Hoare Logic

Testing, Quality Assurance, and Maintenance
Winter 2020

Prof. Arie Gurfinkel

based on slides by Prof. Ruzica Piskac and others

UNIVERSITY OF
WATERLOO

I THINK YOU SHOULD BE MORE SPECIFIC HERE IN STEP TWO
THEN A MIRACLE OCCURS . . .
S. Harris

History

Program verification is almost as old as computer science

- e.g., Checking A Large Routine by Alan Turing, 1949

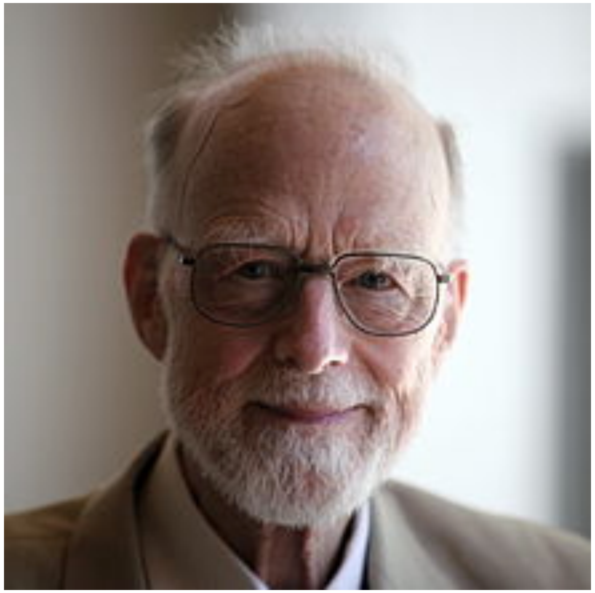
In late 60s, Floyd proposed rules for flow-charts and Hoare for structured languages

Since then, there has been many axiomatic semantics for many substantial languages and many applications / automation

- ESC/Java, SLAM, PCC, SPARK/Ada, KeY, Dafny, Viper, SeaHorn, …

Tony Hoare said…

“Thus the practice of proving programs would seem to lead to solution of three of the most pressing problems in software and programming, namely, reliability, documentation, and compatibility. However, program proving, certainly at present, will be **difficult** even for programmers of high caliber; and may be applicable only to quite simple program designs.”



-- C.A.R Hoare, *An Axiomatic Basis for Computer Programming*,1969

Tony Hoare also said…

“It has been found a serious problem to define these languages [ALGOL, FORTRAN, COBOL] with sufficient rigor to ensure compatibility among all implementations. ... one way to achieve this would be to insist that all implementations of the language shall satisfy the axioms and rules of inference which underlie proofs of properties of programs expressed in the language. In effect, this is equivalent to accepting the axioms and rules of inference as the ultimately definitive specification of the meaning of the language.”

Axiomatic Semantics

An axiomatic semantics consists of:

- a language for stating assertions about programs;
- rules for establishing the truth of assertions.

Some typical kinds of assertions:

- This program terminates.
- If this program terminates, the variables x and y have the same value throughout the execution of the program.
- The array accesses are within the array bounds.

Some typical languages of assertions

- First-order logic
- Other logics (temporal, linear, separation)
- Special-purpose specification languages (Z, Larch, JML)

Assertions for WHILE

The assertions we make about WHILE programs are of the form:

$$\{A\}\ c\ \{B\}$$

with the meaning that:

- If A holds in state *q* and *q* → *q'*
- then B holds in *q'*

A is the precondition and B is the post-condition

For example:

$$\{ y \le x \}\ z := x;\ z := z + 1\ \{ y < z \}$$

is a valid assertion

These are called Hoare triples or Hoare assertions

Assertions for WHILE

{A} c {B} is a partial correctness assertion. It does not imply termination of c.

- If A holds in state q and there exists q' such that $q \to q'$
- then B holds in state q'

[A] c [B] is a total correctness assertion meaning that

- If A holds in state q
- then there exists q' such that $q \to q'$ and B holds in state q'

Now let's be more formal

Formalize the language of assertions, A and B

Define when an assertion holds in a state

Define rules for deriving valid Hoare triples

The Assertion Language

We use first-order predicate logic with WHILE expressions

$$
\begin{aligned}
A ::= {} & \text{true} \mid \text{false} \mid e_1 = e_2 \mid e_1 \geq e_2 \\
& \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \Rightarrow A_2 \mid \forall x.A \mid \exists x.A
\end{aligned}
$$

We are somewhat sloppy and mix the logical variables and the program variables.

Implicitly, all WHILE variables range over integers.

All WHILE Boolean expressions are also assertions.

Semantics of Assertions

Notation $q \models A$ says that assertion A holds in a given state q.

- This is well-defined when q is defined on all variables occurring in A.

The $\models$ judgment is defined inductively on the structure of assertions.

It relies on the semantics of arithmetic expressions from WHILE.

Semantics of Assertions

$q \models \mathrm{true}$	always
$q \models e_1 = e_2$	iff $<e_1,q>\Downarrow \; = \; <e_2,q>\Downarrow$
$q \models e_1 \geq e_2$	iff $<e_1,q>\Downarrow \; \geq \; <e_2,q>\Downarrow$
$q \models A_1 \wedge A_2$	iff $q \models A_1$ and $\mathrm{q} \models A_2$
$q \models A_1 \vee A_2$	iff $q \models A_1$ or $q \models A_2$
$q \models A_1 \Rightarrow A_2$	iff $q \models A_1$ implies $q \models A_2$
$q \models \forall x.\mathrm{A}$	iff $\forall n \in \mathrm{Z}.\ q[x{:=}n] \models \mathrm{A}$
$q \models \exists x.\mathrm{A}$	iff $\exists n \in \mathrm{Z}.\ q[x{:=}n] \models \mathrm{A}$

Semantics of Hoare Triples

Now we can define formally the meaning of a *partial correctness* assertion:

$\models \{A\}\ c\ \{B\}$ iff

$$\forall q \in Q.\ \forall q' \in Q.\ q \models A \wedge q \xrightarrow{c} q' \quad \Rightarrow \quad q' \models B$$

and the meaning of a total correctness assertion:

$\models [A]\ c\ [B]$ iff $\forall q \in Q.\ q \models A \Rightarrow \exists q' \in Q.\ q \xrightarrow{c} q' \wedge q' \models B$

Examples of Hoare Triples

{ true } `x := 5` { }

{ } `x := x + 3` { x = y + 3 }

{ } `x := x*2 + 3` { x > 1 }

{ x = a } `if x < 0 then x := -x` { }

{ false } `x := 3` { }

{ x > 0 } `while x != 0 do x := x - 1` { }

{ x < 0 } `while x != 0 do x := x - 1` { }

Inferring Validity of Assertions

We now have the formal mechanism to decide when $\{A\}\ c\ \{B\}$

- But it is not satisfactory,
- because $\models \{A\}\ c\ \{B\}$ is defined in terms of the operational semantics.
- We practically have to run the program to verify an assertion.
- Thus, it is impossible to effectively verify the truth of a $\forall x.\ A$ assertion (by using the definition of validity)

We need to define a symbolic technique for deriving valid assertions from others that are known to be valid

- We start with validity of first-order formulas

Inference Rules (First Order Logic)

We write $\vdash A$ when A can be inferred from basic axioms

The inference rules for $\vdash A$ are the usual ones from first-order logic with arithmetic

Natural deduction style rules:

$$\frac{\vdash A \qquad \vdash B}{\vdash A \land B} \qquad \frac{\vdash A}{\vdash A \lor B} \qquad \frac{\vdash B}{\vdash A \lor B} \qquad \frac{\vdash A \Rightarrow B \quad \vdash A}{\vdash B}$$

$$\frac{\vdash A[e/x]}{\vdash \exists\, x.\, A} \qquad \frac{\vdash \forall\, x.\, A}{\vdash A[e/x]} \qquad \frac{\vdash A[a/x]}{\vdash \forall\, x.\, A}\ a \text{ is fresh}$$

$$\frac{\begin{array}{c}\vdash A \\ \dots \\ \vdash B\end{array}}{\vdash A \Rightarrow B} \qquad \frac{\vdash \exists\, x.\, A \qquad \begin{array}{c}\vdash A[a/x] \\ \dots \\ \vdash B\end{array}}{\vdash B}\ a \text{ is fresh}$$

Inference Rules for Hoare Triples

We write ⊢ {A} c {B} when we can derive the triple using inference rules

There is one inference rule for each command in the language

Plus, the *rule of consequence*

- e.g., strengthen pre-condition, weaken post-condition

$$\frac{\vdash A' \implies A \qquad \{A\}\, c\, \{B\} \qquad \vdash B \implies B'}{\{A'\}\, c\, \{B'\}}\ \textsc{Conseq}$$

Inference Rules for WHILE language

One rule for each syntactic construct:

$$\vdash \{A\}\ \texttt{skip}\ \{A\}$$

$$\frac{\vdash \{A\}\ s_1\ \{B\} \qquad \vdash \{B\}\ s_2\ \{C\}}{\vdash \{A\}\ s_1;\ s_2\ \{C\}}$$

$$\vdash \{A[e/x]\}\ x\texttt{:=}e\ \{A\}$$

$$\frac{\vdash \{A \wedge b\}\ s_1\ \{B\} \qquad \vdash \{A \wedge \neg b\}\ s_2\ \{B\}}{\vdash \{A\}\ \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{B\}}$$

$$\frac{\vdash \{\mathtt{I} \wedge b\}\ s\ \{\mathtt{I}\}}{\vdash \{\mathtt{I}\}\ \texttt{while}\ b\ \texttt{do}\ s\ \{\mathtt{I} \wedge \neg b\}}$$

Example: Conditional

$$\frac{\begin{array}{c} D1 :: \vdash \{\text{true} \wedge y \le 0\}\ \texttt{x := 1}\ \{x > 0\} \\ D2 :: \vdash \{\text{true} \wedge y > 0\}\ \texttt{x := y}\ \{x > 0\} \end{array}}{\vdash \{\text{true}\}\ \texttt{if y} \le \texttt{0 then x := 1 else x := y}\ \{x > 0\}}$$

D1 is obtained by the rules of consequence and assignment

$$\frac{\vdash \text{true} \wedge y \le 0 \Rightarrow 1 > 0 \qquad\qquad \vdash \{1 > 0\}\ \texttt{x := 1}\ \{x > 0\}}{\vdash \{\text{true} \wedge y \le 0\}\ \texttt{x := 1}\ \{x > 0\}}$$

D2 is obtained by the rules of consequence and assignment

$$\frac{\vdash \text{true} \wedge y > 0 \Rightarrow y > 0 \qquad\qquad \vdash \{y > 0\}\ \texttt{x := y}\ \{x > 0\}}{\vdash \{\text{true} \wedge y > 0\}\ \texttt{x := y}\ \{x > 0\}}$$

Complete proof on one slide

$$\dfrac{\textbf{R1}\ \dfrac{\vdash \text{true} \land y \leq 0 \implies 1 > 0 \qquad \vdash \{1 > 0\}\ x := 1\ \{x > 0\}}{\{\text{true} \land y \leq 0\}\ x := 1\ \{x > 0\}} \qquad \dfrac{\vdash \text{true} \land \neg(y \leq 0) \implies y > 0 \qquad \vdash \{y > 0\}\ x := y\ \{x > 0\}}{\{\text{true} \land \neg(y \leq 0)\}\ x := y\ \{x > 0\}}}{\{\text{true}\}\ \textbf{if}\ y \leq 0\ \textbf{then}\ x := 1\ \textbf{else}\ x := y\ \{x > 0\}}$$

Exercise: Hoare Rules

Is the following alternative rule for assignment still correct?

$$\vdash \{\text{true}\}\ x{:=}e\ \{x = e\}$$

No. The rule is not correct. For example, this is wrong!

$$\frac{\vdash \{\text{true}\}\ x := x + 1\ \{x = x + 1\} \qquad \vdash x = x + 1 \implies \text{false}}{\{\text{true}\}\ x := x + 1\ \{\text{false}\}}$$

Hoare Rules

For some constructs, multiple rules are possible

alternative “forward axiom” for assignment:

$$\vdash \{A\}\ x := e\ \{\exists x_0 \cdot x = e[x_0/x] \wedge A[x_0/x]\}$$

alternative rule for while loops:

$$\frac{\vdash I \wedge b \implies C \qquad \vdash \{C\}\ c\ \{I\} \qquad \vdash I \wedge \neg b \implies B}{\vdash \{I\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{B\}}$$

These alternative rules are derivable from the previous rules, plus the rule of consequence.

$$\frac{\vdash \{\mathtt{I} \wedge b\}\ s\ \{\mathtt{I}\}}{\vdash \{\mathtt{I}\}\quad \texttt{while}\ b\ \text{do}\ s\quad \{\mathtt{I} \wedge \neg b\}}$$

Example: a simple loop

We want to infer that

$$\vdash \{x \le 0\}\ \texttt{while}\ x \le 5\ \texttt{do}\ x := x + 1\ \{x = 6\}$$

Use the rule for while with invariant $\mathtt{I} \equiv x \le 6$

$$\dfrac{\dfrac{\vdash x \le 6 \land x \le 5 \Rightarrow x + 1 \le 6 \qquad \vdash \{x + 1 \le 6\}\ x := x + 1\ \{x \le 6\}}{\vdash \{x \le 6 \land x \le 5\}\ x := x + 1\ \{x \le 6\}}}{\vdash \{x \le 6\}\ \texttt{while}\ x \le 5\ \texttt{do}\ x := x + 1\ \{\ x \le 6 \land x > 5\}}$$

Then finish-off with the rule of consequence

$$\dfrac{\vdash x \le 0 \Rightarrow x \le 6 \qquad \vdash x \le 6 \land x > 5 \Rightarrow x = 6 \qquad \vdash \{x \le 6\}\ \texttt{while}\ \ldots\ \{x \le 6 \land x > 5\}}{\vdash \{x \le 0\}\ \texttt{while}\ \ldots\ \{x = 6\}}$$

Complete proof on one slide

$$\dfrac{\vdash x \leq 0 \implies x \leq 6 \qquad \textbf{R1}\,\dfrac{\dfrac{\vdash x \leq 6 \wedge x \leq 5 \implies x+1 \leq 6 \qquad \vdash \{x+1 \leq 6\}\ x := x+1\ \{x \leq 6\}}{\{x \leq 6 \wedge x \leq 5\}\ x := x+1\ \{x \leq 6\}}}{\{x \leq 6\}\ \textbf{while}\ x \leq 5\ \textbf{do}\ x := x+1\ \{\neg(x \leq 5) \wedge x \leq 6\}} \qquad \vdash \neg x \leq 5 \wedge x \leq 6 \implies x = 6}{\{x \leq 0\}\ \textbf{while}\ x \leq 5\ \textbf{do}\ x := x+1\ \{x = 6\}}$$

Inductive Loop Invariants

$$\frac{\vdash \mathrm{Pre} \Rightarrow \mathrm{Inv} \qquad \vdash \{\,\mathrm{Inv} \wedge b\,\}\ s\ \{\,\mathrm{Inv}\,\} \qquad \vdash \mathrm{Inv} \wedge \neg b \Rightarrow \mathrm{Post}}{\{\,\mathrm{Pre}\,\}\ \texttt{while b do s}\ \{\,\mathrm{Post}\,\}}$$

Inv is an inductive loop invariant if the following three conditions hold:

- (Initiation) Inv holds **initially** whenever the loop is reached. That is, it is true of the pre-condition *Pre*
- (Consecution) Inv is **preserved**: executing the loop body c from any state satisfying Inv and loop condition b ends in a state satisfying Inv
- (Safety) Inv is **strong enough**: Inv and the negation of loop condition b imply the desired post-condition *Post*

Proving simple loop using only one rule

We want to infer that

$$\vdash \{x \leq 0\}\ \texttt{while}\ x \leq 5\ \texttt{do}\ x := x + 1\ \{x = 6\}$$

Using inductive invariant x≤6

$$\frac{\vdash x \leq 0 \Rightarrow x\leq 6 \quad \vdash \{x\leq 6 \wedge x\leq 5\}\ \texttt{x:=x+1}\ \{x\leq 6\} \quad \vdash x>5 \wedge x \leq 6 \Rightarrow x=6}{\{x \leq 0\}\ \texttt{while x<=5 do x:=x+1}\ \{x=6\}}$$

Example: a more interesting program

We want to derive that

$\{n \geq 0\}$

```
p := 0;
x := 0;
while x < n do
  x := x + 1;
  p := p + m
```

$\{p = n * m\}$

Example: a more interesting program

Only applicable rule (except for rule of consequence):

$$\frac{\vdash \{A\}\ c_1\{C\} \quad \vdash \{C\}\ c_2\ \{B\}}{\vdash \{A\}\ c_1;\ c_2\ \{B\}}$$

$$\frac{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{C\} \qquad \vdash\{C\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}{\vdash \underbrace{\{n \geq 0\}}_{A}\ \underbrace{p:=0;\ x:=0}_{c_1};\ \underbrace{\text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)}_{c_2}\ \underbrace{\{p = n * m\}}_{B}}$$

Example: a more interesting program

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$$\frac{\vdash \{I \land b\}\ c\ \{I\}}{\vdash \{I\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{I \land \neg b\}}$$

We can match {I} with {C} but we cannot match {I ∧ ¬b} and {p = n * m} directly. Need to apply the rule of consequence first!

$$\frac{\vdash \{n \geq 0\}\ \text{p:=0; x:=0}\ \{C\} \qquad \vdash \{C\}\ \text{while x < n do (x:=x+1; p:=p+m)}\ \{p = n * m\}}{\vdash \underbrace{\{n \geq 0\}}_{A}\ \underbrace{\text{p:=0; x:=0;}}_{c_1}\ \underbrace{\text{while x < n do (x:=x+1; p:=p+m)}}_{c_2}\ \underbrace{\{p = n * m\}}_{B}}$$

Example: a more interesting program

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$$\frac{\vdash \{I \wedge b\}\ c\ \{I\}}{\vdash \underbrace{\{I\}}_{A}\ \underbrace{\texttt{while}\ b\ \texttt{do}\ c}_{c'}\ \underbrace{\{I \wedge \neg b\}}_{B}}$$

Rule of consequence:

$$\frac{\vdash A' \Rightarrow A \qquad \vdash \{A\}\ c'\ \{B\} \qquad \vdash B \Rightarrow B'}{\vdash \{A'\}\ c'\ \{B'\}}$$

$I = A = A' = C$

$$\frac{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{C\} \qquad \vdash \overbrace{\{C\}}^{A'}\ \overbrace{\text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)}^{c'}\ \overbrace{\{p = n * m\}}^{B'}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0;\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}$$

Example: a more interesting program

What is I? Let’s keep it as a placeholder for now!

Next applicable rule:

$$\frac{\vdash \{A\}\ c_1 \{C\} \qquad \vdash \{C\}\ c_2\ \{B\}}{\vdash \{A\}\ c_1;\ c_2\ \{B\}}$$

$$\dfrac{\vdash \{n \geq 0\}\ \text{p:=0; x:=0}\ \{I\} \qquad \dfrac{\dfrac{\vdash \overbrace{\{I \wedge x<n\}}^{A}\ \overbrace{\text{x := x+1;}}^{c_1}\ \overbrace{\text{p:=p+m}}^{c_2}\ \overbrace{\{I\}}^{B}}{\vdash \{I\}\ \text{while x < n do (x:=x+1; p:=p+m)}\ \{I \wedge x \geq n\}} \qquad \vdash I \wedge x \geq n \Rightarrow p = n * m}{\vdash \{I\}\ \text{while x < n do (x:=x+1; p:=p+m)}\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ \text{p:=0; x:=0; while x < n do (x:=x+1; p:=p+m)}\ \{p = n * m\}}$$

Example: a more interesting program

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$\vdash \{A[e/x]\}\ x{:=}e\ \{A\}$

$$\dfrac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{I\} \qquad \dfrac{\dfrac{\dfrac{\vdash \overbrace{\{I \land x<n\}}^{A}\ \overbrace{x := x+1}^{c_1}\ \{C\} \qquad \vdash \{C\}\ \overbrace{p{:=}p+m}^{c_2}\ \overbrace{\{I\}}^{B}}{\vdash \{I \land x<n\}\ x := x+1;\ p{:=}p+m\ \{I\}}}{\vdash \{I\}\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{I \land x \geq n\}} \qquad \vdash I \land x \geq n \Rightarrow p = n * m}{\vdash \{I\}\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0;\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}$$

Example: a more interesting program

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$\vdash \{A[e/x]\}\ x{:=}e\ \{A\}$

$$\dfrac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{I\} \qquad \dfrac{\dfrac{\dfrac{\vdash\{I \land x<n\}\ x{:=}x+1\ \{I[p+m/p]\} \qquad \vdash\{I[p+m/p]\}\ p{:=}p+m\ \{I\}}{\vdash\{I \land x<n\}\ x{:=}x+1;\ p{:=}p+m\ \{I\}}}{\vdash\{I\}\ \text{while}\ x < n\ \text{do}\ (x{:=}x+1;\ p{:=}p+m)\ \{I \land x \geq n\}} \qquad \vdash I \land x \geq n \Rightarrow p = n * m}{\vdash\{I\}\ \text{while}\ x < n\ \text{do}\ (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0;\ \text{while}\ x < n\ \text{do}\ (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}$$

Example: a more interesting program

Only applicable rule (except for rule of consequence):

$\vdash \{A[e/x]\}\ x{:=}e\ \{A\}$

Need rule of consequence to match $\{I \wedge x<n\}$ and $\{I[x+1/x,\ p+m/p]\}$

$$\dfrac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{I\} \qquad \dfrac{\dfrac{\dfrac{\vdash\{I \wedge x<n\}\ x{:=}x+1\ \{I[p+m/p]\} \qquad \vdash\{I[p+m/p]\}\ p{:=}p+m\ \{I\}}{\vdash\{I \wedge x<n\}\ x{:=}x+1;\ p{:=}p+m\ \{I\}}}{\vdash\{I\}\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{I \wedge x \geq n\}} \qquad \vdash I \wedge x \geq n \Rightarrow p = n * m}{\vdash\{I\}\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0;\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}$$

Example: a more interesting program

Let’s just remember the open proof obligations!

$$\dfrac{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{I\} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdash \{I[x+1/x,\ p+m/p]\}\ x:=x+1\ \{I[p+m/p]\} \qquad \vdash I \land x < n \Rightarrow I[x+1/x,\ p+m/p]}{\vdash \{I \land x<n\}\ x:=x+1\ \{I[p+m/p]\}} \qquad \vdash \{I[p+m/p]\}\ p:=p+m\ \{I\}}{\vdash \{I \land x<n\}\ x:=x+1;\ p:=p+m\ \{I\}}}{\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{I \land x \geq n\}} \qquad \vdash I \land x \geq n \Rightarrow p = n * m}{\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0;\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}$$

Example: a more interesting program

Let’s just remember the open proof obligations!

$$\vdash I \wedge x < n \Rightarrow I[x+1/x, p+m/p]$$

$$\vdash I \wedge x \geq n \Rightarrow p = n * m$$

Continue with the remaining part of the proof tree, as before.

Now we only need to solve the remaining constraints!

$$\dfrac{\dfrac{\dfrac{\vdash n \geq 0 \Rightarrow I[0/p, 0/x] \qquad \vdash \{I[0/p, 0/x]\}\ p:=0\ \{I[0/x]\}}{\vdash \{n \geq 0\}\ p:=0\ \{I[0/x]\}} \qquad \vdash \{I[0/x]\}\ x:=0\ \{I\}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{I\}} \qquad \dfrac{\vdots}{\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0;\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}$$

Example: a more interesting program

Find I such that all constraints are simultaneously valid:

$\vdash n \geq 0 \Rightarrow \mathtt{I}[0/p, 0/x]$

$\vdash \mathtt{I} \wedge x < n \Rightarrow \mathtt{I}[x+1/x, p+m/p]$

$\vdash \mathtt{I} \wedge x \geq n \Rightarrow p = n * m$

$\mathtt{I} \equiv p = x * m \wedge x \leq n$

$\vdash n \geq 0 \Rightarrow 0 = 0 * m \wedge 0 \leq n$

$\vdash p = p * m \wedge x \leq n \wedge x < n \Rightarrow p+m = (x+1) * m \wedge x+1 \leq n$

$\vdash p = x * m \wedge x \leq n \wedge x \geq n \Rightarrow p = n * m$

All constraints are valid!

Back to the example: What did we just do?!

$\{n \geq 0\}$

```
p := 0;
x := 0;
while x < n inv p=x*m ∧ x≤n do
  x := x + 1;
  p := p + m
```

$\{p = n * m\}$

Using Hoare Rules

Hoare rules are mostly syntax directed

There are three obstacles to automation of Hoare logic proofs:

- When to apply the rule of consequence?
- What invariant to use for while?
- How do you prove the implications involved in the rule of consequence?

The last one is how theorem proving gets in the picture

- This turns out to be doable!
- The loop invariants turn out to be the hardest problem!
- Should the programmer give them?

Hoare Logic: Summary

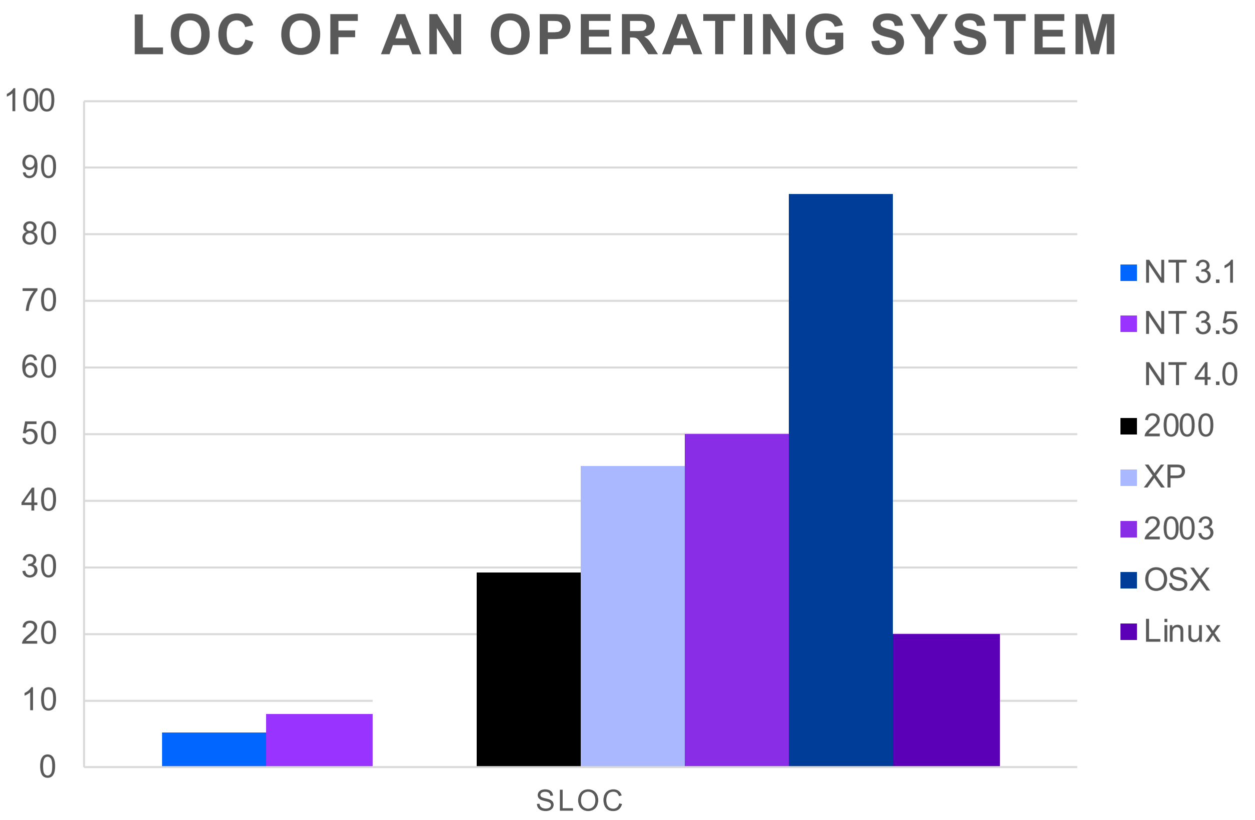
We have a language for asserting properties of programs.

We know when such an assertion is true.

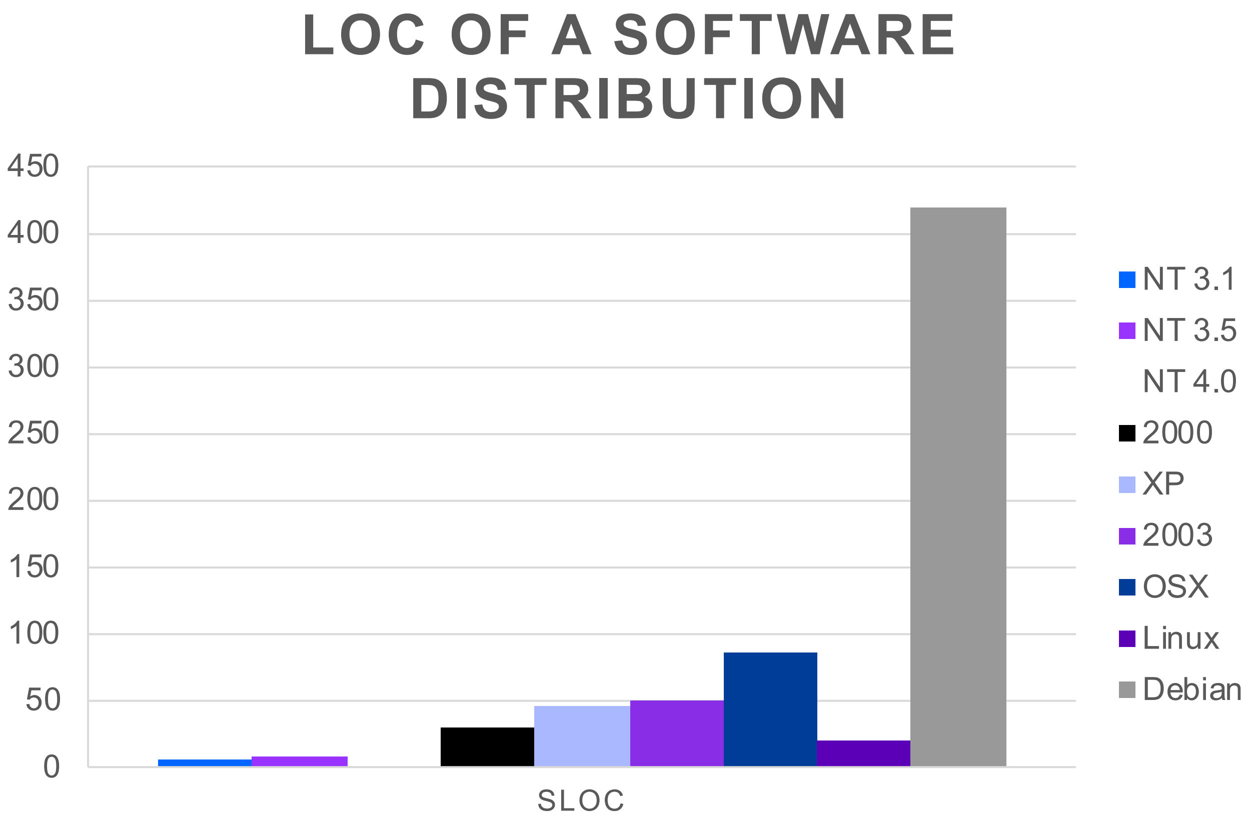
We also have a symbolic method for deriving assertions.

$$\begin{tikzcd}
\begin{array}{c} A \\ \{A\}\ P\ \{B\} \end{array} \arrow[rr, bend left, "\text{semantics}"] \arrow[dr, bend right, "\text{theorem proving}"'] & & \begin{array}{c} \vDash A \\ \vDash \{A\}\ P\ \{B\} \end{array} \arrow[dl, bend left, "\text{completeness}"] \\
& \begin{array}{c} \vdash A \\ \vdash \{A\}\ P\ \{B\} \end{array} \arrow[ur, bend left, "\text{soundness}"] &
\end{tikzcd}$$

Software Size: Operating System

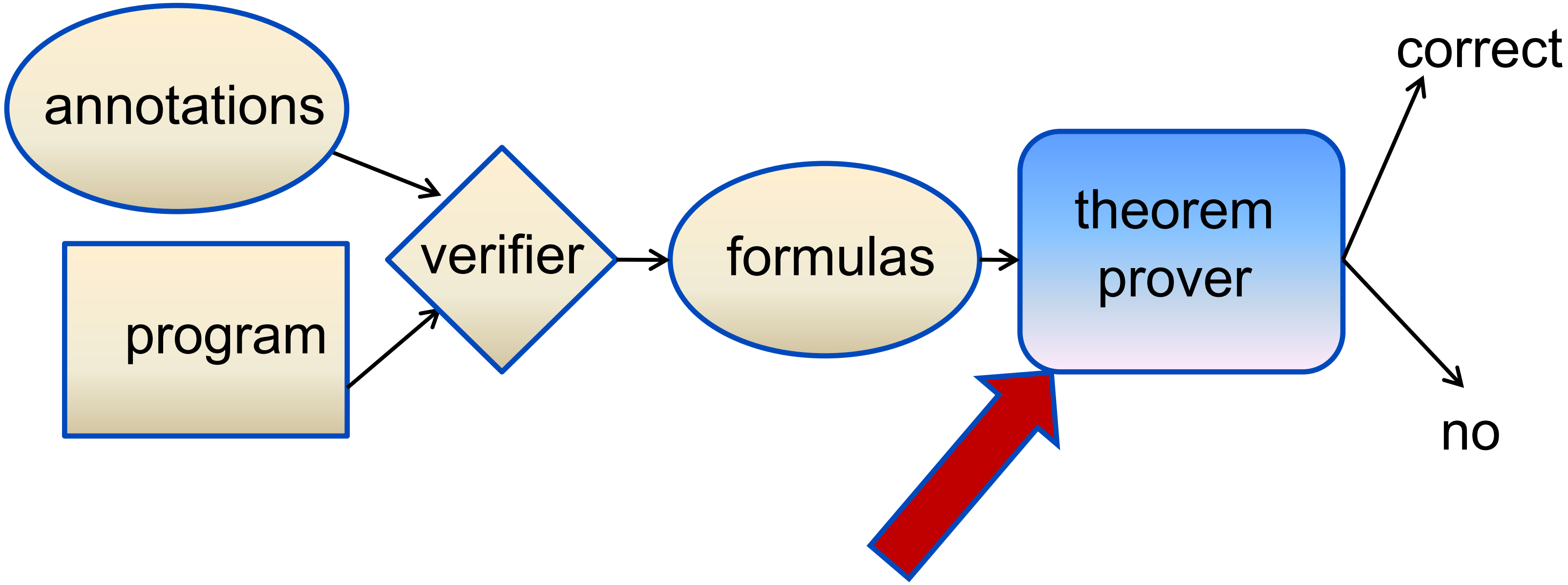


Software Size: Software Distribution



Verification must be automated!

Software Verification (with Dafny)



Prove formulas automatically!