Hoare Logic:
Proving Programs Correct

17-654/17-765

Analysis of Software Artifacts

Jonathan Aldrich

Reading: C.A.R. Hoare, An Axiomatic Basis for Computer Programming

Some presentation ideas from a lecture by K. Rustan M. Leino

Proofs using WHILE Semantics

(minor corrections from class to incorporate strengthened induction hypothesis)

Theorem: ([y↦1, x↦n], while x > 1 do y := y*x; x := x-1)
↦* ([y↦n!, x↦1], skip)

Proof: By induction on n. Strengthened induction hypothesis:
([y↦**m**, x↦n], while x > 1 do y := y*x; x := x-1)
↦* ([y↦**m*n!**, x↦1], skip)

Base case (n=1):
([y↦m, x↦1], while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*1!, x↦1], skip)

Inductive case (assume induction hypothesis for n-1):
([y↦m, x↦n], while x > 1 do y := y*x; x := x-1)
↦ ([y↦m, x↦n], y := y*x; x := x-1; while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*n, x↦n], x := x-1; while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*n, x↦n-1], while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*n*(n-1)!, x↦1], skip) *// using induction hypothesis*
↦ ([y↦m*n!, x↦1], skip) *// 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
  - $x=3$
  - $y > x$
  - $(x \neq 0) \Rightarrow (y+z = w)$
  - $s = \Sigma_{(i \in 1..n)}\ a[i]$
  - $\forall i \in 1..n\ .\ a[i] > a[i-1]$
  - 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.

Function Specifications

```
/*@ 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 ⇒ Q', then Q is the strongest postcondition of S with respect to P
  - check: x = 10 ⇒ true
  - check: x = 10 ⇒ x > 0
  - check: x = 10 ⇒ x = 10 || x = 5
  - check: x = 10 ⇒ 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 ≠ 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 ≠ 0 && x / y < 1 is the *weakest precondition*
- If {P} S {Q} and for all P' such that {P'} S {Q}, P' ⇒ 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}

$$(\eta \vdash P \downarrow true \;\wedge\; (\eta, S) \mapsto^* (\eta', skip)) \;\Rightarrow\; \eta' \vdash Q \downarrow true$$

- Specialization to assignment
  - Hoare rule: { [a/x] P } x := a { P }
  - Soundness condition:

$$(\eta \vdash [a/x]P \downarrow true \;\wedge\; (\eta,\; x{:=}a) \mapsto (\eta',\; skip)) \;\Rightarrow\; \eta' \vdash P \downarrow true$$

Correctness Proof

- To show:

$$(\eta \vdash [a/x]P \downarrow true \;\wedge\; (\eta,\; x{:=}a) \mapsto (\eta',\; skip)) \;\Rightarrow\; \eta' \vdash P \downarrow true$$

- Prove more general property:
  - Use assignment evaluation rule:

$$\frac{\eta \vdash a \downarrow v}{(\eta,\; x{:=}a) \;\mapsto\; (\eta[x{\mapsto}v],\; skip)}$$

  - Substitute *v′* for *true*:

$$(\eta \vdash [a/x]P \downarrow v' \;\wedge\; \eta \vdash a \downarrow v) \;\Rightarrow\; \eta[x{\mapsto}v] \vdash P \downarrow v'$$

Correctness Proof

- $(\eta \vdash [a/x]P \downarrow v' \ \wedge \ \eta \vdash a \downarrow v) \ \Rightarrow \ \eta[x \mapsto v] \vdash P \downarrow v'$
- Proof by induction on structure of P
  - case n: then $v' = n$, and using big-step semantics we get
    $(\eta \vdash [a/x]n \downarrow n \ \wedge \ \eta \vdash a \downarrow v) \ \Rightarrow \ \eta[x \mapsto v] \vdash n \downarrow n$
  - case x: then $v' = v$, and using big-step semantics we get
    $(\eta \vdash [a/x]x \downarrow v \ \wedge \ \eta \vdash a \downarrow v) \ \Rightarrow \ \eta[x \mapsto v] \vdash x \downarrow v$
  - case $y \neq x$: then $v' = \eta(y)$, and using big-step semantics we get
    $(\eta \vdash [a/x]y \downarrow \eta(y) \wedge \eta \vdash a \downarrow v) \Rightarrow \eta[x \mapsto v] \vdash y \downarrow \eta(y)$
  - case $a'\ op\ a''$:
    - We use the induction hypotheses to get
      $(\eta \vdash [a/x]a' \downarrow v' \ \wedge \ \eta \vdash a \downarrow v) \ \Rightarrow \ \eta[x \mapsto v] \vdash a' \downarrow v'$
    - And similar for a'', so that using big-step semantics we get
      $(\eta \vdash [a/x](a'\ op\ a'') \downarrow (v'\ \mathbf{op}\ v'') \ \wedge \ \eta \vdash a \downarrow v)$
      $\Rightarrow \eta[x \mapsto v] \vdash (a'\ op\ a'') \downarrow (v'\ \mathbf{op}\ v'')$
  - 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*(if B then S else T, Q)
    = B ⇒ *wp*(S,Q) && ¬B ⇒ *wp*(T,Q)
  - *wp*(if x>0 then y:=z else y:=-z, y>5)
  - = x>0 ⇒ *wp*(y:=z,y>5) && x≤0 ⇒ *wp*(y:=-z,y>5)
  - = x>0 ⇒ z > 5 && x≤0 ⇒ -z > 5
  - = x>0 ⇒ z > 5 && x≤0 ⇒ 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
    - $(Inv \ \&\& \ \neg B) \Rightarrow Q$
      - The invariant and the loop exit condition imply the postcondition
    - ***Why do we need each condition?***

Loop Example

- Prove array sum correct

```
{ N ≥ 0 }
j := 0;
s := 0;

while (j < N) do

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

end
{ s = (Σi | 0≤i<N • a[i]) }
```

Loop Example

- Prove array sum correct

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

Proof Obligations

- Invariant is initially true
  $\{ N \geq 0 \}$
  $j := 0;$
  $s := 0;$
  $\{ 0 \leq j \leq N \ \&\& \ s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \}$
- 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 implies 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
  $\{ N \geq 0 \}$
  $\{ 0 \leq \mathbf{0} \leq N \;\&\&\; 0 = (\Sigma i \mid 0 \leq i < \mathbf{0} \bullet a[i]) \}$ *// 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
  $\{0 \leq j \leq N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \;\&\&\; j < N\}$
  $\{0 \leq \mathbf{j+1} \leq N \;\&\&\; s+a[\mathbf{j+1}] = (\Sigma i \mid 0 \leq i < \mathbf{j+1} \bullet a[i]) \}$ *// by assignment rule*
  $j := j + 1;$
  $\{0 \leq j \leq N \;\&\&\; \mathbf{s+a[j]} = (\Sigma i \mid 0 \leq i < j+1 \bullet a[i]) \}$ *// 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:
  $(0 \leq j \leq N \;\&\&\; s = (\Sigma i \mid 0 \leq i < j \bullet a[i]) \;\&\&\; j < N)$
  $\quad \Rightarrow (0 \leq j+1 \leq N \;\&\&\; s+a[j+1] = (\Sigma i \mid 0 \leq i < j+1 \bullet a[i]))$

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

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

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

= 0 ≤ j && **j = N** && s = (Σi | 0≤i<j • a[i])
  ⇒ s = (Σi | 0≤i<N • a[i])
  // because (j ≤ N && j ≥ N) = (j = N)

= 0 ≤ **N** && s = (Σi | 0≤i<**N** • a[i]) ⇒ s = (Σi | 0≤i<N • a[i])
  // by substituting N for j, since j = N

= **true** *// because P && Q ⇒ 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:
    - $P \Rightarrow Inv$
      - 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
- 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 } 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
- 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 } S {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 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N}
    s := s + a[j];
    j := j + 1;
    {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) }
end
```

- Variant function for this loop?
  - N-j

Guessing Variant Functions

- Loops with an index
  - N ± 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 ± i ≤ 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 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N)

  ⇒ N-j > 0
- To show: variant function is decreasing

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

  s := s + a[j];

  j := j + 1;

  {N-j < V}

Additional Proof Obligations

- To show: variant function initially positive
  (0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N)
  ⇒ N-j > 0

= (0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N)
  ⇒ **N > j** *// added j to both sides*

= **true** *// (N > j) = (j < N), P && Q ⇒ P*

Additional Proof Obligations

- To show: variant function is decreasing
  {0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N && N-j = V}
  {N-(j+1) < V} *// by assignment*
  s := s + a[j];
  {N-(**j+1**) < V} *// by assignment*
  j := j + 1;
  {N-j < V}
- Need to show:
  (0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N && N-j = V)
  ⇒ (N-(j+1) < V)

Assume 0 ≤ j ≤ N && s = (Σi | 0≤i<j • a[i]) && j < N && 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 ≥ 1 }
k := 1
f := 1
while (k < N) do
    f := f * k
    k := k + 1
end
{ f = N! }

- Loop invariant?
- Variant function?

Factorial

{ N ≥ 1 }
k := 1
f := 1
while (k < N) do
    k := k + 1
    f := f * k
end
{ f = N! }

Need to increment k ***before*** multiplying

- Loop invariant?
  - f = k! && 0 ≤ k ≤ N
- Variant function?
  - N-k

Factorial

{ $N \geq 1$ }
{ $1 = 1! \;\&\&\; 0 \leq 1 \leq N$ }
k := 1
{ $1 = k! \;\&\&\; 0 \leq k \leq N$ }
f := 1
{ $f = k! \;\&\&\; 0 \leq k \leq N$ }
while (k < N) do
    { $f = k! \;\&\&\; 0 \leq k \leq N \;\&\&\; k < N \;\&\&\; N\text{-}k = V$ }
    { $f^*(k+1) = (k+1)! \;\&\&\; 0 \leq k+1 \leq N \;\&\&\; N\text{-}(k+1) < V$ }
    k := k + 1
    { $f^*k = k! \;\&\&\; 0 \leq k \leq N \;\&\&\; N\text{-}k < V$ }
    f := f * k
    { $f = k! \;\&\&\; 0 \leq k \leq N \;\&\&\; N\text{-}k < V$ }
end
{ $f = k! \;\&\&\; 0 \leq k \leq N \;\&\&\; k \geq N$ }
{ $f = N!$ }

Factorial Obligations (1)

$(N \geq 1) \Rightarrow (1 = 1! \;\&\&\; 0 \leq 1 \leq N)$
= $(N \geq 1) \Rightarrow (1 \leq N)$ *// because $1 = 1!$ and $0 \leq 1$*
= true   *// because* $(N \geq 1) = (1 \leq N)$

Factorial Obligations (2)

$(f = k! \ \&\& \ 0 \leq k \leq N \ \&\& \ k < N \ \&\& \ N\text{-}k = V)$
$\Rightarrow (f^*(k+1) = (k+1)! \ \&\& \ 0 \leq k+1 \leq N \ \&\& \ N\text{-}(k+1) < V)$

= $(f = k! \ \&\& \ 0 \leq k < N \ \&\& \ N\text{-}k = V)$
$\Rightarrow (f^*(k+1) = k!^*(k+1) \ \&\& \ 0 \leq k+1 \leq N \ \&\& \ N\text{-}k\text{-}1 < V)$
// by simplification and (k+1)!=k!(k+1)*

Assume $(f = k! \ \&\& \ 0 \leq k < N \ \&\& \ N\text{-}k = V)$

Check each RHS clause:

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

Factorial Obligations (3)

$(f = k! \ \&\& \ 0 \leq k \leq N \ \&\& \ k \geq N) \Rightarrow (f = N!)$

Assume $f = k! \ \&\& \ 0 \leq k \leq N \ \&\& \ k \geq N$

Then k=N by $k \leq N \ \&\& \ k \geq N$

So f = N! by substituting k=N