questions
  - Trade-offs: How does the efficiency change compared to traditional verification techniques?
    - Inefficiency of rule-based solvers
  - Reasoning about specific properties: How to reason about other type of errors like memory leaks or security vulnerabilities?
  - Real-world applications: Have there been successful applications of IL in finding bugs in real-world software systems?

Summary

- Incorrectness logic: reasoning about presence of bugs
  - Under-approximation
  - Vs. Correctness (Hoare) logic
- Design
  - Specifying incorrectness
  - Defining proof system
- Example proof of presence of bugs

Aux: 3rd problem

3.3 Under-approximate Success

Even if we were mainly interested in incorrectness, under-approximate result assertions describing successful computations can help us soundly discover bugs that come after a procedure is called. In particular, if we were to have over-approximate assertions only for successful computations, then our reasoning could go wrong, as the following example illustrates.

```
void mkeven()
/*  presumes: [true],  wrong achieves: [ok: x==2 || x==4]     */
  { x=2; }

void usemkeven()
  {  mkeven();  if (x==4) {error();} }
```

We use `ok:` before an assertion to indicate that it describes a result for normal, not exceptional, termination of a program. The achieves assertion `mkeven()` describes an over-approximation of what the procedure produces, including a possibility (`x==4`) than cannot occur. If we were to use this wrong achieves assertion in `usemkeven()` to conclude that an error is possible then this would be a false positive warning.

For this reason, our formalism will include under-approximate achieves-assertions for both successful and erroneous termination. `mkeven()` achieves "`ok: x==2`", not "`ok: x==2 || x==4`".

Aux: section 2 and section 5

- Section 2: analogy between Hoare and incorrectness logic
- Section 5: semantic foundation of the rules defined