Lecture 12–13: Hoare Logic

17-355/17-665/17-819: Program Analysis

Rohan Padhye

October 21 & 23, 2025

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Heads up: Course Projects

- Scope: ~3 weeks of effort at end of course
- Some options
  - Implement a non-trivial analysis and evaluate it on some code
  - Empirically evaluate an existing analysis tool
  - Contribute meaningfully to an open source analysis tool
  - Explore an extension to the state of the art in program analysis
- Students in the Masters version (17-665) must engage with non-trivial codebases
  - Either the analysis framework or the target program must be in active use by the developer community
- Students in the Ph.D. version (17-819) must engage in research in some way
  - OK to extend your current research work – can be empirical as well

Logical Reasoning about Code

- So far, we've reasoned about code using operational semantics
  - And built program analyses that abstract those semantics
- ***Axiomatic semantics*** define meaning of a program in terms of assertions
  - Enables logic-based reasoning about code
- Enables ***verification***
  - Prove arbitrary properties about code – not just ones built into a particular analysis
  - Goes back to Turing (1949): "Checking a Large Routine"
  - Hoare developed rules in the 1960s for verifying the WHILE 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
- Assertions are in a logic, e.g. first-order logic
  - Alternatives include temporal logic, linear logic, etc.

Assertion Language

$$
\begin{array}{rclllll}
P & ::= & \mathsf{true} & \mid \mathsf{false} & \mid e_1 = e_2 & \mid e_1 \geqslant e_2 & \mid P_1 \wedge P_2 \\
 & \mid & P_1 \vee P_2 & \mid P_1 \Rightarrow P_2 & \mid \forall x.P & \mid \exists x.P &
\end{array}
$$

- We'll be a bit sloppy and mix logical and program variables like x
- We'll treat Boolean expressions as a special case of assertions

Hoare Triple

$$\{ P \} \; S \; \{ Q \}$$

- P is the precondition
- Q is the postcondition
- S is any statement (in WHILE, at least for our class)

- Semantics: if P holds in some state E and if $<S; E> \Downarrow E'$, then Q holds in E'
  - This is *partial correctness*: termination of S is not guaranteed
  - *Total correctness* additionally implies termination, and is written $[P] \; S \; [Q]$

Exercise: Exploring Hoare Triples

- What are reasonable pre- or post- conditions for the following incomplete Hoare triples?

1. { true } x := 5 { }
2. { } x : = x + 3 { x = y + 3 }
3. { } x := x * 2 + 3 { x > 1 }
4. { x = a } if (x < 0) then x := -x { }
5. { false } x := 3 { }
6. { x < 0 } while (x != 0) x := x – 1 { }

Hoare Triple

$$\{\, P \,\}\; S \;\{\, Q \,\}$$

- P is the precondition
- Q is the postcondition
- S is any statement (in WHILE, at least for our class)

- Semantics: if P holds in some state E and if $<S;\, E> \Downarrow E'$, then Q holds in E'
  - This is *partial correctness*: termination of S is not guaranteed
  - *Total correctness* additionally implies termination, and is written $[P]\; S\; [Q]$

Assertion Semantics

- $E \vDash P$ means P is true in E
- Rules:

$$
\begin{array}{ll}
E \vDash \mathbf{true} & \textit{always} \\
E \vDash a_1 = a_2 & \textit{iff } \langle E, a_1\rangle \Downarrow n \text{ and } \langle E, a_2\rangle \Downarrow n \\
E \vDash a_1 \geqslant a_2 & \textit{iff } \langle E, a_1\rangle \Downarrow n_1, \langle E, a_2\rangle \Downarrow n_2, \text{ and } n_1 \geqslant n_2 \\
E \vDash P_1 \wedge P_2 & \textit{iff } E \vDash P_1 \text{ and } E \vDash P_2 \\
\ldots & \\
E \vDash \forall x.P & \textit{iff } \forall n \in \mathbb{Z}.E[x \mapsto n] \vDash P \\
E \vDash \exists x.P & \textit{iff } \exists n \in \mathbb{Z}.E[x \mapsto n] \vDash P
\end{array}
$$

Semantics of Hoare Triples

- A partial correctness assertion $\models \{P\}\; S\; \{Q\}$ is defined formally to mean:

$$\forall E.\forall E'.(E \models P \wedge \langle E, S\rangle \Downarrow E') \Rightarrow E' \models Q$$

- How would we define total correctness $[P]\; S\; [Q]$?

- This is a good formal definition—but it doesn't help us prove many assertions because we have to reason about all environments. How can we do better?

Derivation Rules for Logical Formulas

- We can define rules for proving the validity of logical formulas
  - $\vdash P$ is read "we can prove P"
- Example rule:

$$\frac{\vdash P \quad \vdash Q}{\vdash P \wedge Q}\ \mathit{and}$$

Derivation Rules for Hoare Logic

- Judgment form $\vdash \{P\}\ S\ \{Q\}$ means "we can prove the Hoare triple $\{P\}\ S\ \{Q\}$"

$$\frac{}{\vdash \{P\}\ \text{skip}\ \{P\}}\ \textit{skip} \qquad \frac{}{\vdash \{[a/x]P\}\ x{:=}a\ \{P\}}\ \textit{assign}$$

$$\frac{\vdash \{P\}\ S_1\ \{P'\} \qquad \vdash \{P'\}\ S_2\ \{Q\}}{\vdash \{P\}\ S_1;\ S_2\ \{Q\}}\ \textit{seq} \qquad \frac{\vdash \{P \wedge b\} S_1 \{Q\} \qquad \vdash \{P \wedge \neg b\}\ S_2\ \{Q\}}{\vdash \{P\}\ \text{if}\ b\ \text{then}\ S_1\ \text{else}\ S_2\ \{Q\}}\ \textit{if}$$

$$\frac{\vdash P' \Rightarrow P \qquad \vdash \{P\}\ S\ \{Q\} \qquad \vdash Q \Rightarrow Q'}{\vdash \{P'\}\ S\ \{Q'\}}\ \textit{consq}$$

- Question: What should be the rule for while b do S?

Derivation Rules for Hoare Logic

- Key point: Post-condition of loop body would need to be the pre-condition for next iteration (assuming loop condition continues to hold). This is the ”loop invariant”.

$$\frac{\vdash \{P \wedge b\} S \{P\}}{\vdash \{P\}\ \textbf{while}\ b\ \textbf{do}\ S \{P \wedge \neg b\}}\ \mathit{while}$$

Soundness and Completeness

- Sound: $\texttt{if} \ \vdash \{P\}S\{Q\} \ \texttt{then} \ \models \{P\}S\{Q\}$
- Complete: $\texttt{if} \ \models \{P\}S\{Q\} \ \texttt{then} \ \vdash \{P\}S\{Q\}$

Strongest Postconditions

- Here are a number of valid Hoare Triples:
  - {x = 5} x := x * 2 { true }
  - {x = 5} x := x * 2 { x > 0 }
  - {x = 5} x := x * 2 { x = 10 || x = 5 }
  - {x = 5} x := x * 2 { x = 10 }
- Which one is best?

Strongest Postconditions

- Here are a number of valid Hoare Triples:
  - {x = 5} x := x * 2 { true }
  - {x = 5} x := x * 2 { x > 0 }
  - {x = 5} x := x * 2 { x = 10 || x = 5 }
  - {x = 5} x := x * 2 { x = 10 }
    - All are true, but this one is the most *useful*
    - x=10 is the *strongest postcondition*
- If {P} S {Q} and for all Q' such that {P} S {Q'}, Q $\Rightarrow$ Q', then Q is the strongest postcondition of S with respect to P
  - check: x = 10 $\Rightarrow$ true
  - check: x = 10 $\Rightarrow$ x > 0
  - check: x = 10 $\Rightarrow$ x = 10 || x = 5
  - check: x = 10 $\Rightarrow$ x = 10

Weakest Preconditions

- Here are a number of valid Hoare Triples:
  - $\{x = 5 \;\&\&\; y = 10\}$ $z := x / y$ $\{ z < 1 \}$
  - $\{x < y \;\&\&\; y > 0\}$ $z := x / y$ $\{ z < 1 \}$
  - $\{y \neq 0 \;\&\&\; x / y < 1\}$ $z := x / y$ $\{ z < 1 \}$
- Which one is best?

Weakest Preconditions

- Here are a number of valid Hoare Triples:
  - $\{x = 5 \;\&\&\; y = 10\}$ $z := x / y$ $\{ z < 1 \}$
  - $\{x < y \;\&\&\; y > 0\}$ $z := x / y$ $\{ z < 1 \}$
  - $\{y \neq 0 \;\&\&\; x / y < 1\}$ $z := x / y$ $\{ z < 1 \}$
    - All are true, but this one is the most *useful* because it allows us to invoke the program in the most general condition
    - $y \neq 0 \;\&\&\; x / y < 1$ is the *weakest precondition*
- If {P} S {Q} and for all P’ such that {P’} S {Q}, P’ $\Rightarrow$ P, then P is the weakest precondition *wp*(S,Q) of S with respect to Q

Hoare Triples and Weakest Preconditions

- Theorem: {P} S {Q} holds if and only if P $\Rightarrow$ *wp*(S,Q)
  - In other words, a Hoare Triple is still valid if the precondition is stronger than necessary, but not if it is too weak
  - Can use this to prove {P} S {Q} by computing wp(S,Q) and checking implication.

- Question: Could we state a similar theorem for a strongest postcondition function?
  - e.g. {P} S {Q} holds if and only if *sp*(S,P) $\Rightarrow$ Q
  - A: Yes, but it's harder to compute (see text for why)

Exercise: More Hoare Triples

Consider the following Hoare triples:

A) { z = y + 1 } x := z * 2 { x = 4 }

B) { y = 7 } x := y + 3 { x > 5 }

C) { false } x := 2 / y { true }

D) { y < 16 } x := y / 2 { x < 8 }

- Which of the Hoare triples above are valid?
- Considering the valid Hoare triples, for which ones can you write a stronger postcondition? (Leave the precondition unchanged, and ensure the resulting triple is still valid)
- Considering the valid Hoare triples, for which ones can you write a weaker precondition? (Leave the postcondition unchanged, and ensure the resulting triple is still valid)

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?
    - What is most general value of y such that 3 + y > 0?
    - y > -3

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := e, P) = [e/x] P
    - Resulting triple: { [e/x] P } x := e { P }

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := e, P) = [e/x] P
    - Resulting triple: { [e/x] P } x := e { P }
  - [3 / x] (x + y > 0)
  - = (3) + y > 0
  - = y > -3

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := e, P) = [e/x] P

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := e, P) = [e/x] P
  - [3*y+z / x] (x * y – z > 0)

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := e, P) = [e/x] P
  - [3*y+z / x] (x * y – z > 0)
  - = (3*y+z) * y - z > 0

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := e, P) = [e/x] P
  - [3*y+z / x] (x * y – z > 0)
  - = (3*y+z) * y - z > 0
  - = 3*y^2 + z*y - z > 0

Hoare Logic Rules

- Sequence
  - $\{ P \}\ x := x + 1;\ y := x + y\ \{ y > 5 \}$
  - What is the weakest precondition P?

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - $wp(S;T, Q) = wp(S, wp(T, Q))$
  - $wp(x:=x+1;\ y:=x+y,\ y>5)$

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)
  - = *wp*(x:=x+1, *wp*(y:=x+y, y>5))

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)
  - = *wp*(x:=x+1, *wp*(y:=x+y, y>5))
  - = *wp*(x:=x+1, x+y>5)

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)
  - = *wp*(x:=x+1, *wp*(y:=x+y, y>5))
  - = *wp*(x:=x+1, x+y>5)
  - = x+1+y>5

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)
  - = *wp*(x:=x+1, *wp*(y:=x+y, y>5))
  - = *wp*(x:=x+1, x+y>5)
  - = x+1+y>5
  - = x+y>4

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - *wp*(if B then S else T, Q) = B $\Rightarrow$ *wp*(S,Q) && $\neg$B $\Rightarrow$ *wp*(T,Q)
  - *wp*(if x>0 then y:=z else y:=-z, y>5)

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - $wp(\text{if } B \text{ then } S \text{ else } T, Q) = B \Rightarrow wp(S,Q) \ \&\& \ \neg B \Rightarrow wp(T,Q)$
  - $wp(\text{if } x>0 \text{ then } y{:=}z \text{ else } y{:=}{-z}, y>5) = x>0 \Rightarrow wp(y{:=}z, y>5) \ \&\& \ x\leq 0 \Rightarrow wp(y{:=}{-z}, y>5)$

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - $wp(\text{if B then S else T, Q}) = B \Rightarrow wp(S,Q)\ \&\&\ \neg B \Rightarrow wp(T,Q)$
  - $wp(\text{if x>0 then y:=z else y:=-z, y>5}) = x>0 \Rightarrow wp(y{:=}z, y>5)\ \&\&\ x \leq 0 \Rightarrow wp(y{:=}{-z}, y>5)$

    $= x>0 \Rightarrow z > 5\ \&\&\ x \leq 0 \Rightarrow -z > 5$

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - $wp(\text{if B then S else T, Q}) = B \Rightarrow wp(S,Q)\ \&\&\ \neg B \Rightarrow wp(T,Q)$
  - $wp(\text{if } x>0 \text{ then } y:=z \text{ else } y:=-z,\ y>5) = x>0 \Rightarrow wp(y:=z, y>5)\ \&\&\ x\leq 0 \Rightarrow wp(y:=-z, y>5)$

$$= x>0 \Rightarrow z > 5\ \&\&\ x\leq 0 \Rightarrow -z > 5$$

$$= x>0 \Rightarrow z > 5\ \&\&\ x\leq 0 \Rightarrow z < -5$$

Hoare Logic Rules

- Loops
  - { P } while (i < x) f=f*i; i := i + 1 { f = x! }
  - What is the weakest precondition P?

Hoare Logic Rules

- Loops
  - { P } while (i < x) f=f*i; i := i + 1 { f = x! }
  - What is the weakest precondition P?

- Intuition
  - Must prove by induction
    - Only way to generalize across number of times loop executes
  - Need to guess induction hypothesis
    - Base case: precondition P
    - Inductive case: should be preserved by executing loop body

Proving loops correct

- First consider *partial correctness*
  - The loop may not terminate, but if it does, the postcondition will hold
- {P} while B do S {Q}
  - Find an invariant Inv such that:
    - P $\Rightarrow$ Inv
      - The invariant is initially true
    - { Inv && B } S {Inv}
      - Each execution of the loop preserves the invariant
    - (Inv && $\neg$B) $\Rightarrow$ Q
      - The invariant and the loop exit condition imply the postcondition

Practice: Loop Invariants

Consider the following program:

```
{ N >= 0 }
i := 0;
while (i < N) do
   i := N
{ i = N }
```

Correctness Conditions

$P \Rightarrow Inv$

The invariant is initially true

{ Inv && B } S {Inv}

Loop preserves the invariant

$(Inv \ \&\& \ \neg B) \Rightarrow Q$

Invariant and exit implies postcondition

Which of the following loop invariants are correct?
For those that are incorrect, explain why.

A) i = 0

B) i = N

C) N >= 0

D) i <= N

Loop Example

$\{ N \geq 0 \}$

```
j := 0;
s := 0;

while (j < N) do

   j := j + 1;
   s := s + a[j];

end
```

$\{ s = (\Sigma i \mid 0 \leq i < N \bullet a[i]) \}$

Loop Example

$\{ N \geq 0 \}$

```
j := 0;
s := 0;

while (j < N) do

   j := j + 1;
   s := s + a[j];

end
```

$\{ s = (\Sigma i \mid 0 \leq i < N \bullet a[i]) \}$

How can we find a loop invariant?

Loop Example

$\{ N \geq 0 \}$
j := 0;
s := 0;

How can we find a loop invariant?

while (j < N) do

  j := j + 1;
  s := s + a[j];

Replace N with j
Add information on range of j
Result: $0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i])$

end
$\{ s = (\Sigma i \mid 0 \leq i < N \bullet a[i]) \}$

Loop Example

```
{ N ≥ 0 }
j := 0;
s := 0;
{ 0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }
while (j < N) do
   {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}
   j := j + 1;
   s := s + a[j];
   {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }
end
{ s = (Σi | 0≤i<N • a[i]) }
```

Loop Example

```
{ N ≥ 0 }
j := 0;
s := 0;
{ 0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }
while (j < N) do
   {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}
   j := j + 1;
   s := s + a[j];
   {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }
end
{ s = (Σi | 0≤i<N • a[i]) }
```

Proof obligation #1

Proof obligation #2

&& j ≥ N

Proof obligation #3

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  j := 0;

  s := 0;

  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$
  j := 0;
  s := 0;
  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$
- Invariant is maintained

  $\{0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \ \&\& \ j < N\}$
  j := j + 1;
  s := s + a[j];
  $\{0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  j := 0;

  s := 0;

  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$
- Invariant is maintained

  $\{0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \ \&\& \ j < N\}$

  j := j + 1;

  s := s + a[j];

  $\{0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$
- Invariant and exit condition imply postcondition

  $0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \ \&\& \ j \geq N$

  $\Rightarrow s = (\Sigma i \mid 0 \leq i < N \bullet a[i])$

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  j := 0;

  s := 0;

  $\{ 0 \leq j \leq N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  j := 0;

  $\{ 0 \leq j \leq N \ \&\&\ \mathbf{0} = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$ *// by assignment rule*

  s := 0;

  $\{ 0 \leq j \leq N \ \&\&\ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  $\{ 0 \leq \mathbf{0} \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < \mathbf{0} \bullet a[i]) \}$ *// by assignment rule*

  j := 0;

  $\{ 0 \leq j \leq N \ \&\& \ \mathbf{0} = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$ *// by assignment rule*

  s := 0;

  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  $\{ 0 \leq \mathbf{0} \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < \mathbf{0} \cdot a[i]) \}$ *// by assignment rule*

  j := 0;

  $\{ 0 \leq j \leq N \ \&\& \ \mathbf{0} = (\Sigma i \mid 0 \leq i < j \cdot a[i]) \}$ *// by assignment rule*

  s := 0;

  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \cdot a[i]) \}$

- Need to show that:

  $(N \geq 0) \Rightarrow (0 \leq 0 \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < 0 \cdot a[i]))$

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  $\{ 0 \leq \mathbf{0} \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < \mathbf{0} \bullet a[i]) \}$ *// by assignment rule*

  j := 0;

  $\{ 0 \leq j \leq N \ \&\& \ \mathbf{0} = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$ *// by assignment rule*

  s := 0;

  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

- Need to show that:

  $(N \geq 0) \Rightarrow (0 \leq 0 \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < 0 \bullet a[i]))$

$= (N \geq 0) \Rightarrow (0 \leq N \ \&\& \ 0 = \mathbf{0})$ *// 0 ≤ 0 is true, empty sum is 0*

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$

  $\{ 0 \leq \mathbf{0} \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < \mathbf{0} \bullet a[i]) \}$ *// by assignment rule*

  j := 0;

  $\{ 0 \leq j \leq N \ \&\& \ \mathbf{0} = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$ *// by assignment rule*

  s := 0;

  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

- Need to show that:

  $(N \geq 0) \Rightarrow (0 \leq 0 \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < 0 \bullet a[i]))$

$= (N \geq 0) \Rightarrow (0 \leq N \ \&\& \ 0 = \mathbf{0})$ *// 0 ≤ 0 is true, empty sum is 0*

$= (N \geq 0) \Rightarrow (0 \leq N)$ *// 0=0 is true, P && true is P*

Proof Obligations

- Invariant is initially true

  $\{ N \geq 0 \}$
  $\{ 0 \leq \mathbf{0} \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < \mathbf{0} \bullet a[i]) \}$ *// by assignment rule*
  j := 0;
  $\{ 0 \leq j \leq N \ \&\& \ \mathbf{0} = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$ *// by assignment rule*
  s := 0;
  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$

- Need to show that:

  $(N \geq 0) \Rightarrow (0 \leq 0 \leq N \ \&\& \ 0 = (\Sigma i \mid 0 \leq i < 0 \bullet a[i]))$
  $= (N \geq 0) \Rightarrow (0 \leq N \ \&\& \ 0 = \mathbf{0})$ *// 0 ≤ 0 is true, empty sum is 0*
  $= (N \geq 0) \Rightarrow (0 \leq N)$ *// 0=0 is true, P && true is P*
  = **true**

Proof Obligations

- Invariant is maintained

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}

  j := j + 1;

  s := s + a[j];

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }

Proof Obligations

- Invariant is maintained

  $\{0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \ \&\& \ j < N\}$

  j := j + 1;

  $\{0 \le j \le N \ \&\& \ \mathbf{s+a[j]} = (\Sigma i \mid 0 \le i < j \bullet a[i]) \}$ *// by assignment rule*

  s := s + a[j];

  $\{0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \}$

Proof Obligations

- Invariant is maintained

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}

  {0 ≤ **j +1** ≤ N && s+a[**j+1**] = (Σi | 0≤i<**j+1** • a[i]) } *// by assignment rule*

  j := j + 1;

  {0 ≤ j ≤ N && **s+a[j]** = (Σi | 0≤i<j • a[i]) } *// by assignment rule*

  s := s + a[j];

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }

Proof Obligations

- Invariant is maintained

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}

  {0 ≤ **j +1** ≤ N && s+a[**j+1**] = (Σi | 0≤i<**j+1** • a[i]) } *// by assignment rule*

  j := j + 1;

  {0 ≤ j ≤ N && **s+a[j]** = (Σi | 0≤i<j • a[i]) } *// by assignment rule*

  s := s + a[j];

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }
- Need to show that:

  (0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N)

  ⇒ (0 ≤ j +1 ≤ N && s+a[j+1] = (Σi | 0≤i<j+1 • a[i]))

Proof Obligations

- Invariant is maintained

  $\{0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \ \&\& \ j < N\}$

  $\{0 \leq \mathbf{j+1} \leq N \ \&\& \ s + a[\mathbf{j+1}] = (\Sigma i \mid 0 \leq i < \mathbf{j+1} \bullet a[i])\}$ *// by assignment rule*

  j := j + 1;

  $\{0 \leq j \leq N \ \&\& \ \mathbf{s+a[j]} = (\Sigma i \mid 0 \leq i < j \bullet a[i])\}$ *// by assignment rule*

  s := s + a[j];

  $\{0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i])\}$

- Need to show that:

  $(0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \ \&\& \ j < N)$

  $\qquad \Rightarrow (0 \leq j+1 \leq N \ \&\& \ s + a[j+1] = (\Sigma i \mid 0 \leq i < j+1 \bullet a[i]))$

$=$ $(0 \leq j < N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]))$

  $\qquad \Rightarrow (\mathbf{-1 \leq j < N} \ \&\& \ s + a[j+1] = (\Sigma i \mid 0 \leq i < j+1 \bullet a[i]))$ *// simplify bounds of j*

Proof Obligations

- Invariant is maintained

  {$0 \le j \le N$ && $s = (\Sigma i \mid 0 \le i < j \cdot a[i])$ && $j < N$}

  {$0 \le \mathbf{j+1} \le N$ && $s+a[\mathbf{j+1}] = (\Sigma i \mid 0 \le i < \mathbf{j+1} \cdot a[i])$ } *// by assignment rule*

  j := j + 1;

  {$0 \le j \le N$ && $\mathbf{s+a[j]} = (\Sigma i \mid 0 \le i < j \cdot a[i])$ } *// by assignment rule*

  s := s + a[j];

  {$0 \le j \le N$ && $s = (\Sigma i \mid 0 \le i < j \cdot a[i])$ }
- Need to show that:

  ($0 \le j \le N$ && $s = (\Sigma i \mid 0 \le i < j \cdot a[i])$ && $j < N$)

  $\Rightarrow$ ($0 \le j+1 \le N$ && $s+a[j+1] = (\Sigma i \mid 0 \le i < j+1 \cdot a[i])$)

= ($0 \le j < N$ && $s = (\Sigma i \mid 0 \le i < j \cdot a[i])$)

  $\Rightarrow$ ($\mathbf{-1 \le j < N}$ && $s+a[j+1] = (\Sigma i \mid 0 \le i < j+1 \cdot a[i])$) *// simplify bounds of j*

= ($0 \le j < N$ && $s = (\Sigma i \mid 0 \le i < j \cdot a[i])$)

  $\Rightarrow$ ($-1 \le j < N$ && $s+a[j+1] = (\Sigma i \mid 0 \le i < j \cdot a[i])$ $\mathbf{+ a[j]}$) *// separate last element*

Proof Obligations

- Invariant is maintained
  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}
  {0 ≤ **j +1** ≤ N && s+a[**j+1**] = (Σi | 0≤i<**j+1** • a[i]) } *// by assignment rule*
  j := j + 1;
  {0 ≤ j ≤ N && **s+a[j]** = (Σi | 0≤i<j • a[i]) } *// by assignment rule*
  s := s + a[j];
  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }
- Need to show that:
  (0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N)
  ⇒ (0 ≤ j +1 ≤ N && s+a[j+1] = (Σi | 0≤i<j+1 • a[i]))

= (0 ≤ j < N && s = (Σi | 0≤i<j • a[i]))
  ⇒ (**-1 ≤ j < N** && s+a[j+1] = (Σi | 0≤i<j+1 • a[i])) *// simplify bounds of j*

= (0 ≤ j < N && s = (Σi | 0≤i<j • a[i]))
  ⇒ (-1 ≤ j < N && s+a[j+1] = (Σi | 0≤i<j • a[i]) **+ a[j]**) *// separate last element*

// we have a problem – we need a[j+1] and a[j] to cancel out

Where's the error?

- Prove array sum correct

```
{ N ≥ 0 }
j := 0;
s := 0;

while (j < N) do

   j := j + 1;
   s := s + a[j];

end
{ s = (Σi | 0≤i<N • a[i]) }
```

Where's the error?

- Prove array sum correct

$\{ N \geq 0 \}$

```
j := 0;
s := 0;

while (j < N) do

   j := j + 1;
   s := s + a[j];

end
```

$\{ s = (\Sigma i \mid 0 \leq i < N \bullet a[i]) \}$

Need to add element
before incrementing j

Corrected Code

- Prove array sum correct

```
{ N ≥ 0 }
j := 0;
s := 0;

while (j < N) do

   s := s + a[j];
   j := j + 1;

end
{ s = (Σi | 0≤i<N • a[i]) }
```

Proof Obligations

- Invariant is maintained
  $\{0 \leq j \leq N \ \&\&\ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \ \&\&\ j < N\}$

  s := s + a[j];

  j := j + 1;
  $\{0 \leq j \leq N \ \&\&\ s = (\Sigma i \mid 0 \leq i < j \bullet a[i])\}$

Proof Obligations

- Invariant is maintained

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}

  s := s + a[j];

  {0 ≤ **j +1** ≤ N && s = (Σi | 0≤i<**j+1** • a[i]) } *// by assignment rule*

  j := j + 1;

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }

Proof Obligations

- Invariant is maintained

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}
  {0 ≤ j +1 ≤ N && **s+a[j]** = (Σi | 0≤i<j+1 • a[i]) }   *// by assignment rule*
  s := s + a[j];
  {0 ≤ **j +1** ≤ N && s = (Σi | 0≤i<**j+1** • a[i]) }   *// by assignment rule*
  j := j + 1;
  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }

Proof Obligations

- Invariant is maintained

  {$0 \le j \le N$ && $s = (\Sigma i \mid 0 \le i < j \bullet a[i])$ && $j < N$}

  {$0 \le j + 1 \le N$ && $\mathbf{s+a[j]} = (\Sigma i \mid 0 \le i < j+1 \bullet a[i])$ } *// by assignment rule*

  s := s + a[j];

  {$0 \le \mathbf{j+1} \le N$ && $s = (\Sigma i \mid 0 \le i < \mathbf{j+1} \bullet a[i])$ } *// by assignment rule*

  j := j + 1;

  {$0 \le j \le N$ && $s = (\Sigma i \mid 0 \le i < j \bullet a[i])$ }
- Need to show that:

  ($0 \le j \le N$ && $s = (\Sigma i \mid 0 \le i < j \bullet a[i])$ && $j < N$)

  $\Rightarrow$ ($0 \le j + 1 \le N$ && $s+a[j] = (\Sigma i \mid 0 \le i < j+1 \bullet a[i])$)

Proof Obligations

- Invariant is maintained

  $\{0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \ \&\& \ j < N\}$

  $\{0 \le j + 1 \le N \ \&\& \ \mathbf{s+a[j]} = (\Sigma i \mid 0 \le i < j+1 \bullet a[i]) \}$ *// by assignment rule*

  s := s + a[j];

  $\{0 \le \mathbf{j + 1} \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < \mathbf{j+1} \bullet a[i]) \}$ *// by assignment rule*

  j := j + 1;

  $\{0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \}$

- Need to show that:

  $(0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \ \&\& \ j < N)$

  $\Rightarrow (0 \le j + 1 \le N \ \&\& \ s+a[j] = (\Sigma i \mid 0 \le i < j+1 \bullet a[i]))$

$=$ $(0 \le j < N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]))$

  $\Rightarrow (\mathbf{-1 \le j < N} \ \&\& \ s+a[j] = (\Sigma i \mid 0 \le i < j+1 \bullet a[i]))$ *// simplify bounds of j*

Proof Obligations

- Invariant is maintained

  $\{0 \leq j \leq N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \;\&\&\; j < N\}$

  $\{0 \leq j+1 \leq N \;\&\&\; \mathbf{s+a[j]} = (\Sigma i \mid 0 \leq i < j+1 \bullet a[i])\}$ *// by assignment rule*

  s := s + a[j];

  $\{0 \leq \mathbf{j+1} \leq N \;\&\&\; s = (\Sigma i \mid 0 \leq i < \mathbf{j+1} \bullet a[i])\}$ *// by assignment rule*

  j := j + 1;

  $\{0 \leq j \leq N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i])\}$

- Need to show that:

  $(0 \leq j \leq N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \;\&\&\; j < N)$

  $\Rightarrow (0 \leq j+1 \leq N \;\&\&\; s+a[j] = (\Sigma i \mid 0 \leq i < j+1 \bullet a[i]))$

$=$ $(0 \leq j < N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i]))$

  $\Rightarrow (\mathbf{-1 \leq j < N} \;\&\&\; s+a[j] = (\Sigma i \mid 0 \leq i < j+1 \bullet a[i]))$ *// simplify bounds of j*

$=$ $(0 \leq j < N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i]))$

  $\Rightarrow (-1 \leq j < N \;\&\&\; s+a[j] = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \mathbf{+ a[j]})$ *// separate last part of sum*

Proof Obligations

- Invariant is maintained

  $\{0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \ \&\& \ j < N\}$

  $\{0 \le j+1 \le N \ \&\& \ \mathbf{s+a[j]} = (\Sigma i \mid 0 \le i < j+1 \bullet a[i])\}$ *// by assignment rule*

  s := s + a[j];

  $\{0 \le \mathbf{j+1} \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < \mathbf{j+1} \bullet a[i])\}$ *// by assignment rule*

  j := j + 1;

  $\{0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i])\}$

- Need to show that:

$$(0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \ \&\& \ j < N)$$

$$\Rightarrow (0 \le j+1 \le N \ \&\& \ s+a[j] = (\Sigma i \mid 0 \le i < j+1 \bullet a[i]))$$

$$= (0 \le j < N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]))$$

$$\Rightarrow (\mathbf{-1 \le j < N} \ \&\& \ s+a[j] = (\Sigma i \mid 0 \le i < j+1 \bullet a[i]))$$ *// simplify bounds of j*

$$= (0 \le j < N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]))$$

$$\Rightarrow (-1 \le j < N \ \&\& \ s+a[j] = (\Sigma i \mid 0 \le i < j \bullet a[i]) + \mathbf{a[j]})$$ *// separate last part of sum*

$$= (0 \le j < N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]))$$

$$\Rightarrow (-1 \le j < N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]))$$ *// subtract a[j] from both sides*

Proof Obligations

- Invariant is maintained

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}

  {0 ≤ j +1 ≤ N && **s+a[j]** = (Σi | 0≤i<j+1 • a[i]) } *// by assignment rule*

  s := s + a[j];

  {0 ≤ **j +1** ≤ N && s = (Σi | 0≤i<**j+1** • a[i]) } *// by assignment rule*

  j := j + 1;

  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }

- Need to show that:

  (0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N)

  ⇒ (0 ≤ j +1 ≤ N && s+a[j] = (Σi | 0≤i<j+1 • a[i]))

= (0 ≤ j < N && s = (Σi | 0≤i<j • a[i]))

  ⇒ (**-1 ≤ j < N** && s+a[j] = (Σi | 0≤i<j+1 • a[i])) *// simplify bounds of j*

= (0 ≤ j < N && s = (Σi | 0≤i<j • a[i]))

  ⇒ (-1 ≤ j < N && s+a[j] = (Σi | 0≤i<j • a[i]) **+ a[j]**) *// separate last part of sum*

= (0 ≤ j < N && s = (Σi | 0≤i<j • a[i]))

  ⇒ (-1 ≤ j < N && s = (Σi | 0≤i<j • a[i])) *// subtract a[j] from both sides*

= **true** *// 0 ≤ j ⇒ -1 ≤ j*

Proof Obligations

- Invariant and exit condition implies postcondition

$0 \le j \le N$ && $s = (\Sigma i \mid 0 \le i < j \bullet a[i])$ && $j \ge N$

$\Rightarrow s = (\Sigma i \mid 0 \le i < N \bullet a[i])$

Proof Obligations

- Invariant and exit condition implies postcondition

$0 \le j \le N \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \ \&\& \ j \ge N$

$\Rightarrow s = (\Sigma i \mid 0 \le i < N \bullet a[i])$

$= 0 \le j \ \&\& \ \mathbf{j = N} \ \&\& \ s = (\Sigma i \mid 0 \le i < j \bullet a[i])$

$\Rightarrow s = (\Sigma i \mid 0 \le i < N \bullet a[i])$

// because $(j \le N \ \&\& \ j \ge N) = (j = N)$

Proof Obligations

- Invariant and exit condition implies postcondition

$$0 \le j \le N \;\&\&\; s = (\Sigma i \mid 0 \le i < j \bullet a[i]) \;\&\&\; j \ge N$$

$$\Rightarrow s = (\Sigma i \mid 0 \le i < N \bullet a[i])$$

$$= 0 \le j \;\&\&\; \mathbf{j = N} \;\&\&\; s = (\Sigma i \mid 0 \le i < j \bullet a[i])$$

$$\Rightarrow s = (\Sigma i \mid 0 \le i < N \bullet a[i])$$

// because (j ≤ N && j ≥ N) = (j = N)

$$= \; 0 \le \mathbf{N} \;\&\&\; s = (\Sigma i \mid 0 \le i < \mathbf{N} \bullet a[i]) \Rightarrow s = (\Sigma i \mid 0 \le i < N \bullet a[i])$$

// by substituting N for j, since j = N

Proof Obligations

- Invariant and exit condition implies postcondition

$0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \ \&\& \ j \geq N$

$\quad\quad \Rightarrow s = (\Sigma i \mid 0 \leq i < N \bullet a[i])$

$= 0 \leq j \ \&\& \ \mathbf{j = N} \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i])$

$\quad\quad \Rightarrow s = (\Sigma i \mid 0 \leq i < N \bullet a[i])$

// because $(j \leq N \ \&\& \ j \geq N) = (j = N)$

$= 0 \leq \mathbf{N} \ \&\& \ s = (\Sigma i \mid 0 \leq i < \mathbf{N} \bullet a[i]) \Rightarrow s = (\Sigma i \mid 0 \leq i < N \bullet a[i])$

// by substituting N for j, since j = N

= **true** *// because P && Q ⇒ Q*

Practice: Writing Proof Obligations

- For the program below and the invariant i <= N, write the proof obligations. The form of your answer should be three mathematical implications.

```
{ N >= 0 }

i := 0;

while (i < N) do

   i := N

{ i = N }
```

- Invariant is initially true:
- Invariant is preserved by the loop body:
- Invariant and exit condition imply postcondition:

Invariant Intuition

- For code without loops, we are simulating execution directly
  - We prove one Hoare Triple for each statement, and each statement is executed once
- For code with loops, we are doing *one* proof of correctness for *multiple* loop iterations
  - Proof must cover all iterations
    - Don’t know how many there will be
  - The invariant must be *general* yet *precise*
    - general enough to be true for every execution
    - precise enough to imply the postcondition we need
  - This tension makes inferring loop invariants challenging

Can we also formalize proof obligations?

Yes, with ***verification condition generation*** $P \Rightarrow VCGen(S, Q)$

- Bonus: we can get one formula for correctness of the whole program
- Rather than segmenting into several formulas that we prove individually

$$VCGen(\text{skip}, Q) =$$
$$VCGen(S_1; S_2, Q) =$$
$$VCGen(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) =$$
$$VCGen(x := e, Q) =$$

Can we also formalize proof obligations?

Yes, with ***verification condition generation*** $P \Rightarrow VCGen(S, Q)$

- Bonus: we can get one formula for correctness of the whole program
- Rather than segmenting into several formulas that we prove individually

$$
\begin{aligned}
VCGen(\text{skip}, Q) &= Q \\
VCGen(S_1; S_2, Q) &= VCGen(S_1, VCGen(S_2, Q)) \\
VCGen(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) &= b \Rightarrow VCGen(S_1, Q) \wedge \neg b \Rightarrow VCGen(S_2, Q) \\
VCGen(x := e, Q) &= [e/x]Q
\end{aligned}
$$

- Loops are special—as usual!

$$VCGen(\text{while}_{inv}\ e \text{ do } S, Q) = Inv \wedge (\forall x_1 ... x_n . Inv \Rightarrow (e \Rightarrow VCGen(S, Inv) \wedge \neg e \Rightarrow Q))$$

Verification Condition Generation - Summary & Future Lectures

- **Verification Conditions** make axiomatic semantics practical.
  - We can solve them automatically with SAT solvers
  - We can compute verification conditions **forward** for use on unstructured code (= assembly language). This is sometimes called **symbolic execution**.
- We can add extra invariants or drop paths (dropping is *unsound*) to help verification condition generation **scale**.
- We can model **exceptions**, **memory** operations and **data structures** using verification condition generation.