Inferring invariants

- There is no fully general automation for inferring invariants (as there is for the weakest precondition of assignment statements).

- This is one of the things that makes totally automated verification difficult.

- Finding the right invariant is still a human intellectual activity.
Approaching Invariants through the “Back Door”

- Suppose A is an assertion that we desire to be true after a while loop.
- Suppose the invariant I is unknown.
- We do know that whatever I is, it satisfies:

\[(I \land \neg P) \rightarrow A\]

by comparing the template:

\[
\{I\} \quad \text{while } P \text{ do } B \quad \{I \land \neg P \}
\]

\[
\{I\} \quad \text{while } P \text{ do } B \quad \{A\}
\]
Approximating I

• For starters, then, we could “over–approximate” I by A.

• Then working backward through the body B, we derive the “weakest pre-condition” corresponding to A, denoted:
  \[ wp_B(A) \]

• We also know that
  \[(I \land \neg P) \rightarrow wp_B(A)\]
  so we can use \( wp_B(A) \) as another approximation to I.

• Continuing, I is approximated by
  \[ A \lor wp_B(A) \lor wp_B(wp_B(A)) \lor wp_B(wp_B(wp_B(A))) \lor ... \]
Example

- \( i := 0; \)
  while \( i < n \)
  do
    \( i := i + 1 \)
  od
{\( i = n \)}

\( \wp_i := i + 1 (i = n) \) is \( i + 1 = n \) which is the same as \( i = n - 1 \).
\( \wp_i := i + 1 (\wp_i := i + 1 (i = n)) \) is \( i + 1 = n - 1 \) which is \( i = n - 2 \)

So the continued approximation is:
\( (i = n) \lor (i = n - 1) \lor (i = n - 2) \lor ... \)

suggesting as the invariant \( i \leq n \), which we know works.

Moreover, for \( n \geq 0, i \leq n \) is implied by \( i = 0 \), which is also required.
Proof Using JAPE

WHERE DISTINCT i, n IS
{0 ≤ n}

(i := 0)

{i ≤ n}

while i < n
  do i := i + 1 od

{i = n}
Proof Steps

\[ \ldots \]
\[ 1: \{0 \leq n\}(i:=0)\{i \leq n\}\text{while }i<n\text{ do }i:=i+1\text{ od}\{i=n\} \]

nTuple rule

\[ \ldots \]
\[ 1: \{0 \leq n\}(i:=0)\{i \leq n\} \]
\[ \ldots \]
\[ 2: \{i \leq n\}\text{while }i<n\text{ do }i:=i+1\text{ od}\{i=n\} \]
\[ 3: \{0 \leq n\}(i:=0)\{i \leq n\}\text{while }i<n\text{ do }i:=i+1\text{ od}\{i=n\} \text{ Ntuple 1,2} \]

Proof of Initialization

\[ 1: \{0 \leq n\}(i:=0)\{i \leq n\} \]
\[ \ldots \]
\[ 2: \{i \leq n\}\text{while }i<n\text{ do }i:=i+1\text{ od}\{i=n\} \]
\[ 3: \{0 \leq n\}(i:=0)\{i \leq n\}\text{while }i<n\text{ do }i:=i+1\text{ od}\{i=n\} \text{ Ntuple 1,2} \]
Expand the while

\[ \ldots \]

\[ 2: \{ i \leq n \land i < n \} (i := i + 1) \{ i \leq n \} \]

\[ \ldots \]

\[ 3: i \leq n \land i < n \rightarrow _M > 0 \]

\[
\begin{array}{l}
4: \text{integer } K_m \\
\ldots \\
5: \{ i \leq n \land i < n \land _M = K_m \} (i := i + 1) \{ _M < K_m \}
\end{array}
\]

\[ \{ i \leq n \} \text{ while } i < n \text{ do } i := i + 1 \text{ od} \{ i \leq n \land \neg (i < n) \} \]

\[ \quad \text{ while } 2, 3, 4, 5 \]

\[ \ldots \]

\[ 7: i \leq n \land \neg (i < n) \rightarrow i = n \]

\[ 8: \{ i \leq n \} \text{ while } i < n \text{ do } i := i + 1 \text{ od} \{ i = n \} \]

\[ \text{ consequence(R) 6,7 } \]
Body Partial Correctness

2: \(i \leq n \land i < n\)
3: \(i < n\)
4: \(i + 1 \leq n\)

5: \(i \leq n \land i < n \rightarrow i + 1 \leq n\)
6: \(\{i + 1 \leq n\}(i_0 = i + 1)\{i \leq n\}\)

assumption
\(\land\) elim 2
\(A + 1 \leq B \Rightarrow A < B\) 3
\(\rightarrow\) intro 2–4
variable-assignment

body
Set the Variant: n-i

8: $i \leq n \land i < n \rightarrow n-i > 0$

9: integer $K_m$

... body ...

10: $\{i \leq n \land i < n \land (n-i=K_m) (i:=i+1) \land n-i < K_m\}$

11: $\{i \leq n\} \text{while } i < n \text{ do } i := i+1 \text{ od} \{i \leq n \land \neg(i < n)\}$

assumption

while 7,8,9–10
Prove the Variant Continuation Condition

8: \( i \leq n \land i < n \)
9: \( i < n \)
10: \( n - i > 0 \)
11: \( i \leq n \land i < n \rightarrow n - i > 0 \)

assumption
\( \land \) elim 8
\( i < n \rightarrow n - i > 0 \) 9
\( \rightarrow \) intro 8–10
Prove Body Decreases Variant

12: integer Km
13: \[i \leq n \land i < n \land n - i = Km\]
14: \[n - i = Km\]
15: \[n - (i + 1) < Km\]
16: \[i \leq n \land i < n \land n - i = Km \rightarrow n - (i + 1) < Km\]
17: \[\{n - (i + 1) < Km\}(i := i + 1)\{n - i < Km\}\]
18: \[\{i \leq n \land i < n \land n - i = Km\}(i := i + 1)\{n - i < Km\}\]
19: \[\{i \leq n\} while i < n \text{ do } i := i + 1 \text{ od } \{i \leq n \land \neg (i < n)\}\]

assumption
\land elim 13
n - i = X \rightarrow n - (i + 1) < X 14
→ intro 13-15
variable-assignment
consequence(L) 16,17
while 7,11,12-18
Loop exit implies final expectation

\[
\begin{align*}
20: & \quad i \leq n \land \neg (i < n) \\
21: & \quad i \leq n \\
22: & \quad \neg (i < n) \\
23: & \quad i = n \\
24: & \quad i \leq n \land \neg (i < n) \rightarrow i = n
\end{align*}
\]

assumption
\[
\begin{align*}
\land \text{ elim } 20 \\
\land \text{ elim } 20 \\
i \leq n, \neg (i < n) \vdash i = n \quad 21,22 \\
\rightarrow \text{ intro } 20-23
\end{align*}
\]
Can you identify all the parts?
1: \{0 \leq n \} \{ i := 0 \} \{ i \leq n \}
2: \{ \leq n \} \{ i \leq n \}
3: \{ i \leq n \}
4: \{ i + 1 \leq n \}
5: \{ i \leq n \} \{ i + 1 \leq n \}
6: \{ i + 1 \leq n \} \{ i := i + 1 \} \{ i \leq n \}
7: \{ \leq n \} \{ i \leq n \} \{ i := i + 1 \} \{ i \leq n \}
8: \{ i \leq n \}
9: \{ i < n \}
10: \{ n - i > 0 \}
11: \{ i \leq n \} \{ i < n \} \{ n - i > 0 \}

12: \text{integer } K_m
13: \{ i \leq n \} \{ n - i = K_m \}
14: \{ n - i = K_m \}
15: \{ n - (i + 1) < K_m \}
16: \{ i \leq n \} \{ n - (i + 1) < K_m \}
17: \{ n - (i + 1) = K_m \} \{ i := i + 1 \} \{ n - i < K_m \}
18: \{ i \leq n \} \{ i < n \} \{ n - i = K_m \} \{ i := i + 1 \} \{ n - i < K_m \}

19: \{ \leq n \} \{ i < n \} \{ i := i + 1 \} \{ \text{while triple} \}
20: \{ \leq n \} \{ i < n \}
21: \{ i = n \}
22: \{ -(i < n) \}
23: \{ i = n \}
24: \{ i \leq n \} \{ -(i < n) \} \{ i = n \}

25: \{ \leq n \} \{ i < n \} \{ i := i + 1 \} \{ \text{while triple} \}
26: \{ 0 \leq n \} \{ i := 0 \} \{ i \leq n \} \{ \text{while triple + consequent} \} \{ nTuple 1, 25 \}
Proof as Trees

Partial Correctness Tree

assign

pure logic

1: \{0 \leq n\}(i := 0)\{i \leq n\}

24: i \leq n \land \neg (i < n) \rightarrow i = n

assign

7: \{i \leq n \land i < n\}(i := i + 1)\{i \leq n\}

19: \{i \leq n\}\text{while } i < n \text{ do } i := i + 1 \text{ od}\{i \leq n \land \neg (i < n)\}

25: \{i \leq n\}\text{while } i < n \text{ do } i := i + 1 \text{ od}\{i = n\}

26: \{0 \leq n\}(i := 0)\{i \leq n\}\text{while } i < n \text{ do } i := i + 1 \text{ od}\{i = n\}

Termination Tree

pure logic

11: i \leq n \land i < n \rightarrow n - i > 0

pure logic

16: i \leq n \land i < n \land n - i = K_m \rightarrow n - (i + 1) < K_m

assign

17: \{n - (i + 1) < K_m\}(i := i + 1)\{n - i < K_m\}

18: \{i \leq n \land i < n \land n - i = K_m\}(i := i + 1)\{n - i < K_m\}

19: \{i \leq n\}\text{while } i < n \text{ do } i := i + 1 \text{ od}\{i \leq n \land \neg (i < n)\}
Subtleties About Loop Invariants

• Can the following Ntuple be proved?

\{0 \leq n\}

(j := 1; i := 0; y := 0)

\{y = i \times i \land 0 \leq i \land i \leq n\} \quad \text{Is the invariant truly invariant?}

while i < n

do

\begin{align*}
y & := y + j; \\
j & := j + 2; \\
i & := i + 1
\end{align*}

do

\{y = n \times n\}
Loop Body Correctness Proof Reduces to

\begin{align*}
11: \quad & y = i \times i \land 0 \leq i \leq n \land i < n \\
& \quad \ldots \\
12: \quad & y + j = (i + 1) \times (i + 1) \land 0 \leq i + 1 \land i + 1 \leq n
\end{align*}

This cannot be proved. j is not even mentioned in the hypothesis.

What went wrong?
Important: Strength of Invariants

- Our invariant was “too weak” to enable its own inductive proof.

- In making an invariant stronger, we get more information from which to derive the post condition.

- At the same time, however, we have more to prove with a stronger invariant, so we don’t want to go too far.
The Lattice of Invariants

\[ \top \]
very weak, implies nothing but itself,
but implied by everything

\[ \uparrow \]
I'
other invariants, implied by I*, but
too weak in themselves to enable
inductive proof

\[ \uparrow \]
I*
an **inductive** invariant, “just right”,
but not necessarily unique

\[ \uparrow \] implies

\[ \bot \]
very strong, implies everything,
but implied by nothing but itself
An Inductive Invariant

\{0 \leq n\}

(j := 1; i := 0; y := 0)

\{y=i\times i \land j = 2\times i \land 0 \leq i \land i \leq n\}

while i < n
  do
    y := y+j;
    j := j+2;
    i := i+1
  od

\{y=n\times n\}
Now the Body Proof Works

\begin{align*}
&14: \ y = i \times i, \ j = 2 \times i + 1 \land 0 \leq i \land i \leq n \land i < n \\
&15: \ y = i \times i \\
&16: \ j = 2 \times i + 1 \\
&17: \ 0 \leq i \\
&18: \ i < n \\
&19: \ y + j = (i + 1) \times (i + 1) \\
&20: \ j + 2 = 2 \times (i + 1) + 1 \\
&21: \ 0 \leq i + 1 \\
&22: \ i + 1 \leq n \\
&23: \ y + j = (i + 1) \times (i + 1) \land j + 2 = 2 \times (i + 1) + 1 \land 0 \leq i + 1 \land i + 1 \leq n
\end{align*}
The Completed Proof (lines 1-24)
(From a different run, so there may be minor differences.)

<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(y=0)</td>
</tr>
<tr>
<td>2</td>
<td>(i=0)</td>
</tr>
<tr>
<td>3</td>
<td>(n \geq 0)</td>
</tr>
<tr>
<td>4</td>
<td>(y = i \times i)</td>
</tr>
<tr>
<td>5</td>
<td>(i \leq n)</td>
</tr>
<tr>
<td>6</td>
<td>(i \geq 0)</td>
</tr>
<tr>
<td>7</td>
<td>(1 = 2 \times i + 1)</td>
</tr>
<tr>
<td>8</td>
<td>(y = i \times i \leq n \land i \geq 0 \land 1 = 2 \times i + 1)</td>
</tr>
<tr>
<td>9</td>
<td>(y = 0 \land i = 0 \land n \geq 0)</td>
</tr>
<tr>
<td>10</td>
<td>(y = i \times i \leq n \land i \geq 0 \land 1 = 2 \times i + 1)</td>
</tr>
<tr>
<td>11</td>
<td>(y = i \times i \leq n \land i \geq 0 \land 1 = 2 \times i + 1)</td>
</tr>
<tr>
<td>12</td>
<td>(y = 0 \land i = 0 \land n \geq 0)</td>
</tr>
<tr>
<td>13</td>
<td>(y = i \times i \leq n \land i \geq 0 \land j = 2 \times i + 1)</td>
</tr>
<tr>
<td>14</td>
<td>(y = i \times i)</td>
</tr>
<tr>
<td>15</td>
<td>(i \geq 0)</td>
</tr>
<tr>
<td>16</td>
<td>(j = 2 \times i + 1)</td>
</tr>
<tr>
<td>17</td>
<td>(i &lt; n)</td>
</tr>
<tr>
<td>18</td>
<td>(y+j = (i+1) \times (i+1))</td>
</tr>
<tr>
<td>19</td>
<td>(i+1 \leq n)</td>
</tr>
<tr>
<td>20</td>
<td>(i+1 \geq 0)</td>
</tr>
<tr>
<td>21</td>
<td>(j+2 = 2 \times (i+1) + 1)</td>
</tr>
<tr>
<td>22</td>
<td>(y+j = (i+1) \times (i+1) \land i+1 \leq n \land i+1 \geq 0 \land j=2 \times (i+1) + 1)</td>
</tr>
<tr>
<td>23</td>
<td>(y = i \times i \leq n \land i \geq 0 \land j = 2 \times i + 1)</td>
</tr>
<tr>
<td>24</td>
<td>(y = j \times (i+1) \leq n \land i+1 \geq 0 \land j+2 = 2 \times (i+1) + 1)</td>
</tr>
<tr>
<td></td>
<td>({y = j \times (i+1) \leq n \land i+1 \geq 0 \land j+2 = 2 \times (i+1) + 1} \land \text{variable-assignment})</td>
</tr>
</tbody>
</table>

Initialization

First assignment in loop body
The Completed Proof (lines 25-51)

25. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \leq n \)  
26. \( y = (i + 1) x (i + 1) a i + 1 \leq n \)  
27. \( y = (i + 1) x (i + 1) a i + 1 \leq n \)  
28. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 \)  

29. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \leq n \)  
30. \( i < n \)  
31. \( n - i > 0 \)  
32. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \leq n \)  

33. integer \( K_m \)  
34. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \)  
35. \( n - i = K_m \)  
36. \( n = (i + 1) < K_m \)  
37. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \)  
38. \( n = (i + 1) < K_m \)  
39. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \)  
40. \( n = (i + 1) < K_m \)  
41. \( n = (i + 1) < K_m \)  
42. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \)  
43. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 \)  

44. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \)  
45. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 \)  
46. \( i < n \)  
47. \( i < n \)  
48. \( n = n \)  
49. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 a i \)  
50. \( y = i x i a i n i a i \geq 0 a j = 2 x i + 1 \)  
51. \( y = 0 a i = 0 a n \geq 0 a j = 2 x i + 1 \)  

Other assignments in loop body

Continuation condition on \( _M \)

Termination of loop body

Exit consequence
Verifying Array Programs

- Arrays present extra challenges and interesting issues.

- A useful dichotomy:
  - Programs with read-only arrays
  - Programs with modifiable arrays
Array Mathematics

- An array can be treated as if a function:
  - It maps indices into values.
  - e.g. a 1-dimensional array with dimension 10 maps \{0, ..., 9\} into values of the type stored in the array.
  - \(a[i]\) is the value of this function with argument \(i\)
Read-Only Array Example: Finding a zero element

\{n \geq 0 \land \text{length}(a) = n\}

i := 0; j := n

\{i \leq n \land i \geq 0 \land \text{length}(a) = n \land j < n \rightarrow a[j] = 0\}

while i < n do
  if a[i] = 0
    then j := i
    else skip fi;
  i := i + 1 od

\{j < n \rightarrow a[j] = 0\}
JAPE Type-in (for reference)

WHERE DISTINCT i,j,n,a IS
{0≤n∧length(a)=n}
(i := 0; j := n)
{i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)}
while i < n
   do
      if a[i] = 0 then j := i else skip fi;
      i := i+1
   od
{\{j < n \rightarrow a[j] = 0\}}
Salient Parts of Proof

**Initialization**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>$0 \leq n \land \text{length}(a) = n$</td>
</tr>
<tr>
<td>2:</td>
<td>$0 \leq n$</td>
</tr>
<tr>
<td>3:</td>
<td>$\text{length}(a) = n$</td>
</tr>
<tr>
<td>4:</td>
<td>$0 \leq 0$</td>
</tr>
<tr>
<td>5:</td>
<td>$n &lt; n$</td>
</tr>
<tr>
<td>6:</td>
<td>$\perp$</td>
</tr>
<tr>
<td>7:</td>
<td>$a[n] = 0$</td>
</tr>
<tr>
<td>8:</td>
<td>$n &lt; n \rightarrow a[n] = 0$</td>
</tr>
<tr>
<td>9:</td>
<td>$0 \leq n \land 0 \leq 0 \land \text{length}(a) = n \land (n &lt; n \rightarrow a[n] = 0)$</td>
</tr>
<tr>
<td>10:</td>
<td>$0 \leq n \land \text{length}(a) = n \rightarrow 0 \leq n \land 0 \leq 0 \land \text{length}(a) = n \land (n &lt; n \rightarrow a[n] = 0)$</td>
</tr>
<tr>
<td>11:</td>
<td>${0 \leq n \land 0 \leq 0 \land \text{length}(a) = n \land (n &lt; n \rightarrow a[n] = 0)}(i := 0)$</td>
</tr>
<tr>
<td>12:</td>
<td>${0 \leq n \land \text{length}(a) = n}(i := 0)[i \leq n \land 0 \leq i \land \text{length}(a) = n \land (n &lt; n \rightarrow a[n] = 0)]$</td>
</tr>
<tr>
<td>13:</td>
<td>${i \leq n \land 0 \leq i \land \text{length}(a) = n \land (n &lt; n \rightarrow a[n] = 0)}(j := n)[i \leq n \land 0 \leq i \land \text{length}(a) = n \land (j &lt; n \rightarrow a[j] = 0)]$</td>
</tr>
<tr>
<td>14:</td>
<td>${0 \leq n \land \text{length}(a) = n}(i := 0; j := n)[i \leq n \land 0 \leq i \land \text{length}(a) = n \land (j &lt; n \rightarrow a[j] = 0)]$</td>
</tr>
</tbody>
</table>
Loop Body Partial Correctness

15. \( \begin{align*} &i \leq n \land i \leq \text{length}(a) = n \land (j < n - a[j] = 0) \land i < n \\ &\land (a[i] = 0 \land i + 1 \leq n \land 0 \leq i + 1 \land \text{length}(a) = n \land (i < n - a[i] = 0) \\ &\land \neg (a[i] = 0 \land i + 1 \leq n \land 0 \leq i + 1 \land \text{length}(a) = n \land (j < n - a[j] = 0)) \land 0 \leq i \land i < \text{length}(a) \end{align*} \)

16. \( \begin{align*} &\text{assumption} \quad \{ \text{cut} \} \\ &\text{intro 15–16} \quad \text{variable-assignment} \quad \text{skip} \quad \text{choice 18,19} \quad \text{consequence(L) 17,20} \quad \text{variable-assignment} \quad \text{sequence 21,22} \end{align*} \)
Variant Continuation and Decrease

Without collapsing, the proof is about 74 lines.
Quantifiers

- Quantifiers are handy representing information about arrays, e.g.

- \( \forall i ((0 < i) \land (i < n)) \rightarrow a[i-1] \leq a[i] \)

- \( \exists i ((0 \leq i) \land (i < n) \land a[i] = 0) \)
Read-Only Example with Quantifiers
A program whose correct working depends on values stored in the array.

\[
\{ \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \} \quad (i := 0)
\]

\[
\{ 0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \} \text{while } a[i] \neq 0 \text{ do } i := i + 1 \text{ od} \{a[i] = 0\}
\]

- The **assumption** is that the array contains a **zero element**.

- The **expectation** is to find the index of such an element.

- **Termination critically depends on this assumption.**

- The **invariant** says that the element such that \( a[x] = 0 \) is **still to be found**.
... 

\{\exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)\} (i := 0) 

\{0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0)\} 

while \ a[i] \neq 0 \ do \ i := i + 1 \ od \ \{a[i] = 0\}
Consequence for Initialization

We’ll use $\exists$ elim.

```
1: $\exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)$
   assumption
...
2: $0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)$

1: $\exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)$
   assumption
2: integer $i_1$
3: $0 \leq i_1 \land i_1 < \text{length}(a) \land a[i_1] = 0$
   assumption
...
4: $0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)$

$\exists$ elim 1,2–4
```
Done with Initialization

1. \( \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)
2. integer \( i \)
3. \( 0 \leq i \land i < \text{length}(a) \land a[i] = 0 \)
4. \( 0 \leq i \)
5. \( i < \text{length}(a) \)
6. \( 0 \leq 0 \)
7. \( 0 < \text{length}(a) \)
8. \( 0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)
9. \( 0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)
10. \( \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)
   \( \rightarrow 0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)
11. \( \{0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)\} \)
   \( (i := 0) \{0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0)\} \)
12. \( \{\exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)\} (i := 0) \)
   \( \{0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0)\} \)

assumption
assumption
assumption
\( \land \) elim 3
\( \land \) elim 3
\( A \leq A \)
\( A \leq B, B < C \vdash A < C \) 4,5
\( \land \) intro 6,7,1
\( \exists \) elim 1,2–8
\( \rightarrow \) intro 1–9

variable-assignment

consequence(L) 10,11
On to the while loop

A subtlety arises in proving this invariant.

If the loop does not stop, it means $a[i] \neq 0$.

This is used to re-establish the invariant.
Body Partial Correctness

\[\{0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0\}\]

\[(i := i + 1)\{0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0)\}\]

Use the assignment rule. This generates a consequent implication.

\[0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0\]

\[\text{assumption}\]

\[0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0)\]

\[\rightarrow \text{intro 16-17}\]

It is tempting to try backward \& Introduction now. Don’t!
Use Forward \( \land \) Elimination

\begin{align*}
16: & \quad 0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0 \\
17: & \quad 0 \leq i \\
18: & \quad i < \text{length}(a) \\
19: & \quad \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \\
20: & \quad a[i] \neq 0 \\
\ldots \\
21: & \quad 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0)
\end{align*}
Now use $\exists$ Elimination, introducing $i_2$

```
16: 0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0  
17: 0 \leq i  
18: i < \text{length}(a)  
19: \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0)  
20: a[i] \neq 0 
21: \text{integer } i_2  
22: i \leq i_2 \land i_2 < \text{length}(a) \land a[i_2] = 0 
...  
23: 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0)  
24: 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0)  
```

Note that $i_2$ cannot equal $i$. Why?
We need to show this, to get the post-condition.
To get $i_2 \neq i$

- Set up a bifurcation using $i \leq i_2$, i.e. $i < i_2 \lor i = i_2$.
- Then use $\lor$ Elimination:
Now try backward ∧ Introductions.

• What to use for $x$ in $\exists x...$?

```
26: a[i2]=0
27: i<i2
   ...
28: 0\leq i+1
   ...
29: i+1<\text{length}(a)
   ...
30: \exists x. (i+1 \leq x \wedge x < \text{length}(a) \wedge a[x]=0)
31: 0 \leq i+1 \wedge i+1 < \text{length}(a) \wedge \exists x. (i+1 \leq x \wedge x < \text{length}(a) \wedge a[x]=0)
```
i2 is the obvious choice for x
Closure of the top half

23: \(i \leq i_2\)
24: \(i < i_2 \lor i = i_2\)
25: \(i_2 < \text{length}(a)\)
26: \(a[i_2] = 0\)

\[
\begin{aligned}
27: & \ i < i_2 \\
28: & \ 0 \leq i + 1 \\
29: & \ i + 1 < \text{length}(a) \\
30: & \ i + 1 \leq i_2 \\
31: & \ i + 1 \leq i_2 \land i_2 < \text{length}(a) \land a[i_2] = 0 \\
32: & \ \exists x.(i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0) \\
33: & \ 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x.(i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0) \\
\end{aligned}
\]
Bottom Half: We want a contradiction

Introduce ⊥, then figure out how to get it.
How to pick \(_B1\)?

20: \(a[i] \neq 0\)
21: \(\text{integer } i2\)
22: \(i \leq i2 \land i2 < \text{length}(a) \land a[i2] = 0\)
23: \(i \leq i2\)
24: \(i < i2 \lor i = i2\)
25: \(i2 < \text{length}(a)\)
26: \(a[i2] = 0\)

34: \(i = i2\)
35: \(_B1\)
36: \(\neg _B1\)
37: \(\bot\)
38: \(0 \leq i+1 \land i+1 < \text{length}(a) \land \exists x. (i+1 \leq x \land x < \text{length}(a) \land a[x] = 0)\)
For _B1 use a[i] = 0

Then use a comparison rule to get the second conclusion from a[i] ≠ 0:

\[ A \neq B \iff \neg (A = B) \]
Now use equality, to get 35 from 26 and 34.
Closure

26: \( a[i2] = 0 \)

\[\begin{align*}
33: & \quad i = i2 \\
34: & \quad a[i] = 0 \\
35: & \quad \neg (a[i] = 0) \\
36: & \quad \bot \\
37: & \quad 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0) \end{align*}\]

\( \wedge \) elim 22

assumption

equality-substitution 33, 25

\( A \neq B \triangleq \neg (A = B) \) 19

\( \neg \) elim 34, 35

contra (constructive) 36
What about termination?

- What to use for \_M?
Completed proof, using several lemmas 1/3

1: \( \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)  
   assumption

2: \( \text{integer } i \)  
   assumption

3: \( 0 \leq i \land i < \text{length}(a) \land a[i] = 0 \)  
   assumption

4: \( 0 \leq i \)  
   \( \land \) elim 3

5: \( i < \text{length}(a) \)  
   \( \land \) elim 3

6: \( 0 \leq 0 \)  
   \( A \leq A \)

7: \( 0 < \text{length}(a) \)  
   \( A \leq B, B < C \vdash A < C \) 4,5

8: \( 0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)  
   \( \land \) intro 6,7,1

9: \( 0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)  
   \( \exists \) elim 1,2–8

10: \( \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \to 0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0) \)

11: \( \{0 \leq 0 \land 0 < \text{length}(a) \land \exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)\} \)

   \( i := 0 \}\{0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0)\}\)  
   variable-assignment

12: \( \{\exists x. (0 \leq x \land x < \text{length}(a) \land a[x] = 0)\} (i := 0) \{0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0)\} \)  
   consequence(L) 10,11

13: \( 0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \)  
   assumption

14: \( 0 \leq i \land i < \text{length}(a) \)  
   \( \land \) elim(L) 13

15: \( 0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \to 0 \leq i \land i < \text{length}(a) \)  
   \( \to \) intro 13–14
\[
\begin{align*}
16: & \quad 0 \leq i \land i < \text{length}(a) \land \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0 \\
17: & \quad 0 \leq i \\
18: & \quad \exists x. (i \leq x \land x < \text{length}(a) \land a[x] = 0) \\
19: & \quad a[i] \neq 0 \\
20: & \quad \text{integer } i_2 \\
21: & \quad i \leq i_2 \land i_2 < \text{length}(a) \land a[i_2] = 0 \\
22: & \quad i \leq i_2 \\
23: & \quad i < i_2 \lor i = i_2 \\
24: & \quad i_2 < \text{length}(a) \\
25: & \quad a[i_2] = 0 \\
26: & \quad i < i_2 \\
27: & \quad 0 \leq i + 1 \\
28: & \quad i + 1 < \text{length}(a) \\
29: & \quad i + 1 \leq i_2 \\
30: & \quad i + 1 \leq i_2 \land i_2 < \text{length}(a) \land a[i_2] = 0 \\
31: & \quad \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0) \\
32: & \quad 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0) \\
33: & \quad i = i_2 \\
34: & \quad a[i] = 0 \\
35: & \quad \neg (a[i] = 0) \\
36: & \quad \bot \\
37: & \quad 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0) \\
38: & \quad 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0) \\
39: & \quad 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x. (i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0)
\end{align*}
\]
\[0 \leq i \land i \leq \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0 \]
\[\rightarrow 0 \leq i + 1 \land i + 1 < \text{length}(a) \land \exists x.(i + 1 \leq x \land x < \text{length}(a) \land a[x] = 0)
\]
\[(i := i + 1)[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0)]\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0\]
\[i := i + 1[0 \leq i \land i < \text{length}(a) \land \exists x.(i < x \land x < \text{length}(a) \land a[x] = 0)]\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0 \rightarrow \text{length}(a) \rightarrow i > 0\]

\[\text{integer } Km\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0 \land \text{length}(a) \rightarrow i = Km\]
\[\text{length}(a) \rightarrow i = Km\]
\[\text{length}(a) \rightarrow (i + 1) < Km\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0 \land \text{length}(a) \rightarrow (i = Km) \]
\[- \text{length}(a) \rightarrow (i + 1) < Km\]
\[\text{length}(a) \rightarrow (i + 1) < Km\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land a[i] \neq 0 \land \text{length}(a) \rightarrow i = Km\]
\[(i := i + 1)[\text{length}(a) \rightarrow i < Km]\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land \text{while } a[i] = 0 \text{ do } i := i + 1 \text{ od } 0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land \neg \neg (a[i] \neq 0)\]
\[\neg (a[i] \neq 0)\]
\[a[i] = 0\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land \neg \neg (a[i] \neq 0) \rightarrow a[i] = 0\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land \text{while } a[i] = 0 \text{ do } i := i + 1 \text{ od } a[i] = 0\]
\[\exists x.(0 \leq x \land x < \text{length}(a) \land a[x] = 0)[i := 0]\]
\[0 \leq i \land i < \text{length}(a) \land \exists x.(i \leq x \land x < \text{length}(a) \land a[x] = 0) \land \text{while } a[i] = 0 \text{ do } i := i + 1 \text{ od } a[i] = 0\]

\[\rightarrow \text{intro } 16-39\]

\[\text{variable-assignment}\]

\[\text{assumption}\]

\[\land \text{elim } 43\]

\[i \rightarrow \text{length}(a) \rightarrow \text{length}(a)... 44\]

\[\rightarrow \text{intro } 43-45\]

\[\text{assumption}\]

\[\land \text{elim } 48\]

\[\text{length}(a) \rightarrow i = x \rightarrow \text{length}(a)... 49\]

\[\rightarrow \text{intro } 48-50\]

\[\text{variable-assignment}\]

\[\text{assumption}\]

\[\land \text{elim } 51, 52\]

\[\text{while } 15, 42, 46, 47-53\]

\[\text{assumption}\]

\[\land \text{elim } 55\]

\[A = B \equiv (A = B) 56\]

\[\rightarrow \text{intro } 55-57\]

\[\text{consequence}(\) 54, 58\]

\[\text{Ntuple } 12, 59\]
Modifiable Arrays

- Arrays are like functions.

- Assigning to an array element is like creating a new function.

- The new function differs from the old in that one element may be different from before.
JAPE Array Modification Notation

- Jape Notation: \( a \oplus i \rightarrow v \) is the array that is like \( a \) except that the value of \( a[i] \) is \( v \).

- \( (a \oplus i \rightarrow v)[i] = v \) same index, new value

- \( (a \oplus i \rightarrow v)[j] = a[j] \) if \( j \neq i \) different index, old value
Array Element Assignment

- Consider a triple
  \[
  \{??\}
  \begin{align*}
  a[i] & := E \\
  \{B\}
  \end{align*}
  \]

- If this is viewed mathematically as an array replacement:
  \[
  \{??\}
  \begin{align*}
  a & := (a \oplus i \mapsto E) \\
  \{B\}
  \end{align*}
  \]

- then it is obvious what ?? should be.
Isn’t it?

• If this is viewed mathematically as an array replacement:
  \{??\}
  a := (a ⊕ i → E)
  \{B\}
  then it is obvious what ?? should be:

  B[(a ⊕ i → E)/a]

• i.e. replace all free instances of a in B with (a ⊕ i → E)/a.

  \{B[(a ⊕ i → E)/a]\}
  a := (a ⊕ i → E)
  \{B\}
Example

- \{??\}
  \[a[1] := 99\]
  \[\{a[0] = 98 \land a[1] > 50\}\]

- \{(a ⊕ 1 \rightarrow 99)[0] = 98 \land (a ⊕ 1 \rightarrow 99)[1] > 50\}\)
  \[a := (a ⊕ 1 \rightarrow 99)\]
  \[\{a[0] = 98 \land a[1] > 50\}\]
JAPE Example

\[
\]

- \{??\}
  \[a[1] := 99\]
  \{a[0] = 98 \land a[1] > 50\}

- \{(a \oplus 1 \rightarrow 99)[0] = 98 \land (a \oplus 1 \rightarrow 99)[1] > 50\}
  a := (a \oplus 1 \rightarrow 99)
  \{a[0] = 98 \land a[1] > 50\}

1: \{(a \oplus 1 \rightarrow 99)[0] = 98 \land (a \oplus 1 \rightarrow 99)[1] > 50 \land 0 \leq 1 \land 1 < \text{length}(a)\}_{\text{array-element-assignment}}
(a[1] := 99)\{a[0] = 98 \land a[1] > 50\}
Bornat’s Array Assignment Rule in JAPE

\[
\{ \ (a[E] \text{ computes}) \land (F \text{ computes}) \land B^a_{a \oplus E \rightarrow F} \ \} \ a[E] := F \{ B \}
\]

What \( a[E] \text{ computes} \) means:

\[
\begin{align*}
\vdots & \quad \vdots \\
E \text{ computes} & \quad 0 \leq E < \text{length}(a) \\
\hline
a[E] \text{ computes} \\
\end{align*}
\]
Simplifying Array Indexing Expressions

• JAPE Provides two equality rules for simplifying indexing expressions.

• These are needed for proving just about any array program. They are in the Indexing menu:

```
FROM E = G INFER (A @ E -> F)[G] = F
FROM E ≠ G INFER (A @ E -> F)[G] = A[G]
```
Simplifying Array Indexing Expressions

- The first rule deals with the case where the index G on the “new” array is the same as the index E that was modified.

- The second deals with the case where they are not the same.

- One of these cases needs to be established to perform any simplification.
Simplification Example

• Prove

\[
\{a[0] = 98 \land a[1] = 5\} \\
a[1] := 99 \\
\{a[0] = 98 \land a[1] > 50\}
\]

Candidates for Simplification

1: \{a[0]=98\land a[1]=5\}(a[1]:=99)
2: \{(a\oplus 1\mapsto 99)[0]=98\land(a\oplus 1\mapsto 99)[1]>50\land0\leq1\land1<\text{length}(a)\}
3: \{a[0]=98\land a[1]=5\}(a[1]:=99){a[0]=98\land a[1]>50\}
Using Indexing Rule where $E \neq G$

Note that the entire indexed expression must be selected, not just the array portion.
Result

when 0 ≠ 1, \((a \oplus 1 \rightarrow 99)[0]\) is same as \(a[0]\) whatever that happens to be.
Using Indexing Rule where \( E = G \)

\[
\begin{align*}
1: & \ 1 \neq 0 \\
2: & \ a[0] = 98 \land a[1] = 5 \to a[0] = 98 \land (a \oplus 1 \to 99)[1] > 50 \land 0 \leq 1 \land 1 < \text{length}(a) \\
3: & \ a[0] = 98 \land a[1] = 5 \\
& \quad \to (a \oplus 1 \to 99)[0] = 98 \land (a \oplus 1 \to 99)[1] > 50 \land 0 \leq 1 \land 1 < \text{length}(a) \\
4: & \{(a \oplus 1 \to 99)[0] = 98 \land (a \oplus 1 \to 99)[1] > 50 \land 0 \leq 1 \land 1 < \text{length}(a)\} \\
& \quad (a[1] := 99) \{a[0] = 98 \land a[1] > 50\} \\
5: & \{a[0] = 98 \land a[1] = 5\} (a[1] := 99) \{a[0] = 98 \land a[1] > 50\}
\end{align*}
\]
where 1=1, \((a\oplus 1 \rightarrow 99)[1]\) is same as RHS of assignment.
Using Bounds Inference
Once an index is used, it is assumed to be valid.

1: $a[i] = 2$
   ...
2: $a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a)$
3: $a[i] = 2 \rightarrow a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a)$
4: $a[i] = 2 \rightarrow (a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a)$
5: $(a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a)) (a[i] := a[i] + 1)\{a[i] = 3\}$
6: $\{a[i] = 2\} (a[i] := a[i] + 1)\{a[i] = 3\}$

Select only indexed expression, not entire statement

Rule

Extras

Window

Help

$A = A$

$A = ..$

$.. = B$

obviously

boundedness from (in)equality

$\rightarrow$ intro 1-2

FROM $E = G$ INFER $(A \oplus E \rightarrow F)[G] = F$

3 array-element-assignment

consequence(L) 4,5
Result

1: \( a[i] = 2 \)  
   assumption

2: \( 0 \leq i \land i < \text{length}(a) \)  
   bounded 1

3: \( a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a) \)

4: \( a[i] = 2 \rightarrow a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a) \)  
   \( \rightarrow \) intro 1-3

5: \( a[i] = 2 \rightarrow (a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a) \)

6: \( (a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a) \)[a[i]:=a[i]+1]{a[i]=3}  
   array-element-assignment

7: \{a[i]=2\}, \{a[i]:=a[i]+1\}, \{a[i]=3\}  
   consequence(L) 5,6
Completion of Proof, using Equality (complex)

Equation

Goal selected, and expression selected

Rule
Result

1: \(a[i] = 2\)
2: \(0 \leq i \land i < \text{length}(a)\)
3: \(0 \leq i\)
4: \(i < \text{length}(a)\)
5: \(2 + 1 = 3\)
6: \(a[i] + 1 = 3\)
7: \(a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a)\)
8: \(a[i] = 2 \rightarrow a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a)\)
9: \(a[i] = 2 \rightarrow (a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a)\)
10: \(((a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a)) \rightarrow a[i] = a[i] + 1 \rightarrow a[i] = 3\)
11: \{a[i] = 2\} \{a[i] = a[i] + 1\} \{a[i] = 3\}

assumption
bounded 1
\(\land\) elim 2
\(\land\) elim 2
equality-substitution 1, 5
\(\land\) intro 6, 3, 4
\(\rightarrow\) intro 1-7
FROM E = G INFER (A \oplus (E \rightarrow F))(G) = F 8
array-element-assignment
consequence(L) 9, 10
Completed Proof

1: \( a[i] = 2 \)
2: \( 0 \leq i \land i < \text{length}(a) \)
3: \( 0 \leq i \)
4: \( i < \text{length}(a) \)
5: \( 2 + 1 = 3 \)
6: \( a[i] + 1 = 3 \)
7: \( a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a) \)
8: \( a[i] = 2 \rightarrow a[i] + 1 = 3 \land 0 \leq i \land i < \text{length}(a) \)
9: \( a[i] = 2 \rightarrow (a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a) \)
10: \{(a \oplus i \rightarrow a[i] + 1)[i] = 3 \land 0 \leq i \land i < \text{length}(a)\}(a[i] := a[i] + 1)\{a[i] = 3\}
11: \{a[i] = 2\}(a[i] := a[i] + 1)\{a[i] = 3\}

assumption
bounded 1
\land\ elim 2
\land\ elim 2
obviously
equality-substitution 1, 5
\land\ intro 6, 3, 4
\rightarrow\ intro 1-7
FROM E = G INFERENCE (A \land E \rightarrow F)[G] = F
8
array-element-assignment
consequence(L) 9, 10
Consecutive Array Modifications

• Could simplify array-modification expressions as soon as they arise, or

• Wait, and deal with nested expressions.

• The first is probably better.
Consecutive Modification Example

\{a[i] = 0\}

\[a[i] := a[i] + 1;\]

\[a[i] := a[i] + 1;\]

\{a[2] = 2\}

\[\ldots\]

1: \{a[i]=0\}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}

\[\ldots\]

1: \{a[i]=0\}(a[i]:=a[i]+1){_B2}

\[\ldots\]

2: {_B2}(a[i]:=a[i]+1){a[i]=2}

3: \{a[i]=0\}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2} sequence 1,2
Early Simplification

... 
1: \{a[i]=0\}a[i]:=a[i]+1\{a[i]=a[i]+1\}[i]=2 \land 0 \leq i \land i < \text{length}(a)\}
2: \{(a\oplus i \rightarrow a[i]+1)[i]=2 \land 0 \leq i \land i < \text{length}(a)\}(a[i]:=a[i]+1)(a[i]=2\} \text{ array-element-assignment}
3: \{a[i]=0\}(a[i]:=a[i]+1; a[i]:=a[i]+1)(a[i]=2\} \text{ sequence 1,2}

... 
1: \{a[i]=0\}a[i]:=a[i]+1\{a[i]+1=2 \land 0 \leq i \land i < \text{length}(a)\}
2: \{a[i]=0\}a[i]:=a[i]+1\{(a\oplus i \rightarrow a[i]+1)[i]=2 \land 0 \leq i \land i < \text{length}(a)\} \text{ FROM E=G INFERENCE (A\oplus E \rightarrow F)[G]=F 1}
3: \{(a\oplus i \rightarrow a[i]+1)[i]=2 \land 0 \leq i \land i < \text{length}(a)\}(a[i]:=a[i]+1)(a[i]=2\} \text{ array-element-assignment}
4: \{a[i]=0\}(a[i]:=a[i]+1; a[i]:=a[i]+1)(a[i]=2\} \text{ sequence 2,3}
Early Simplification, step 2

1: \( a[i] = 0 \rightarrow (a \oplus i \rightarrow a[i] + 1)[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \)

2: \{(a \oplus i \rightarrow a[i] + 1)[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a)\}
   \( a[i] := a[i] + 1 \)\( a[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \)

3: \{a[i] = 0\} (a[i] := a[i] + 1) \{a[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a)\}

4: \{a[i] = 0\} (a[i] := a[i] + 1) \{(a \oplus i \rightarrow a[i] + 1)[i] = 2 \land 0 \leq i \land i < \text{length}(a)\} \text{ FROM } E = G \text{ INF } (A \oplus E \rightarrow F)[G] = F \)

5: \{(a \oplus i \rightarrow a[i] + 1)[i] = 2 \land 0 \leq i \land i < \text{length}(a)\} (a[i] := a[i] + 1) \{a[i] = 2\}

6: \{a[i] = 0\} (a[i] := a[i] + 1; a[i] := a[i] + 1) \{a[i] = 2\}

---

1: \( a[i] = 0 \rightarrow a[i] + 1 + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \)

2: \( a[i] = 0 \rightarrow (a \oplus i \rightarrow a[i] + 1)[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \) \text{ FROM } E = G \text{ INF } (A \oplus E \rightarrow F)[G] = F \)

3: \{(a \oplus i \rightarrow a[i] + 1)[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a)\}
   \( a[i] := a[i] + 1 \)\( a[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \)

4: \{a[i] = 0\} (a[i] := a[i] + 1) \{a[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a)\}

5: \{a[i] = 0\} (a[i] := a[i] + 1) \{(a \oplus i \rightarrow a[i] + 1)[i] = 2 \land 0 \leq i \land i < \text{length}(a)\} \text{ FROM } E = G \text{ INF } (A \oplus E \rightarrow F)[G] = F \)

6: \{(a \oplus i \rightarrow a[i] + 1)[i] = 2 \land 0 \leq i \land i < \text{length}(a)\} (a[i] := a[i] + 1) \{a[i] = 2\}

7: \{a[i] = 0\} (a[i] := a[i] + 1; a[i] := a[i] + 1) \{a[i] = 2\}

---
Completed Proof

1: \( a[i] = 0 \)  
2: \( 0 \leq i \land i < \text{length}(a) \)  
3: \( 0 \leq i \)  
4: \( i < \text{length}(a) \)  
5: \( 0 + 1 + 1 = 2 \)  
6: \( a[i] + 1 + 1 = 2 \)  
7: \( a[i] + 1 + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \)  
8: \( a[i] = 0 \rightarrow a[i] + 1 + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \)  
9: \( a[i] = 0 \rightarrow (a \oplus i \rightarrow a[i] + 1)[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a) \)  
10: \( ((a \oplus i \rightarrow a[i] + 1)[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a)) \)  
\( (a[i] := a[i] + 1)[a[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a)] \)  
11: \( \{ a[i] = 0\}(a[i] := a[i] + 1)[a[i] + 1 = 2 \land 0 \leq i \land i < \text{length}(a)] \)  
12: \( \{ a[i] = 0\}(a[i] := a[i] + 1)[(a \oplus i \rightarrow a[i] + 1)[i] = 2 \land 0 \leq i \land i < \text{length}(a)) \)  
\( \{ a[i] := a[i] + 1\}(a[i] = 2) \)  
13: \( \{ (a \oplus i \rightarrow a[i] + 1)[i] = 2 \land 0 \leq i \land i < \text{length}(a)\}(a[i] := a[i] + 1)[a[i] = 2] \)  
14: \( \{ a[i] = 0\}(a[i] := a[i] + 1; a[i] := a[i] + 1)[a[i] = 2] \)
Deferred Simplification Alternative

- Use sequence, then array-assignment twice (from the bottom up) to get to this point:
Completed Proof, using Deferred Simplification

Provided:
DISTINCT a, i

1: a[i]=0
2: 0+1+1=2
3: a[i]+1+1=2
4: (a⊕i→a[i]+1)[i]+1=2
5: (a⊕i→a[i]+1⊕i→(a⊕i→a[i]+1)[i]+1)[i]=2
6: 0≤i\wedge i<\text{length}(a)
7: 0≤i
8: i<\text{length}(a)
9: (a⊕i→a[i]+1⊕i→(a⊕i→a[i]+1)[i]+1)[i]=2\wedge 0≤i\wedge i<\text{length}(a)
10: a[i]=0→(a⊕i→a[i]+1⊕i→(a⊕i→a[i]+1)[i]+1)[i]=2\wedge 0≤i\wedge i<\text{length}(a)\quad\rightarrow\text{intro 1–9}
11: ((a⊕i→a[i]+1⊕i→(a⊕i→a[i]+1)[i]+1)[i]=2\wedge 0≤i\wedge i<\text{length}(a))
    (a[i]:=a[i]+1)(a⊕i→a[i]+1)[i]=2\wedge 0≤i\wedge i<\text{length}(a))
12: {a[i]=0}(a[i]:=a[i]+1){(a⊕i→a[i]+1)[i]=2\wedge 0≤i\wedge i<\text{length}(a)})
13: {(a⊕i→a[i]+1)[i]=2\wedge 0≤i\wedge i<\text{length}(a))\{a[i]:=a[i]+1}{a[i]=2}
14: {a[i]=0}{a[i]:=a[i]+1}{a[i]:=a[i]+1}{a[i]=2}

assumption
obviously
equality–substitution 1, 2
FROM \text{E→G INFERENCE}\ (A\& E→F)[G]=F \quad 3
FROM \text{E→G INFERENCE}\ (A\& E→F)[G]=F \quad 4
bounded 1
\wedge \text{elim 6}
\wedge \text{elim 6}
\wedge \text{intro 5,7,8}

array–element–assignment
consequence(L) 10,11
array–element–assignment
sequence 12,13
Read-Write Example

- Suppose we wish to copy one array \( a \) into another \( b \).

- We’ll assume both arrays are the same length.

- We need to assert that they have the same elements in the same positions.
Assumption and Expectation

\[\{n=\text{length}(a) \land n=\text{length}(b) \land 0 \leq n\}\]

... array-copy program ...

\[\forall j.((0 \leq j \land j < n) \rightarrow (b[j] = a[j]))\]

assuming indices run 0, ... n-1
Proposed Program

\{n=\text{length}(a) \land n=\text{length}(b) \land 0 \leq n\}

(i := 0)

while i < n
    do
        b[i] := a[i];
        i := i + 1
    od

\{\forall j.((0 \leq j \land j < n) \rightarrow (b[j]=a[j]))\}

One would hope for a short proof, but ...
Program with Invariant (for JAPE)

WHERE DISTINCT n,a,b,i IS

{n=length(a) \land n=length(b) \land 0\leq n}

(i := 0)

{n=length(a) \land n=length(b) \land 0\leq n \land 0\leq i \land i\leq n
\land \forall j.((0\leq j \land j<i) \rightarrow (b[j]=a[j]))}

while i < n do b[i] := a[i]; i := i+1 od

{\forall j.((0\leq j \land j<n) \rightarrow (b[j]=a[j]))}
Invariant Part of Loop Body

18: \( n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i \land i \leq n \land \forall j.((0 \leq j \land j < i) \rightarrow (b[j] = a[j])) \land i < n \)

19: \( n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i + 1 \land i + 1 \leq n \land \forall j.((0 \leq j \land j < i + 1) \rightarrow (b \oplus i - a[i][j] = a[j])) \land 0 \leq i \land i < \text{length}(b) \land i < \text{length}(a) \)

20: \( n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i \land i \leq n \land \forall j.((0 \leq j \land j < i) \rightarrow (b[j] = a[j])) \land i < n \\
\rightarrow n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i + 1 \land i + 1 \leq n \land \forall j.((0 \leq j \land j < i + 1) \rightarrow (b \oplus i - a[i][j] = a[j])) \land 0 \leq i \land i < \text{length}(b) \land i < \text{length}(a) \)

21: \( \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i + 1 \land i + 1 \leq n \land \forall j.((0 \leq j \land j < i + 1) \rightarrow (b \oplus i - a[i][j] = a[j])) \land 0 \leq i \land i < \text{length}(b) \land i < \text{length}(a) \}
\{ b[i] := a[i] \} \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i + 1 \land i + 1 \leq n \land \forall j.((0 \leq j \land j < i + 1) \rightarrow (b \oplus i - a[i][j] = a[j])) \}

22: \( \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i \land i \leq n \land \forall j.((0 \leq j \land j < i) \rightarrow (b[j] = a[j])) \land i < n \} \{ b[i] := a[i] \} \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i \land i \leq n \land \forall j.((0 \leq j \land j < i) \rightarrow (b[j] = a[j])) \}

23: \( \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i + 1 \land i + 1 \leq n \land \forall j.((0 \leq j \land j < i + 1) \rightarrow (b[j] = a[j])) \}
\{ i := i + 1 \} \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i \land i \leq n \land \forall j.((0 \leq j \land j < i) \rightarrow (b[j] = a[j])) \}

24: \( \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i \land i \leq n \land \forall j.((0 \leq j \land j < i) \rightarrow (b[j] = a[j])) \land i < n \}
\{ b[i] := a[i]; i := i + 1 \} \{ n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq n \land 0 \leq i \land i \leq n \land \forall j.((0 \leq j \land j < i) \rightarrow (b[j] = a[j])) \} \)
\begin{align*}
47. & \quad n = \text{length}(a) \land n = \text{length}(b) \land 0 \leq x \leq n - 1 \Rightarrow V_i(0 \leq i < n, a_i \land b_j = a_i) \\
48. & \quad V_i(0 \leq i < n, a_i \lor b_j = a_i) = 1 \\
49. & \quad V_i(0 \leq i < n, a_i \land b_j = a_i) \\
50. & \quad V_i(0 \leq i < n, a_i \lor b_j = a_i) = \text{length}(a) \\
51. & \quad V_i(0 \leq i < n, a_i \land b_j = a_i) \\
52. & \quad V_i(0 \leq i < n, a_i \lor b_j = a_i) = \text{length}(a) \\
\end{align*}
Exercises

- Provide assumption and expectation specifications for a sorting program, then prove its total correctness.

- Work through the minimal sum-section (Huth&Ryan, Example 4.19). Note the implications for algorithmic efficiency (linear for the given method vs. $O(n^3)$ for the naïve method).