Program analysis: from proving correctness to proving incorrectness

Roberto Bruni, Roberta Gori
(University of Pisa)
Lecture #03

BISS 2024
March 11-15, 2024
Program incorrectness: pragmatic motivations
Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their code, as well as that supported by automated testing and static analysis tools. This paper describes a simple logic for program incorrectness which is, in a sense, the other side of the coin to Hoare’s logic of correctness.

1 INTRODUCTION
When reasoning informally about a program, people make abstract inferences about what might go wrong, as well as about what must go right. A programmer might ask “will the program crash if we give it a large string?”, without saying which large string. In this paper we investigate the hypothesis that reasoning about the presence of bugs can be underpinned by sound techniques in a principled logical system, just as reasoning about correctness (absence of bugs) has been demonstrated to have sound logical principles in an extensive research literature. We also consider the relationship of the principles to automated reasoning tools for finding bugs in software.

We explore our hypothesis by defining incorrectness logic, a formalism that is similar to Hoare’s logic of program correctness [Hoare 1969], except that it is oriented to proving incorrectness rather than correctness. Hoare’s theory is based on specifications of the form

\[ \{\text{pre-condition}\} \text{code} \{\text{post-condition}\} \]

which say that the post-condition over-approximates (describes a superset of) the states reachable upon termination when the code is executed starting from states satisfying the pre-condition (the so-called strongest post). Conversely, we use a specification form

\[ \{\text{presumption}\} \text{code} \{\text{result}\} \]

which says that the post-assertion result be an under-approximation (subset) of the final states that can be reached starting from states satisfying the presumption.

The under-approximate triples were studied (with a different but equivalent definition) previously by de Vries and Koutavas [2011] in their reverse Hoare logic, which they used to specify randomized algorithms. Incorrectness logic adds post-assertions for errors as well as for normal termination, and these assertions describe erroneous states that can be reached by actual program executions. Dijkstra [1976] famously remarked that “testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absence,” and he made this remark while arguing for the
Picturing incorrectness

Hoare partial correctness triple

\( \{ - \} c \{ - \} \)

Postcondition

\( \text{post}(c) \)

O'Hearn incorrectness triple

\([-]c[-]\)

\(\cup\)

Predicates

Unapproximation

\(\text{under approximation}\)

\(\text{strongest postcondition}\)

\(\text{weakest under approximated post}\)
Correctness vs incorrectness

Over-approximation:
good for proving correctness

\[[c]P\]

true negatives

false negatives

false positive

true positives

bad for bug-finding

Under-approximation:
bad for proving correctness

\[[c]P\]

true negatives

false negatives

false positive

true positives

good for bug-finding

Over-approximation:
good for proving correctness

\[[c]P\]

false negatives

true positives

false positives

true negatives

bad for bug-finding

Under-approximation:
bad for proving correctness

\[[c]P\]

false negatives

true positives

false positives

true negatives

good for bug-finding
Correctness workflow, ideally

- developer
- code repository
- deployment
- code inspector

>1M source control commands run per day
>100K commits per week

Correct!

some scalability issues in a production environment:
- analysis takes time (overnight?), warnings are received late,
- false positives mine credibility

MOVE FAST AND BREAK THINGS
THE FOOLISH WAIT
DONE IS BETTER THAN PERFECT
Design principles

Low friction
do not rely on manual annotations

Act fast
able to report errors in less than 15’

Be compositional
whole program analysis is discouraged

Occam
do not use complex techniques (unless forced)

True positive theorem!
(under certain assumptions) the analyzer reports no false positives

“do not spam the developers!”
Incorrectness Logic (IL)
Hoare’s triples

{P} c {Q}

for any input matching the precondition
executing the command establishes the postcondition

[[c]]P ⊆ Q

over approximation!
can include non reachable states

O’Hearn’s triples

[P] c [Q]

any output matching the postcondition
can be reached by executing the command
on some input matching the precondition

[[c]]P ⊇ Q

under approximation!
includes just reachable states
As first order formulas

\[\{P\} \ c \ \{Q\}\]

\[[c]\]P \subseteq Q \equiv \forall \sigma \in P. \forall \sigma' \in [c]\sigma. \sigma' \in Q\]

any reachable output satisfies the postcondition

\[[P] \ c \ [Q]\]

\[[c]\]P \supseteq Q \equiv \forall \sigma' \in Q. \exists \sigma \in P. \sigma' \in [c]\sigma\]

any output in the postcondition is reachable
Regular commands

\[ r ::= \]
\[ e \]
\[ r_1 ; r_2 \]
\[ r_1 + r_2 \]
\[ r^* \]

atomic command

choice

Kleene star

\[ e ::= \]
\[ \text{skip} \]
\[ x ::= a \]
\[ b? \]
\[ \text{error( )} \]
\[ x ::= \text{nondet( )} \]
\[ ... \]
Exit condition

\[
[P] \quad r \quad [\epsilon : Q]
\]

\(\epsilon\) is the exit condition

ok: normal execution

er: erroneous execution

\([y = v] \ x := y \quad [ok : x = y = v] \quad [y = v] \ error( ) \ [er : y = v]\)
Notation

\[[P] \; r \; [\text{ok} : Q_1] [\text{er} : Q_2]\]

stands for

\[[P] \; r \; [\text{ok} : Q_1] \text{ and } [P] \; r \; [\text{er} : Q_2]\]
Floyd’s axiom for assignment

\[ [P] \ x := a \ [\text{ok} : \exists x'. P[x'/x] \land x = a[x'/x]] \ [\text{er} : \text{false}] \]

\[ [y = 42] \ x := 42 \ [\text{ok} : x = y = 42] \]
Hoare’s axiom for assignment?

\[ [Q[a/x]] \ x := a \ [ok : Q][er : false] \]

\[ [y = 42] \ x := 42 \ [ok : x = y] \]

unsound!

\( \sigma \triangleq [x \mapsto 3, y \mapsto 3] \) not reachable
Other atomic commands

\[ [P] \text{skip} \ [\text{ok} : P][\text{er} : \text{false}] \]

\[ [P] \ b? \ [\text{ok} : P \land b][\text{er} : \text{false}] \]

\[ [P] \text{error}( ) \ [\text{ok} : \text{false}][\text{er} : P] \]

\[ [P] \ x := \text{nondet}( ) \ [\text{ok} : \exists x . P][\text{er} : \text{false}] \]
Short circuiting of errors

\[
\begin{align*}
[P] r_1 [\text{ok} : R] & \quad [R] r_2 [\epsilon : Q] \\
\hline
[P] r_1; r_2 [\epsilon : Q] & \quad [P] r_1 [\text{er} : Q] \\
\hline
[P] r_1; r_2 [\text{er} : Q]
\end{align*}
\]

\[ [y = v] \text{error()} ; x := y [\text{er} : y = v] \]
Dropping disjuncts

\[
\begin{align*}
[P] r_1 \ [e : Q] \\
\frac{[P] r_1 + r_2 \ [e : Q]}{[P] r_2 \ [e : Q]} \\
\frac{[P] r_1 + r_2 \ [e : Q]}{[P] r_1 + r_2 \ [e : Q]}
\end{align*}
\]

sound under-approximation!
scalable bug detection

\[
\begin{align*}
[y = v] \ \text{error( ) + x := y} \ [\text{er : } y = v] \\
[y = v] \ \text{error( ) + x := y} \ [\text{ok : } x = y = v]
\end{align*}
\]
Example

\[ y = 0 \] if \( \text{even}(x) \) then \( y := 42 \) \[ \text{ok : } y = 42 \] \( \times \)

is it a valid IL triple?

\((y = 42) \triangleq \{ \text{[} x \mapsto 0, y \mapsto 42 \text{]}, \text{[} x \mapsto 1, y \mapsto 42 \text{]}, \text{[} x \mapsto 2, y \mapsto 42 \text{]}, \ldots \} \)

\( \checkmark \) \( \times \)
Example

\[ y = 0 \] if \( even(x) \) then \( y := 42 \) \[ ok : y = 42 \land even(x) \] ✔

is it a valid IL triple?

\( y = 42 \land even(x) \triangleq \{ [x \mapsto 0, y \mapsto 42], [x \mapsto 2, y \mapsto 42], \ldots \} \)

✔ ✔
IL vs HL

\[ y = 0 \] if even(x) then \( y := 42 \) [ok : \( y = 42 \land \text{even}(x) \)]

\{ y = 0 \} if even(x) then \( y := 42 \) \{ y = 42 \land \text{even}(x) \}  \times

\{ y = 0 \land \text{even}(x) \} if even(x) then \( y := 42 \) \{ y = 42 \}  \checkmark

\[ y = 0 \land \text{even}(x) \] if even(x) then \( y := 42 \) [ok : \( y = 42 \)]  \times
Bounded loop unrolling

\[ [P] \, r^\ast \, [\text{ok} : P] \]

\[ [P] \, r^\ast \, [\varepsilon : Q] \]

sound under-approximation!
scalable bug detection

\[ [x = 0] \, (x := x + 1)^\ast \, [\text{ok} : x = 0] \]

\[ [x = 0] \, (x := x + 1)^\ast \, [\text{ok} : x = 2] \]
Backwards variant (weak)

∀n ∈ ℕ. [P_n] r [ok : P_{n+1}]

\[ P_0 \] r* [ok : P_k]

loop invariants are inherently over-approximations sub-variants to reason about loop under-approximation

\[
[x = 0] (x := x + 1)^* [ok : x = 2^{42}] // P_n \triangleq (x = n)
\]

\[
[x = 0] (x := x + 1)^* ; \text{if } (x = 2^{42}) \text{ then error() } [er : x = 2^{42}]
\]
Consequence rule

\[ P' \Rightarrow P \quad [P'] \ r \ [\epsilon : Q'] \quad Q \Rightarrow Q' \]

\[ \frac{[P] \ r \ [\epsilon : Q]}{[P] \ r \ [\epsilon : Q]} \]

shrink the post!

scalable bug detection

\[ P \Rightarrow P' \quad \{P'\} \ r \ \{Q'\} \quad Q' \Rightarrow Q \]

\[ \frac{\{P\} \ r \ \{Q\}}{} \]
Some dualities

\[[P] \; r \; [Q_1] \; \land \; [P] \; r \; [Q_2] \iff [P] \; r \; [Q_1 \lor Q_2]\]

\\{P\} \; r \; \{Q_1\} \; \land \; \{P\} \; r \; \{Q_2\} \iff \{P\} \; r \; \{Q_1 \land Q_2\}\]
Some dualities

dropping disjuncts (by conseq. rule)

\[
[P] \rightarrow [Q \lor R]
\]

\[
[P] \rightarrow [Q]
\]

dropping conjuncts (by conseq. rule)

\[
\{P\} \rightarrow \{Q \land R\}
\]

\[
\{P\} \rightarrow \{Q\}
\]
A duality

For correctness reasoning
You get to forget information as you go along a path, but you must remember all the paths.

For incorrectness reasoning
You must remember information as you go along a path, but you get to forget some of the paths.

(Slide courtesy of Peter O'Hearn)
Principle of agreement

Th.

If \([P'] r [Q']\) \land

\(P' \Rightarrow P \land\)

\(\{P\} r \{Q\}\)

then \(Q' \Rightarrow Q\)

Proof.

\(Q' \subseteq\) \hspace{1cm} // by IL

\([[r]P'] \subseteq\) \hspace{1cm} // \(P' \Rightarrow P\)

\([[r]P] \subseteq\) \hspace{1cm} // by HL

\(Q\)

partially correct programs cannot exhibit counterexamples
Principle of denial

Th.
If \([P'] \mathrel{r} [Q']\) \land
\[P' \Rightarrow P \land \{P\} \mathrel{r} \{Q\}\]
then \(Q' \Rightarrow Q\)

Cor.
If \([P'] \mathrel{r} [Q']\) \land
\[P' \Rightarrow P \land \neg (Q' \Rightarrow Q)\]
then \(\neg (\{P\} \mathrel{r} \{Q\})\)

any derivable counterexample witnesses program incorrectness
Examples

[true]
if \( x \geq 0 \) then
  \([x \geq 0]\)
skip
  \([x \geq 0]\)
else
  \([x < 0]\)
  \(x := -x\)
  \([\exists x'. x' < 0 \land x = -x'] \equiv [x > 0]\)
[ok : x \geq 0]
Examples

\[ z = 11 \]
if \( \text{even}(x) \) then
\[ z = 11 \land \text{even}(x) \]
if \( \text{odd}(y) \) then
\[ z = 11 \land \text{even}(x) \land \text{odd}(y) \]
\( z := 42 \)
\[ z = 42 \land \text{even}(x) \land \text{odd}(y) \]
\[ \text{ok} : z = 42 \land \text{even}(x) \land \text{odd}(y) \]
Finite unrolling of while loops

while \( b \) do \( c \triangleq (b?; c)^*; \neg b? \)

\[
[P] \text{ while } b \text{ do } c \quad [\text{ok} : P \land \neg b]
\]

\[
[P \land b] \quad c \quad [\text{ok} : Q]
\]

\[
[P] \text{ while } b \text{ do } c \quad [\text{ok} : (P \lor Q) \land \neg b]
\]
Finite unrolling of while loops

while $b$ do $c \triangleq (b?; c)^*; \neg b$?

\[ [P] \quad (b?; c)^* \quad [\text{ok} : P] \quad [P] \quad \neg b? \quad [\text{ok} : P \land \neg b] \]

\[ [P] \quad \text{while } b \text{ do } c \quad [\text{ok} : P \land \neg b] \]
Finite unrolling of while loops

while $b$ do $c \triangleq (b?; c)^*; \neg b$

$r \triangleq b?; c$

\[
\begin{align*}
&P b? [\text{ok : } P \land b] & [P \land b] c [\text{ok : } Q] \\
&P r^* [\text{ok : } P] & [P] r [\text{ok : } Q] \\
&P r^*; r [\text{ok : } Q] \\
&P r^* [\text{ok : } Q] & [Q] \neg b? [\text{ok : } Q \land \neg b] \\
&P \text{while } b \text{ do } c [\text{ok : } (P \lor Q) \land \neg b]
\end{align*}
\]
Examples

[true]

\( n := \text{nondet}(); \)

[true]

\( x := 0; \)

[x = 0]

While \( n > 0 \) do ( \( [x = 0 \land n > 0] \)

\( x := x + n; \)

[x = n \land n > 0]

\( n := \text{nondet}(); \)

\( [\exists n. x = n \land n > 0] \equiv [x > 0] \)

\( [\text{ok} : x \geq 0 \land n \leq 0] \)

\[ [P \land b] \; c \; [\text{ok} : Q] \]

\[ [P] \; \text{while} \; b \; \text{do} \; c \; [\text{ok} : (P \lor Q) \land \neg b] \]
Validity, soundness, completeness
Validity

A IL triple $[P] r [Q]$ is valid if $Q \subseteq [r]P$

Is $[x > 0] x := 10x [x > 10]$ valid?  ❌

Is $[x > 0, y > 0] x := yx [x \geq 0]$ valid?  ❌

Is $[x > 0, y > 0] x := yx [x = 42, y = 7]$ valid?  ✔

Is $[xy > 0] (x := yx)^* [x > 0, y \neq 0]$ valid?  ✔
Relational semantics

\[[r] : \wp(\Sigma) \rightarrow \wp(\Sigma)\]

\[[r]\epsilon \subseteq \Sigma \times \Sigma\]

\[[r]\text{ok} \subseteq \Sigma \times \Sigma\]

\[[r]\text{er} \subseteq \Sigma \times \Sigma\]
Semantics: atomic commands

\[
\begin{align*}
[[\text{skip}]] \text{ok} & \triangleq \{ (\sigma, \sigma) \mid \sigma \in \Sigma \} \\
[[\text{skip}]] \text{er} & \triangleq \emptyset \\
[[b?]] \text{ok} & \triangleq \{ (\sigma, \sigma) \mid \sigma \models b \} \\
[[b?]] \text{er} & \triangleq \emptyset \\
[[x := a]] \text{ok} & \triangleq \{ (\sigma, \sigma[x \mapsto [[a]]\sigma]) \mid \sigma \in \Sigma \} \\
[[x := a]] \text{er} & \triangleq \emptyset
\end{align*}
\]
Semantics: atomic commands

\[
\begin{align*}
[[\text{error()}]]\text{ok} & \triangleq \emptyset \\
[[\text{error()}]]\text{er} & \triangleq \{ (\sigma, \sigma) \mid \sigma \in \Sigma \}
\end{align*}
\]

\[
\begin{align*}
[[x := \text{nondet()}]]\text{ok} & \triangleq \{ (\sigma, \sigma[x \mapsto v]) \mid \sigma \in \Sigma, v \in \mathbb{Z} \} \\
[[x := \text{nondet()}]]\text{er} & \triangleq \emptyset
\end{align*}
\]
Semantics: compositions

\[ S, T \subseteq \Sigma \times \Sigma \]
\[ T \circ S \triangleq \{ (\sigma_1, \sigma_2) \mid \exists \sigma. (\sigma_1, \sigma) \in S \land (\sigma, \sigma_2) \in T \} \subseteq \Sigma \times \Sigma \]

\[[r_1; r_2]_{\text{ok}} \triangleq [r_2]_{\text{ok}} \circ [r_1]_{\text{ok}}\]
\[[r_1; r_2]_{\text{er}} \triangleq [r_1]_{\text{er}} \cup ([r_2]_{\text{er}} \circ [r_1]_{\text{ok}})\]

\[[r_1 + r_2]_{\epsilon} \triangleq [r_1]_{\epsilon} \cup [r_2]_{\epsilon}\]

\[[r^*]_{\epsilon} \triangleq \bigcup_{k \in \mathbb{N}} [r^k]_{\epsilon}\]

where \( r^k \triangleq r; \cdots; r \)

\( k \) times
Minimal set of rules

\[
\begin{align*}
[P] & \quad e \quad [[e]]P \quad \text{[atom]} \\
[P] & \quad r_1 \quad [R] \quad [R] \quad r_2 \quad [Q] \quad \text{[seq]} \\
& \quad [P] \quad r_1; \quad r_2 \quad [Q] \\
\forall i \in \{1, 2\} \quad [P] & \quad r_i \quad [Q_i] \quad \text{[choice]} \\
& \quad [P] \quad r_1 + \quad r_2 \quad [Q_1 \cup \quad Q_2] \\
\forall n \geq 0. \quad [P_n] & \quad r \quad [P_{n+1}] \quad \text{[iter]} \\
& \quad [P_0] \quad r^* \quad [\exists k. \quad P_k] \\
\end{align*}
\]

\[
P' \implies P \quad [P'] \quad r \quad [Q] \quad Q \implies Q' \quad \text{[cons]}
\]
Auxiliary rules

\[
\frac{[P_1] \ r \ [Q_1] \quad [P_2] \ r \ [Q_2]}{[P_1 \lor P_2] \ r \ [Q_1 \lor Q_2]} \quad \text{[disj]}
\]

\[
\frac{[P] \ r^* \ [P]}{[P] \ r^* \ [P]} \quad \text{[iter0]}
\]

\[
\frac{[P] \ r^* ; \ r \ [Q]}{[P] \ r^* \ [Q]} \quad \text{[unroll]}
\]

\[
P' \Rightarrow P \quad [P] \ r \ [Q] \quad \text{[weak]}
\]

\[
\frac{[P] \ r \ [Q'] \quad Q \Rightarrow Q'}{[P] \ r \ [Q]} \quad \text{[stren]}
\]

assigned variables in \( r \) are disjoint from free variables in \( R \)
Correctness

**Th.** Any derivable IL triple is valid

**Proof.** By induction on the derivation tree
(Relative) Completeness

**Th.** Any valid IL triple can be derived.

**Proof.** (Assuming an oracle to decide implications.)
Roughly, by structural induction on the command \( r \).

Atomic commands: \([\text{atom}] + [\text{cons}]\)
Choice and sequence: by inductive hyp. + \([\text{disj}] + [\text{cons}]\)
Kleene star: see O’Hearn’s paper
Questions
Question 1
Which IL triples are valid for any $r$ and $P$?

- $[P] r [\text{ok} : \text{false}][\text{er} : \text{false}]$ ✔️
- $[P] r [\text{ok} : \text{true}]$ ✗
- $[\text{true}] r [\text{ok} : P]$ ✗
- $[\text{wlp}(r, P)] r [\text{ok} : P]$ ✗
Question 2

Find a derivation for the IL triple

\[ \text{[true] if } x \geq y \text{ then } z := x \text{ else } z := y \text{ [ok : } z = \max(x, y)\]}

\[ \text{[true] if } x \geq y \text{ then}
\]
\[ [x \geq y]
\]
\[ z := x
\]
\[ [z = x \geq y] \equiv [x \geq y, z = \max(x, y)]
\]

\[ \text{else}
\]
\[ [x < y]
\]
\[ z := y
\]
\[ [z = y > x] \equiv [y > x, z = \max(x, y)]
\]
\[ [\text{ok : } z = \max(x, y)]\]
Question 3

Show that the following rule for assignment is not sound

\[
[P] \quad x := a [\text{ok} : P[a/x]]
\]

Consider the instance \([x = y] \quad x := 0 [\text{ok} : y = 0]\)
then \((x \mapsto 1, y \mapsto 0) \models (y = 0)\) but is not a reachable state!
Prove that rule [conj] is unsound

\[
\frac{[P_1] \ r \ [\epsilon : Q_1] \quad [P_2] \ r \ [\epsilon : Q_2]}{[P_1 \land P_2] \ r \ [\epsilon : Q_1 \land Q_2]} \quad [\text{conj}]
\]
* Exam 4

Is this “mixed” HL+IL inference rule valid?

\[
[P \land b] \ c \ [\text{ok : } P] \\
\{P\} \text{ while } b \text{ do } c \{P \land \lnot b\}
\]