Proofs using \textsc{While} Semantics

(minor corrections from class to incorporate strengthened induction hypothesis)

\textbf{Theorem:} \([\{y \mapsto 1, x \mapsto n\}, \text{while } x > 1 \text{ do } y := y \times x; x := x-1] \mapsto^* [\{y \mapsto n!, x \mapsto 1\}, \text{skip}]\)

\textbf{Proof:} By induction on \(n\). Strengthened induction hypothesis:
\([\{y \mapsto m, x \mapsto n\}, \text{while } x > 1 \text{ do } y := y \times x; x := x-1] \mapsto^* [\{y \mapsto m \times n!, x \mapsto 1\}, \text{skip}]\)

\textbf{Base case (n=1):}
\([\{y \mapsto m, x \mapsto 1\}, \text{while } x > 1 \text{ do } y := y \times x; x := x-1] \ni \ni (\{y \mapsto m \times 1!, x \mapsto 1\}, \text{skip})\)

\textbf{Inductive case (assume induction hypothesis for n-1):}
\([\{y \mapsto m, x \mapsto n\}, \text{while } x > 1 \text{ do } y := y \times x; x := x-1] \ni \ni (\{y \mapsto m \times n, x \mapsto n-1\}, \text{while } x > 1 \text{ do } y := y \times x; x := x-1)\)
\ni \ni (\{y \mapsto m \times n, x \mapsto n-1\}, \text{while } x > 1 \text{ do } y := y \times x; x := x-1)\)
\ni \ni (\{y \mapsto m \times n \times (n-1)!\}, x \mapsto 1\}, \text{skip}) // \textit{using induction hypothesis}\)
\ni \ni (\{y \mapsto m \times n!, x \mapsto 1\}, \text{skip}) // \textit{arithmetic simplification}\)
How would you argue that this program is correct?

float sum(float *array, int length) {
    float sum = 0.0;
    int i = 0;
    while (i < length) {
        sum = sum + array[i];
        i = i + 1;
    }
    return sum;
}

Function Specifications

• Predicate: a boolean function over program state
  • $x=3$
  • $y > x$
  • $(x \neq 0) \Rightarrow (y+z = w)$
  • $s = \sum_{i=1..n} a[i]$
  • $\forall i \in 1..n . a[i] > a[i-1]$
  • $true$
Function Specifications

• Contract between client and implementation
  • Precondition:
    • A predicate describing the condition the function relies on for correct operation
  • Postcondition:
    • A predicate describing the condition the function establishes after correctly running
  • Correctness with respect to the specification
    • If the client of a function fulfills the function’s precondition, the function will execute to completion and when it terminates, the postcondition will be true
  • What does the implementation have to fulfill if the client violates the precondition?
    • A: Nothing. It can do anything at all.

/*@ requires len >= 0 && array.length = len @
@ ensures \result == <(sum int j; 0 <= j && j < len; array[j])>
@*/
float sum(int array[], int len) {
    float sum = 0.0;
    int i = 0;
    while (i < length) {
        sum = sum + array[i];
        i = i + 1;
    }
    return sum;
}
Hoare Triples

- Formal reasoning about program correctness using pre- and postconditions
- Syntax: \{ P \} S \{ Q \}
  - P and Q are predicates
  - S is a program
- If we start in a state where P is true and execute S, S will terminate in a state where Q is true

Hoare Triple Examples

- \{ true \} x := 5 \{ x=5 \}
- \{ x = y \} x := x + 3 \{ x = y + 3 \}
- \{ x > 0 \} x := x * 2 \{ x > -2 \}
- \{ x=a \} if (x < 0) then x := -x \{ x=|a| \}
- \{ false \} x := 3 \{ x = 8 \}
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: \(x = 10 \Rightarrow \text{true}\)
    - 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\}
    - 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

• \{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
• 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

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: \( \{ [E/x] P \} x := E \{ P \} \)
  - \( [3 / x] (x + y > 0) \)
  - \( = (3) + y > 0 \)
  - \( = y > -3 \)

Hoare Logic Rules

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

- Assignment rule
  - \( wp(x := E, P) = [E/x] P \)
  - \( [3*y+z / x] (x * y - z > 0) \)
  - \( = (3*y+z) * y - z > 0 \)
  - \( = 3*y^2 + z*y - z > 0 \)
Correctness of Assignment

- Use language semantics to show soundness of rule
  - General soundness condition for \{P\} S \{Q\}
    \[(\eta \vdash P \downarrow \text{true} \land (\eta, S) \rightarrow^* (\eta', \text{skip})) \Rightarrow \eta' \vdash Q \downarrow \text{true}\]

- Specialization to assignment
  - Hoare rule: \{ [a/x] P \} x := a \{ P \}
  - Soundness condition:
    \[(\eta \vdash [a/x] P \downarrow \text{true} \land (\eta, \ x:=a) \rightarrow (\eta', \skip)) \Rightarrow \eta' \vdash P \downarrow \text{true}\]

Correctness Proof

- To show:
  \[(\eta \vdash [a/x] P \downarrow \text{true} \land (\eta, \ x:=a) \rightarrow (\eta', \skip)) \Rightarrow \eta' \vdash P \downarrow \text{true}\]

- Prove more general property:
  - Use assignment evaluation rule:
    \[\eta \vdash a \downarrow v \quad \Rightarrow \quad (\eta, \ x:=a) \rightarrow (\eta[x\mapsto v], \text{skip})\]

- Substitute \(v'\) for \text{true}:
  \[(\eta \vdash [a/x] P \downarrow v' \land \eta \vdash a \downarrow v) \Rightarrow \eta[x\mapsto v] \vdash P \downarrow v'\]
Correctness Proof

- \((\eta \vdash [a/x] P \downarrow v' \land \eta \vdash a \downarrow v) \Rightarrow \eta[x \mapsto v] \vdash P \downarrow v'\)
- Proof by induction on structure of \(P\)
  - case \(n\): then \(v' = n\), and using big-step semantics we get
    \((\eta \vdash [a/x] n \downarrow n \land \eta \vdash a \downarrow v) \Rightarrow \eta[x \mapsto v] \vdash n \downarrow n\)
  - case \(x\): then \(v' = v\), and using big-step semantics we get
    \((\eta \vdash [a/x] x \downarrow v \land \eta \vdash a \downarrow v) \Rightarrow \eta[x \mapsto v] \vdash x \downarrow v\)
  - case \(y \neq x\): then \(v' = \eta(y)\), and using big-step semantics we get
    \((\eta \vdash [a/x] y \downarrow \eta(y) \land \eta \vdash a \downarrow v) \Rightarrow \eta[x \mapsto v] \vdash y \downarrow \eta(y)\)
  - case \(a' \text{ op } a''\):
    - We use the induction hypotheses to get
      \((\eta \vdash [a/x] a' \downarrow v' \land \eta \vdash a \downarrow v) \Rightarrow \eta[x \mapsto v] \vdash a' \downarrow v'\)
    - And similar for \(a''\), so that using big-step semantics we get
      \((\eta \vdash [a/x] (a' \text{ op } a'') \downarrow (v' \text{ op } v'') \land \eta \vdash a \downarrow v) \Rightarrow \eta[x \mapsto v] \vdash (a' \text{ op } a'') \downarrow (v' \text{ op } v'')\)
    - other cases are similar

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 \( P \)?

- **Conditional rule**
  - \( wp(\text{if } B \text{ then } S \text{ else } T, Q) \)
    - \( B \Rightarrow wp(S, Q) \) \&\& \( \neg B \Rightarrow wp(T, Q) \)
  - \( wp(\text{if } x > 0 \text{ then } y := z \text{ else } y := -z, y > 5) \)
  - \( = x > 0 \Rightarrow wp(y := z, y > 5) \) \&\& \( x \leq 0 \Rightarrow wp(y := -z, y > 5) \)
  - \( = x > 0 \Rightarrow z > 5 \) \&\& \( x \leq 0 \Rightarrow -z > 5 \)
  - \( = x > 0 \Rightarrow z > 5 \) \&\& \( x \leq 0 \Rightarrow z < -5 \)

---

Hoare Logic Rules

- **Loops**
  - \( \{ P \} \) while \( (i < x) \) \( f = f^i; i := i + 1 \) \{ f = x! \}
  - What is the weakest precondition \( P \)?
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 \neg B) \Rightarrow Q
    • The invariant and the loop exit condition imply the postcondition
  • *Why do we need each condition?*

---

Loop Example

• 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 \cdot a[i]) \}
Loop Example

- Prove array sum correct

\begin{align*}
\{ & N \geq 0 \} \\
& j := 0; \\
& s := 0; \\
& \{ 0 \leq j \leq N \&\& s = (\Sigma i \mid 0\leq i<j \cdot a[i]) \} \\
\text{while} \ (j < N) \ \text{do} \\
& \{0 \leq j \leq N \&\& s = (\Sigma i \mid 0\leq i<j \cdot a[i]) \&\& j < N\} \\
& \quad j := j + 1; \\
& \quad s := s + a[j]; \\
& \{0 \leq j \leq N \&\& s = (\Sigma i \mid 0\leq i<j \cdot a[i]) \} \\
\text{end} \\
& \{ s = (\Sigma i \mid 0\leq i<N \cdot a[i]) \}
\end{align*}

Proof Obligations

- Invariant is initially true

\begin{align*}
\{ & N \geq 0 \} \\
& j := 0; \\
& s := 0; \\
& \{ 0 \leq j \leq N \&\& s = (\Sigma i \mid 0\leq i<j \cdot a[i]) \} \\
\end{align*}

- Invariant is maintained

\begin{align*}
& \{0 \leq j \leq N \&\& s = (\Sigma i \mid 0\leq i<j \cdot a[i]) \&\& j < N\} \\
& \quad j := j + 1; \\
& \quad s := s + a[j]; \\
& \{0 \leq j \leq N \&\& s = (\Sigma i \mid 0\leq i<j \cdot a[i]) \} \\
\end{align*}

- Invariant and exit condition implies postcondition

\begin{align*}
0 \leq j \leq N \&\& s = (\Sigma i \mid 0\leq i<j \cdot a[i]) \&\& j \geq N \\
\Rightarrow s = (\Sigma i \mid 0\leq i<N \cdot a[i])
\end{align*}
Proof Obligations

- Invariant is initially true
  \[
  \{ N \geq 0 \} \land \{ 0 \leq 0 \leq N \land 0 = (\sum_{i} | 0 \leq i < 0 \cdot a[i]) \} \land \{ j := 0 \};
  \{ 0 \leq j \leq N \land 0 = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \land \{ s := 0 \};
  \{ 0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \]

- Need to show that:
  \[
  (N \geq 0) \implies (0 \leq 0 \leq N \land 0 = (\sum_{i} | 0 \leq i < 0 \cdot a[i]))
  = (N \geq 0) \implies (0 \leq 0 \leq N \land 0 = 0) \quad \text{// 0 \leq 0 is true, empty sum is 0}
  = (N \geq 0) \implies (0 \leq N) \quad \text{// 0=0 is true, P \land true is P}
  = \text{true}
  \]

---

Proof Obligations

- Invariant is maintained
  \[
  \{ 0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \land j < N \}
  \{ 0 \leq j + 1 \leq N \land s+a[j+1] = (\sum_{i} | 0 \leq i < j + 1 \cdot a[i]) \} \land \{ j := j + 1; \}
  \{ 0 \leq j \leq N \land s+a[j] = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \land \{ s := s + a[j]; \}
  \{ 0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \]

- Need to show that:
  \[
  (0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \land j < N)
  \implies (0 \leq j + 1 \leq N \land s+a[j+1] = (\sum_{i} | 0 \leq i < j + 1 \cdot a[i]) \land \{ j := j + 1; \}
  \{ 0 \leq j \leq N \land s+a[j] = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \land \{ s := s + a[j]; \}
  \{ 0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \}
  = (0 \leq j < N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \land j < N)
  \implies (0 \leq j + 1 \leq N \land s+a[j+1] = (\sum_{i} | 0 \leq i < j + 1 \cdot a[i]) \land \{ j := j + 1; \}
  \{ 0 \leq j \leq N \land s+a[j] = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \land \{ s := s + a[j]; \}
  \{ 0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \}
  = (-1 \leq j < N \land s+a[j+1] = (\sum_{i} | 0 \leq i < j \cdot a[i]) + a[j] \land \{ j := j + 1; \}
  \{ 0 \leq j \leq N \land s+a[j] = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \land \{ s := s + a[j]; \}
  \{ 0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \}
  = (-1 \leq j < N \land s+a[j+1] = (\sum_{i} | 0 \leq i < j \cdot a[i]) + a[j] \land \{ j := j + 1; \}
  \{ 0 \leq j \leq N \land s+a[j] = (\sum_{i} | 0 \leq i < j \cdot a[i]) \} \land \{ s := s + a[j]; \}
  \{ 0 \leq j \leq N \land s = (\sum_{i} | 0 \leq i < j \cdot a[i]) \}
  = \text{we have a problem – we need a[j+1] and a[j] to cancel out}
  \]

---
Where’s the error?

- Prove array sum correct

\[
\begin{align*}
\{ N \geq 0 \} & \\
j & := 0; \\
s & := 0; \\
while \ (j < N) \ do & \\
j & := j + 1; \\
s & := s + a[j]; \\
end & \\
\{ s = (\Sigma i \mid 0 \leq i < N \cdot a[i]) \}
\end{align*}
\]

Corrected Code

- Prove array sum correct

\[
\begin{align*}
\{ N \geq 0 \} & \\
j & := 0; \\
s & := 0; \\
while \ (j < N) \ do & \\
s & := s + a[j]; \\
j & := j + 1; \\
end & \\
\{ s = (\Sigma i \mid 0 \leq i < N \cdot a[i]) \}
\end{align*}
\]
Proof Obligations

- Invariant is maintained
  \[0 \leq j \leq N \land s = (\sum_i | 0 \leq i < j \land a[i]) \land j < N\]
  \[0 \leq j + 1 \leq N \land s + a[j] = (\sum_i | 0 \leq i < j + 1 \land a[i])\] // by assignment rule
  \[s := s + a[i];\]
  \[0 \leq j + 1 \leq N \land s = (\sum_i | 0 \leq i < j + 1 \land a[i])\] // by assignment rule
  \[j := j + 1;\]
  \[0 \leq j \leq N \land s = (\sum_i | 0 \leq i < j \land a[i])\]

- Need to show that:
  \[0 \leq j \leq N \land s = (\sum_i | 0 \leq i < j \land a[i]) \land j < N\]
  \[\Rightarrow (0 \leq j + 1 \leq N \land s + a[j] = (\sum_i | 0 \leq i < j + 1 \land a[i]))\]
  \[= (0 \leq j < N \land s = (\sum_i | 0 \leq i < j \land a[i]))\]
  \[\Rightarrow (-1 \leq j < N \land s + a[j] = (\sum_i | 0 \leq i < j \land a[i]))\] // simplify bounds of j
  \[= (0 \leq j < N \land s = (\sum_i | 0 \leq i < j \land a[i]))\]
  \[\Rightarrow (-1 \leq j < N \land s + a[j] = (\sum_i | 0 \leq i < j \land a[i]) + a[j])\] // separate last part of sum
  \[= (0 \leq j < N \land s = (\sum_i | 0 \leq i < j \land a[i]))\]
  \[\Rightarrow (-1 \leq j < N \land s = (\sum_i | 0 \leq i < j \land a[i]))\] // subtract a[j] from both sides
  \[= \text{true}\] // 0 \leq j \Rightarrow -1 \leq j

Proof Obligations

- Invariant and exit condition implies postcondition
  \[0 \leq j \leq N \land s = (\sum_i | 0 \leq i < j \land a[i]) \land j \geq N\]
  \[\Rightarrow s = (\sum_i | 0 \leq i < N \land a[i])\]
  \[= 0 \leq j \land j = N \land s = (\sum_i | 0 \leq i < j \land a[i])\]
  \[\Rightarrow s = (\sum_i | 0 \leq i < N \land a[i])\] // because \(j \leq N \land j \geq N\) = \(j = N\)
  \[= 0 \leq N \land s = (\sum_i | 0 \leq i < N \land a[i]) \Rightarrow s = (\sum_i | 0 \leq i < N \land a[i])\] // by substituting \(N\) for \(j\), since \(j = N\)
  \[= \text{true}\] // because \(P \land Q \Rightarrow Q\)
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
  - Don’t know how many iterations there will be
  - Need our proof to cover all of them
  - The invariant expresses a general condition that is true for every execution, but is still strong enough to give us the postcondition we need
  - This tension between generality and precision can make coming up with loop invariants hard

Total Correctness for Loops

- \{P\} while B do S \{Q\}
- Partial correctness:
  - Find an invariant Inv such that:
    - P \implies Inv
      - The invariant is initially true
    - (Inv \&\& B) S (Inv)
      - Each execution of the loop preserves the invariant
    - (Inv \&\& \neg B) \implies Q
      - The invariant and the loop exit condition imply the postcondition
- Total correctness
  - Loop will terminate
  - How to show this?
Total Correctness for Loops

- \{P\} while B do S \{Q\}
- **Partial correctness:**
  - Find an invariant Inv such that:
    - \( P \Rightarrow Inv \)
      - The invariant is initially true
    - \( \{Inv \&\& B\} \Rightarrow Inv\)
      - Each execution of the loop preserves the invariant
    - \( (Inv \&\& \neg B) \Rightarrow Q\)
      - The invariant and the loop exit condition imply the postcondition
- **Termination bound**
  - Find a variant function \( v \) such that:
    - \( (Inv \&\& B) \Rightarrow v > 0\)
      - The variant function evaluates to a finite integer value greater than zero at the beginning of the loop
    - \( \{Inv \&\& B \&\& v = v\} \Rightarrow v < V\)
      - The value of the variant function decreases each time the loop body executes (here \( V \) is a constant)

Total Correctness Example

while \((j < N)\) do
  \{0 \leq j \leq N \&\& s = (\sum i \mid 0 \leq i < j \cdot a[i]) \&\& j < N\}
  
  \( s := s + a[j]; \)
  \( j := j + 1; \)
  \{0 \leq j \leq N \&\& s = (\sum i \mid 0 \leq i < j \cdot a[i]) \}
end

- Variant function for this loop?
  - \( N-j \)
Guessing Variant Functions

- Loops with an index
  - $N \pm i$
  - Applies if you always add or always subtract a constant, and if you exit the loop when the index reaches some constant
  - Use $N-i$ if you are incrementing $i$, $N+i$ if you are decrementing $i$
  - Set $N$ such that $N \pm i \leq 0$ at loop exit

- Other loops
  - Find an expression that is an upper bound on the number of iterations left in the loop

Additional Proof Obligations

- Variant function for this loop: $N-j$
- To show: variant function initially positive
  \[
  0 \leq j \leq N \land \land s = (\Sigma i \mid 0 \leq i < j \land a[i]) \land j < N
  \Rightarrow N-j > 0
  \]
- To show: variant function is decreasing
  \[
  \{0 \leq j \leq N \land \land s = (\Sigma i \mid 0 \leq i < j \land a[i]) \land j < N \land \land N-j = V\}
  s := s + a[j];
  j := j + 1;
  \{N-j < V\}
Additional Proof Obligations

- To show: variant function initially positive
  \[(0 \leq j \leq N \land s = (\Sigma_{i=0}^{i<j} a[i]) \land j < N) \Rightarrow N-j > 0\]
  \[= (0 \leq j \leq N \land s = (\Sigma_{i=0}^{i<j} a[i]) \land j < N) \Rightarrow N > j \quad \text{// added } j \text{ to both sides}\]
  \[= \text{true} \quad \text{// } (N > j) = (j < N), \quad P \land Q \Rightarrow P\]

Additional Proof Obligations

- To show: variant function is decreasing
  \[\{0 \leq j \leq N \land s = (\Sigma_{i=0}^{i<j} a[i]) \land j < N \land N-j = V\}\]
  \[\{N-(j+1) < V\} \quad \text{// by assignment}\]
  \[s := s + a[j];\]
  \[\{N-(j+1) < V\} \quad \text{// by assignment}\]
  \[j := j + 1;\]
  \[\{N-j < V\}\]
- Need to show:
  \[(0 \leq j \leq N \land s = (\Sigma_{i=0}^{i<j} a[i]) \land j < N \land N-j \neq V) \Rightarrow (N-(j+1) < V)\]
  Assume \(0 \leq j \leq N \land s = (\Sigma_{i=0}^{i<j} a[i]) \land j < N \land N-j = V\)
  By weakening we have \(N-j = V\)
  Therefore \(N-j-1 < V\)
  But this is equivalent to \(N-(j+1) < V\), so we are done.
Factorial

\{ N \geq 1 \}

k := 1
f := 1

while (k < N) do
  f := f \times k
  k := k + 1
end

\{ f = N! \}

• Loop invariant?

• Variant function?

---

Analysis of Software Artifacts -
Spring 2006

39

Factorial

\{ N \geq 1 \}

k := 1
f := 1

while (k < N) do
  k := k + 1
  f := f \times k
end

\{ f = N! \}

• Loop invariant?
  • f = k! && 0 \leq k \leq N
• Variant function?
  • N-k

---

Analysis of Software Artifacts -
Spring 2006

40

Need to increment k

before multiplying
Factorial

\[
\begin{align*}
\{ N \geq 1 \} \\
\{ 1 = 1! \land \& \ 0 \leq 1 \leq N \} \\
k \leftarrow 1 \\
\{ 1 = k! \land \& \ 0 \leq k \leq N \} \\
f \leftarrow 1 \\
\{ f = k! \land \& \ 0 \leq k \leq N \} \\
\text{while} (k < N) \text{ do} \\
\{ f = k! \land \& \ 0 \leq k \leq N \land \& \ k < N \land \& \ N-k = V \} \\
\{ f^*(k+1) = (k+1)! \land \& \ 0 \leq k+1 \leq N \land \& \ N-(k+1) < V \} \\
k \leftarrow k + 1 \\
\{ f^*k = k! \land \& \ 0 \leq k \leq N \land \& \ N-k < V \} \\
f \leftarrow f^*k \\
\{ f = k! \land \& \ 0 \leq k \leq N \land \& \ N-k < V \} \\
\text{end} \\
\{ f = k! \land \& \ 0 \leq k \leq N \land \& \ k \geq N \} \\
\{ f = N! \} \\
\end{align*}
\]

Factorial Obligations (1)

\[
\begin{align*}
(N \geq 1) \Rightarrow (1 = 1! \land \& \ 0 \leq 1 \leq N) \\
= (N \geq 1) \Rightarrow (1 \leq N) \quad \text{// because } 1 = 1! \text{ and } 0 \leq 1 \\
= \text{true} \quad \text{// because } (N \geq 1) = (1 \leq N)
\end{align*}
\]
Factorial Obligations (2)

(f = k! & 0 ≤ k ≤ N & k < N & N-k = V) 
⇒ (f*(k+1) = (k+1)! & 0 ≤ k+1 ≤ N & N-(k+1) < V) 
= (f = k! & 0 ≤ k < N & N-k = V) 
⇒ (f*(k+1) = k!*N & 0 ≤ k+1 ≤ N & N-k-1 < V) 
// by simplification and (k+1)! = k!*N

Assume (f = k! & 0 ≤ k < N & N-k = V)

Check each RHS clause:

- (f*(k+1) = k!*N)
  = (f = k!) // division by (k+1) (nonzero by assumption)
  = true // by assumption
- 0 ≤ k+1
  = true // by assumption that 0 ≤ k
- k+1 ≤ N
  = true // by assumption that k < N
- N-k-1 < V
  = N-k+1 < N-k // by assumption that N-k = V
  = N-1 < V // by addition of k
  = true // by properties of <

Factorial Obligations (3)

(f = k! & 0 ≤ k ≤ N & k ≥ N) ⇒ (f = N!)

Assume f = k! & 0 ≤ k ≤ N & k ≥ N

Then k=N by k ≤ N & k ≥ N

So f = N! by substituting k=N