(Imperative) Program Logic Part 2

Robert Keller
March 2010

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:

$(I \wedge \neg P) \rightarrow A$

by comparing the template:

{I} **while P do B** $\{I \wedge \neg P\}$

{I} **while P do B** {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
  $(I \wedge \neg P) \rightarrow wp_B(A)$
  so we can use $wp_B(A)$ as another approximation to I.
- Continuing, I is approximated by
  $A \vee wp_B(A) \vee wp_B(wp_B(A)) \vee wp_B(wp_B(wp_B(A))) \vee \ldots$

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:
$(i = n) \vee (i = n-1) \vee (i = n-2) \vee \ldots$

suggesting as the invariant $i \le n$, which we know works.

Moreover, for $n \ge 0$, $i \le n$ 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

...

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

nTuple rule

...

1: $\{0\le n\}(i:=0)\{i\le n\}$ Initialization

...

2: $\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i=n\}$ while loop

3: $\{0\le n\}(i:=0)\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i=n\}$ Ntuple 1,2

Proof of Initialization

1:	$\{0\le n\}(i:=0)\{i\le n\}$	variable-assignment
	...	
2:	$\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i=n\}$	
3:	$\{0\le n\}(i:=0)\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i=n\}$	Ntuple 1,2

Expand the while

	...	body	
2:	$\{i\le n\wedge i<n\}(i:=i+1)\{i\le n\}$	Body partial correctness	
	...		
3:	$i\le n\wedge i<n\rightarrow \_M>0$	Variant continuation condition	
4:	integer Km		assumption
	...	body	
5:	$\{i\le n\wedge i<n\wedge \_M=Km\}(i:=i+1)\{\_M<Km\}$	Body decreases variant	
6:	$\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i\le n\wedge\neg(i<n)\}$		while 2,3,4–5
	...		
7:	$i\le n\wedge\neg(i<n)\rightarrow i=n$	loop exit implies final expectation	
8:	$\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i=n\}$		consequence(R) 6,7

Body Partial Correctness

2:	$i\le n\wedge i<n$	assumption
3:	$i<n$	$\wedge$ elim 2
4:	$i+1\le n$	$A+1\le B \triangleq A<B$ 3
5:	$i\le n\wedge i<n\rightarrow i+1\le n$	$\rightarrow$ intro 2–4
6:	$\{i+1\le n\}(i:=i+1)\{i\le n\}$	variable-assignment
	body	

Set the Variant: n-i

	...	
8:	$i\le n\wedge i<n\rightarrow \boxed{n-i}>0$	
9:	integer Km	assumption
	... body	
10:	$\{i\le n\wedge i<n\wedge \boxed{n-i}=Km\}(i:=i+1)\{\boxed{n-i}<Km\}$	
11:	$\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i\le n\wedge\neg(i<n)\}$	while 7,8,9–10

Prove the Variant Continuation Condition

8:	$i\le n\wedge i<n$	assumption
9:	$i<n$	$\wedge$ elim 8
10:	$n-i>0$	$i<n \vdash n-i>0$ 9
11:	$i\le n\wedge i<n\rightarrow n-i>0$	$\rightarrow$ intro 8–10

Prove Body Decreases Variant

12:	integer Km	assumption
13:	$i\le n\wedge i<n\wedge n-i=Km$	assumption
14:	$n-i=Km$	$\wedge$ elim 13
15:	$n-(i+1)<Km$	$n-i=X \vdash n-(i+1)<X$ 14
16:	$i\le n\wedge i<n\wedge n-i=Km\rightarrow n-(i+1)<Km$	$\rightarrow$ intro 13–15
17:	$\{n-(i+1)<Km\}(i:=i+1)\{n-i<Km\}$	variable-assignment
	body	
18:	$\{i\le n\wedge i<n\wedge \boxed{n-i=Km}\}(i:=i+1)\{\boxed{n-i<Km}\}$	consequence(L) 16,17
19:	$\{i\le n\}\text{while } i<n \text{ do } i:=i+1 \text{ od}\{i\le n\wedge\neg(i<n)\}$	while 7,11,12–18

Loop exit implies final expectation

20:	$i\le n\wedge\neg(i<n)$	assumption
21:	$i\le n$	$\wedge$ elim 20
22:	$\neg(i<n)$	$\wedge$ elim 20
23:	$i=n$	$i\le n, \neg(i<n) \vdash i=n$ 21,22
24:	$i\le n\wedge\neg(i<n)\rightarrow i=n$	$\rightarrow$ intro 20–23

Completed Proof

Can you identify all the parts?

1:	$\{0\le n\}(i:=0)\{i\le n\}$	variable-assignment
2:	$i\le n\wedge i<n$	assumption
3:	$i<n$	$\wedge$ elim 2
4:	$i+1\le n$	$A+1\le B\triangleq A<B$ 3
5:	$i\le n\wedge i<n\rightarrow i+1\le n$	$\rightarrow$ intro 2–4
6:	$\{i+1\le n\}(i:=i+1)\{i\le n\}$	variable-assignment
7:	$\{i\le n\wedge i<n\}(i:=i+1)\{i\le n\}$	consequence(L) 5,6
8:	$i\le n\wedge i<n$	assumption
9:	$i<n$	$\wedge$ elim 8
10:	$n-i>0$	$i<n\vdash n-i>0$ 9
11:	$i\le n\wedge i<n\rightarrow n-i>0$	$\rightarrow$ intro 8–10
12:	integer Km	assumption
13:	$i\le n\wedge i<n\wedge n-i=Km$	assumption
14:	$n-i=Km$	$\wedge$ elim 13
15:	$n-(i+1)<Km$	$n-i=X\vdash n-(i+1)<X$ 14
16:	$i\le n\wedge i<n\wedge n-i=Km\rightarrow n-(i+1)<Km$	$\rightarrow$ intro 13–15
17:	$\{n-(i+1)<Km\}(i:=i+1)\{n-i<Km\}$	variable-assignment
18:	$\{i\le n\wedge i<n\wedge n-i=Km\}(i:=i+1)\{n-i<Km\}$	consequence(L) 16,17
19:	$\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i\le n\wedge\neg(i<n)\}$	while 7,11,12–18
20:	$i\le n\wedge\neg(i<n)$	assumption
21:	$i\le n$	$\wedge$ elim 20
22:	$\neg(i<n)$	$\wedge$ elim 20
23:	$i=n$	$i\le n, \neg(i<n)\vdash i=n$ 21,22
24:	$i\le n\wedge\neg(i<n)\rightarrow i=n$	$\rightarrow$ intro 20–23
25:	$\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i=n\}$	consequence(R) 19,24
26:	$\{0\le n\}(i:=0)\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i=n\}$	Ntuple 1,25

Completed Proof

1:	$\{0\le n\}(i:=0)\{i\le n\}$	variable-assignment	Initialization triple
2:	$i\le n\wedge i<n$	assumption	Body partial correctness
3:	$i<n$	$\wedge$ elim 2	
4:	$i+1\le n$	$A+1\le B\triangleq A<B$ 3	
5:	$i\le n\wedge i<n\rightarrow i+1\le n$	$\rightarrow$ intro 2–4	
6:	$\{i+1\le n\}(i:=i+1)\{i\le n\}$	variable-assignment	
7:	$\{i\le n\wedge i<n\}(i:=i+1)\{i\le n\}$	consequence(L) 5,6	
8:	$i\le n\wedge i<n$	assumption	Variant continuation condition
9:	$i<n$	$\wedge$ elim 8	
10:	$n-i>0$	$i<n\vdash n-i>0$ 9	
11:	$i\le n\wedge i<n\rightarrow n-i>0$	$\rightarrow$ intro 8–10	
12:	integer Km	assumption	Body decreases variant
13:	$i\le n\wedge i<n\wedge n-i=Km$	assumption	
14:	$n-i=Km$	$\wedge$ elim 13	
15:	$n-(i+1)<Km$	$n-i=X\vdash n-(i+1)<X$ 14	
16:	$i\le n\wedge i<n\wedge n-i=Km\rightarrow n-(i+1)<Km$	$\rightarrow$ intro 13–15	
17:	$\{n-(i+1)<Km\}(i:=i+1)\{n-i<Km\}$	variable-assignment	
18:	$\{i\le n\wedge i<n\wedge n-i=Km\}(i:=i+1)\{n-i<Km\}$	consequence(L) 16,17	
19:	$\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i\le n\wedge\neg(i<n)\}$	while 7,11,12–18	while triple
20:	$i\le n\wedge\neg(i<n)$	assumption	while exit implies final expectation
21:	$i\le n$	$\wedge$ elim 20	
22:	$\neg(i<n)$	$\wedge$ elim 20	
23:	$i=n$	$i\le n, \neg(i<n)\vdash i=n$ 21,22	
24:	$i\le n\wedge\neg(i<n)\rightarrow i=n$	$\rightarrow$ intro 20–23	
25:	$\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i=n\}$	consequence(R) 19,24	while triple + consequent
26:	$\{0\le n\}(i:=0)\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i=n\}$	Ntuple 1,25	program nTuple

Proof as Trees

Partial Correctness Tree

assign

7: $\{i\le n\wedge i<n\}(i:=i+1)\{i\le n\}$

pure logic

24: $i\le n\wedge\neg(i<n)\rightarrow i=n$ 19: $\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i\le n\wedge\neg(i<n)\}$

assign

1: $\{0\le n\}(i:=0)\{i\le n\}$ 25: $\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i=n\}$

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

Termination Tree

pure logic assign

16: $i\le n\wedge i<n\wedge n-i=Km\rightarrow n-(i+1)<Km$ 17: $\{n-(i+1)<Km\}(i:=i+1)\{n-i<Km\}$

pure logic

11: $i\le n\wedge i<n\rightarrow \boxed{n-i>}0$ 18: $\{i\le n\wedge i<n\boxed{\wedge n-i=}Km\}(i:=i+1)\boxed{\{n-i<}Km\}$

19: $\{i\le n\}$while $i<n$ do $i:=i+1$ od$\{i\le n\wedge\neg(i<n)\}$

Subtleties About Loop Invariants

- Can the following Ntuple be proved?

```
{0≤n}

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

{y=i×i ∧ 0≤i ∧ i≤n}

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

{y=n×n}
```

Is the invariant truly invariant?

Loop Body Correctness Proof Reduces to

11:	$y=i\times i\wedge 0\le i\wedge i\le n\wedge i<n$
	...
12:	$y+j=(i+1)\times(i+1)\wedge 0\le i+1\wedge i+1\le n$

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

$\perp$ very strong, implies everything, but implied by nothing but itself

An Inductive Invariant

```
{0≤n}

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

{y=i×i ∧ j = 2×i ∧ 0≤i ∧ i≤n}

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

{y=n×n}
```

Now the Body Proof Works

14:	$y=i\times i \wedge \boxed{j=2\times i+1 \wedge} 0\le i \wedge i\le n \wedge i<n$	assumption
15:	$y=i\times i$	$\wedge$ elim 14
16:	$j=2\times i+1$	$\wedge$ elim 14
17:	$0\le i$	$\wedge$ elim 14
18:	$i<n$	$\wedge$ elim 14
19:	$y+j=(i+1)\times(i+1)$	$y=i\times i, j=2\times i+1 \vdash y\ldots$ 15,16
20:	$j+2=2\times(i+1)+1$	$j=2\times i+1 \vdash j+2=2\times(i+\ldots$ 16
21:	$0\le i+1$	$A\le B \vdash A\le B+1$ 17
22:	$i+1\le n$	$A<N \vdash A+1\le N$ 18
23:	$y+j=(i+1)\times(i+1) \wedge j+2=2\times(i+1)+1 \wedge 0\le i+1 \wedge i+1\le n$	$\wedge$ intro 19,20,21,22

The Completed Proof (lines 1-24)

(From a different run, so there may be minor differences.)

Initialization

1:	$y=0 \wedge i=0 \wedge n\ge 0$	assumption
2:	$y=0$	$\wedge$ elim 1
3:	$i=0$	$\wedge$ elim 1
4:	$n\ge 0$	$\wedge$ elim 1
5:	$y=i\times i$	obviously, from 3,2
6:	$i\le n$	obviously, from 4,3
7:	$i\ge 0$	obviously, from 3
8:	$1=2\times i+1$	obviously, from 3
9:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge 1=2\times i+1$	$\wedge$ intro 5,6,7,8
10:	$y=0 \wedge i=0 \wedge n\ge 0 \rightarrow y=i\times i \wedge i\le n \wedge i\ge 0 \wedge 1=2\times i+1$	$\rightarrow$ intro 1-9
11:	$\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge 1=2\times i+1\}(j:=1)\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1\}$	variable-assignment
12:	$\{y=0 \wedge i=0 \wedge n\ge 0\}(j:=1)\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1\}$	consequence(L) 10,11

First assignment in loop body

13:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n$	assumption
14:	$y=i\times i$	$\wedge$ elim 13
15:	$i\ge 0$	$\wedge$ elim 13
16:	$j=2\times i+1$	$\wedge$ elim 13
17:	$i<n$	$\wedge$ elim 13
18:	$y+j=(i+1)\times(i+1)$	obviously, from 16,14
19:	$i+1\le n$	obviously, from 17
20:	$i+1\ge 0$	obviously, from 15
21:	$j+2=2\times(i+1)+1$	obviously, from 16
22:	$y+j=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j+2=2\times(i+1)+1$	$\wedge$ intro 18,19,20,21
23:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n \rightarrow y+j=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j+2=2\times(i+1)+1$	$\rightarrow$ intro 13-22
24:	$\{y+j=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j+2=2\times(i+1)+1\}(y:=y+j)\{y=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j+2=2\times(i+1)+1\}$	variable-assignment

The Completed Proof (lines 25-51)

Other assignments in loop body

25:	$\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n\}(y:=y+j)\{y=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j+2=2\times(i+1)+1\}$	consequence(L) 23,24
26:	$\{y=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j+2=2\times(i+1)+1\}(j:=j+2)\{y=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j=2\times(i+1)+1\}$	variable-assignment
27:	$\{y=(i+1)\times(i+1) \wedge i+1\le n \wedge i+1\ge 0 \wedge j=2\times(i+1)+1\}(i:=i+1)\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1\}$	variable-assignment
28:	$\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n\}(y:=y+j;j:=j+2;i:=i+1)\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1\}$	sequence 25,26,27

Continuation condition on _M

29:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n$	assumption
30:	$i<n$	$\wedge$ elim 29
31:	$n-i>0$	obviously, from 30
32:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n \rightarrow n-i>0$	$\rightarrow$ intro 29-31

Termination of loop body

33:	integer Km	assumption
34:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n \wedge n-i=Km$	assumption
35:	$n-i=Km$	$\wedge$ elim 34
36:	$n-(i+1)<Km$	obviously, from 35
37:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n \wedge n-i=Km \rightarrow n-(i+1)<Km$	$\rightarrow$ intro 34-36
38:	$\{n-(i+1)<Km\}(y:=y+j)\{n-(i+1)<Km\}$	variable-assignment
39:	$\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n \wedge n-i=Km\}(y:=y+j)\{n-(i+1)<Km\}$	consequence(L) 37,38
40:	$\{n-(i+1)<Km\}(j:=j+2)\{n-(i+1)<Km\}$	variable-assignment
41:	$\{n-(i+1)<Km\}(i:=i+1)\{n-i<Km\}$	variable-assignment
42:	$\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge i<n \wedge n-i=Km\}(y:=y+j;j:=j+2;i:=i+1)\{n-i<Km\}$	sequence 39,40,41
43:	$\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1\}$ while $i<n$ do $y:=y+j;j:=j+2;i:=i+1$ od $\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge \neg(i<n)\}$	while 28,32,33-42

Exit consequence

44:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge \neg(i<n)$	assumption
45:	$y=i\times i$	$\wedge$ elim 44
46:	$i\le n$	$\wedge$ elim 44
47:	$\neg(i<n)$	$\wedge$ elim 44
48:	$y=n\times n$	obviously, from 47,46,45
49:	$y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1 \wedge \neg(i<n) \rightarrow y=n\times n$	$\rightarrow$ intro 44-48
50:	$\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1\}$ while $i<n$ do $y:=y+j;j:=j+2;i:=i+1$ od $\{y=n\times n\}$	consequence(R) 43,49
51:	$\{y=0 \wedge i=0 \wedge n\ge 0\}(j:=1)\{y=i\times i \wedge i\le n \wedge i\ge 0 \wedge j=2\times i+1\}$ while $i<n$ do $y:=y+j;j:=j+2;i:=i+1$ od $\{y=n\times n\}$	Ntuple 12,50

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≥0 ∧ length(a)=n}

i := 0; j := n

{i≤n ∧ i≥0 ∧ 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→ 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 →a[j] = 0}
```

Salient Parts of Proof Initialization

```
1: 0≤n∧length(a)=n
2: 0≤n
3: length(a)=n
4: 0≤0
5: n<n
6: ⊥
7: a[n]=0
8: n<n→a[n]=0
9: 0≤n∧0≤0∧length(a)=n∧(n<n→a[n]=0)
10: 0≤n∧length(a)=n→0≤n∧0≤0∧length(a)=n∧(n<n→a[n]=0)
11: {0≤n∧0≤0∧length(a)=n∧(n<n→a[n]=0)}(i:=0){i≤n∧0≤i∧length(a)=n∧(n<n→a[n]=0)}
12: {0≤n∧length(a)=n}(i:=0){i≤n∧0≤i∧length(a)=n∧(n<n→a[n]=0)}
13: {i≤n∧0≤i∧length(a)=n∧(n<n→a[n]=0)}(j:=n){i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)}
14: {0≤n∧length(a)=n}(i:=0;j:=n){i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)}
```

Loop Body Partial Correctness

(Collapsed)

```
15: i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧i<n                                   assumption
16: (a[i]=0→i+1≤n∧0≤i+1∧length(a)=n∧(i<n→a[i]=0))
    ∧(¬(a[i]=0)→i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0))∧0≤i∧i<length(a)      {cut}
17: i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧i<n→(a[i]=0→i+1≤n∧0≤i+1∧length(a)=n∧(i<n→a[i]=0))
    ∧(¬(a[i]=0)→i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0))∧0≤i∧i<length(a)      → intro 15-16
18: {i+1≤n∧0≤i+1∧length(a)=n∧(i<n→a[i]=0)}(j:=i){i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0)}      variable-assignment
19: {i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0)}skip{i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0)}      skip
20: {(a[i]=0→i+1≤n∧0≤i+1∧length(a)=n∧(i<n→a[i]=0))∧(¬(a[i]=0)→i+1≤n∧0≤i+1∧length(a)=n∧(j<n
    →a[j]=0))∧0≤i∧i<length(a)}if a[i]=0 then j:=i else skip fi{i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0)}      choice 18,19
21: {i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧i<n}if a[i]=0 then j:=i else skip fi{i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0)}      consequence(L) 17,20
22: {i+1≤n∧0≤i+1∧length(a)=n∧(j<n→a[j]=0)}(i:=i+1){i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)}      variable-assignment
23: {i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧i<n}(if a[i]=0 then j:=i else skip fi;i:=i+1){i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)}      equence 21,22
```

Variant Continuation and Decrease

(Collapsed)

24:	i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧i<n	assumption
25:	n−i>0	{cut}
26:	i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧i<n→n−i>0	→ intro 24–25
27:	integer Km	assumption
28:	{i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧i<n∧n−i=Km}(if a[i]=0 then j:=i else skip fi;i:=i+1){n−i<Km}	{cut}

Remainder

29:	{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 {i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧¬(i<n)}	while 23,26,27–28
30:	i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧¬(i<n)	assumption
31:	j<n→a[j]=0	∧ elim 30
32:	i≤n∧0≤i∧length(a)=n∧(j<n→a[j]=0)∧¬(i<n)→j<n→a[j]=0	→ intro 30–31
33:	{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→a[j]=0}	consequence(R) 29,32
34:	{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→a[j]=0}	Ntuple 14,33

Without collapsing, the proof is about 74 lines.

Quantifiers

- Quantifiers are handy representing information about arrays, e.g.
- $\forall i\ ((0 < i) \wedge (i < n)) \rightarrow a[i-1] \leq a[i]$
- $\exists i\ ((0 \leq i) \wedge (i < n) \wedge a[i] = 0)$

Read-Only Example with Quantifiers

A program whose correct working depends on values stored in the array.

{∃x.(0≤x∧x<length(a)∧a[x]=0)}(i:=0)
{0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}while a[i]≠0 do i:=i+1 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**.

Start of Proof

. . .
{∃x.(0≤x∧x<length(a)∧a[x]=0)}(i:=0)
1: {0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}
while a[i]≠0 do i:=i+1 od{a[i]=0}

Consequence for Initialization

1:	∃x.(0≤x∧x<length(a)∧a[x]=0)	assumption
	. . .	
2:	0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	

We'll use ∃ elim.

1:	∃x.(0≤x∧x<length(a)∧a[x]=0)	assumption
2:	integer i1	assumption
3:	0≤i1∧i1<length(a)∧a[i1]=0	assumption
	. . .	
4:	0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	
5:	0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	∃ elim 1,2–4

Done with Initialization

1:	∃x.(0≤x∧x<length(a)∧a[x]=0)	assumption
2:	integer i1	assumption
3:	0≤i1∧i1<length(a)∧a[i1]=0	assumption
4:	0≤i1	∧ elim 3
5:	i1<length(a)	∧ elim 3
6:	0≤0	A≤A
7:	0<length(a)	A≤B, B<C ⊢ A<C 4,5
8:	0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	∧ intro 6,7,1
9:	0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	∃ elim 1,2–8
10:	∃x.(0≤x∧x<length(a)∧a[x]=0) →0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	→ intro 1–9
11:	{0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)} (i:=0){0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}	variable-assignment
12:	{∃x.(0≤x∧x<length(a)∧a[x]=0)}(i:=0) {0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}	consequence(L) 10,11

On to the while loop

13: $\{0\le i\wedge i<\mathrm{length}(a)\wedge\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)\}$
while $a[i]\ne 0$ do $i:=i+1$ od$\{a[i]=0\}$

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

...

14: $\{0\le i\wedge i<\mathrm{length}(a)\wedge\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)\wedge a[i]\ne 0\}$
$(i:=i+1)\{0\le i\wedge i<\mathrm{length}(a)\wedge\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)\}$

Use the assignment rule.
This generates a consequent implication.

16:	$0\le i\wedge i<\mathrm{length}(a)\wedge\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)\wedge a[i]\ne 0$	assumption
	...	
17:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	
18:	$0\le i\wedge i<\mathrm{length}(a)\wedge\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)\wedge a[i]\ne 0$ $\rightarrow 0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	$\rightarrow$ intro 16–17

It is tempting to try backward $\wedge$ Introduction now. Don't!

Use Forward $\wedge$ Elimination

16:	$0\le i\wedge i<\mathrm{length}(a)\wedge\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)\wedge a[i]\ne 0$	assumption
17:	$0\le i$	$\wedge$ elim 16
18:	$i<\mathrm{length}(a)$	$\wedge$ elim 16
19:	$\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	$\wedge$ elim 16
20:	$a[i]\ne 0$	$\wedge$ elim 16
	...	
21:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	

Now use $\exists$ Elimination, introducing i2

16:	$0\le i\wedge i<\mathrm{length}(a)\wedge\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)\wedge a[i]\ne 0$	assumption
17:	$0\le i$	$\wedge$ elim 16
18:	$i<\mathrm{length}(a)$	$\wedge$ elim 16
19:	$\exists x.(i\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	$\wedge$ elim 16
20:	$a[i]\ne 0$	$\wedge$ elim 16
21:	integer i2	assumption
22:	$i\le i2\wedge i2<\mathrm{length}(a)\wedge a[i2]=0$	assumption
	...	
23:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	
24:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	$\exists$ elim 19,21–23

Note that i2 cannot equal i. Why?
We need to show this, to get the post-condition.

To get $i2\neq i$

- Set up a bifurcation using $i \le i2$, i.e. $i < i2 \vee i = i2$.
- Then use $\vee$ Elimination:

20:	$a[i]\ne 0$	$\wedge$ elim 16
21:	integer i2	assumption
22:	$i\le i2\wedge i2<\mathrm{length}(a)\wedge a[i2]=0$	assumption
23:	$i\le i2$	$\wedge$ elim 22
24:	$i<i2\vee i=i2$	$A\le B\triangleq A<B\vee A=B$ 23
25:	$i2<\mathrm{length}(a)$	$\wedge$ elim 22
26:	$a[i2]=0$	$\wedge$ elim 22
27:	$i<i2$	assumption
	...	
28:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	
29:	$i=i2$	assumption
	...	
30:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	
31:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	$\vee$ elim 24,27–28,29–30
32:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	$\exists$ elim 19,21–31

Now try backward $\wedge$ Introductions.

- What to use for x in $\exists$x…?

26:	$a[i2]=0$	$\wedge$ elim 22
27:	$i<i2$	assumption
	...	
28:	$0\le i+1$	
	...	
29:	$i+1<\mathrm{length}(a)$	
	...	
30:	$\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	
31:	$0\le i+1\wedge i+1<\mathrm{length}(a)\wedge\exists x.(i+1\le x\wedge x<\mathrm{length}(a)\wedge a[x]=0)$	$\wedge$ intro 28,29,30

i2 is the obvious choice for x

23:	$i \le i2$	∧ elim 22
24:	$i < i2 \lor i = i2$	$A \le B \triangleq A < B \lor A = B$ 23
25:	$i2 < length(a)$	∧ elim 22
26:	$a[i2] = 0$	∧ elim 22
27:	$i < i2$	assumption
	...	
28:	$0 \le i+1$	
	...	
29:	$i+1 < length(a)$	
	...	
30:	$i+1 \le i2 \land i2 < length(a) \land a[i2] = 0$	
31:	$\exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$	∃ intro 30
32:	$0 \le i+1 \land i+1 < length(a) \land \exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$	∧ intro 28,29,31

Closure of the top half

23:	$i \le i2$	∧ elim 22
24:	$i < i2 \lor i = i2$	$A \le B \triangleq A < B \lor A = B$ 23
25:	$i2 < length(a)$	∧ elim 22
26:	$a[i2] = 0$	∧ elim 22
27:	$i < i2$	assumption
28:	$0 \le i+1$	$A \le B \vdash A \le B+1$ 17
29:	$i+1 < length(a)$	$i < i2,\ i2 < length(a) \vdash \ldots$ 27,25
30:	$i+1 \le i2$	$A < N \vdash A+1 \le N$ 27
31:	$i+1 \le i2 \land i2 < length(a) \land a[i2] = 0$	∧ intro 30,25,26
32:	$\exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$	∃ intro 31
33:	$0 \le i+1 \land i+1 < length(a) \land \exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$	∧ intro 28,29,32

Bottom Half: We want a contradiction

20:	$a[i] \ne 0$	∧ elim 16
21:	integer i2	assumption
22:	$i \le i2 \land i2 < length(a) \land a[i2] = 0$	assumption
23:	$i \le i2$	∧ elim 22
24:	$i < i2 \lor i = i2$	$A \le B \triangleq A < B \lor A = B$ 23
25:	$i2 < length(a)$	∧ elim 22
26:	$a[i2] = 0$	∧ elim 22
34:	$i = i2$	
	...	
35:	$0 \le i+1 \land i+1 < length(a) \land \exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$	

Introduce ⊥, then figure out how to get it.

How to pick _B1?

20:	$a[i] \ne 0$	
21:	integer i2	
22:	$i \le i2 \land i2 < length(a) \land a[i2] = 0$	
23:	$i \le i2$	
24:	$i < i2 \lor i = i2$	
25:	$i2 < length(a)$	
26:	$a[i2] = 0$	
34:	$i = i2$	assumption
	...	
35:	_B1	
	...	
36:	¬_B1	
37:	$\bot$	¬ elim 35,36
38:	$0 \le i+1 \land i+1 < length(a) \land \exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$	contra (constructive) 37

For _B1 use a[i] = 0

34:	$i = i2$
	...
35:	$a[i] = 0$
	...
36:	$\neg(a[i] = 0)$
37:	$\bot$
38:	$0 \le i+1 \land i+1 < length(a) \land \exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$

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

$$A \ne B \triangleq \neg(A = B)$$

Status

26:	$a[i2] = 0$	∧ elim 22
34:	$i = i2$	assumption
	...	
35:	$a[i] = 0$	
36:	$\neg(a[i] = 0)$	$A \ne B \triangleq \neg(A = B)$ 20
37:	$\bot$	¬ elim 35,36
38:	$0 \le i+1 \land i+1 < length(a) \land \exists x.(i+1 \le x \land x < length(a) \land a[x] = 0)$	contra (constructive) 37

Now use equality, to get 35 from 26 and 34.

Closure

26:	a[i2]=0	∧ elim 22
33:	i=i2	assumption
34:	a[i]=0	equality-substitution 33,25
35:	¬(a[i]=0)	A≠B≜¬(A=B) 19
36:	⊥	¬ elim 34,35
37:	0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	contra (constructive) 36

What about termination?

- What to use for _M?

Completed proof, using several lemmas 1/3

1:	∃x.(0≤x∧x<length(a)∧a[x]=0)	assumption
2:	integer i1	assumption
3:	0≤i1∧i1<length(a)∧a[i1]=0	assumption
4:	0≤i1	∧ elim 3
5:	i1<length(a)	∧ elim 3
6:	0≤0	A≤A
7:	0<length(a)	A≤B, B<C ⊢ A<C 4,5
8:	0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	∧ intro 6,7,1
9:	0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	∃ elim 1,2–8
10:	∃x.(0≤x∧x<length(a)∧a[x]=0)→0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)	→ intro 1–9
11:	{0≤0∧0<length(a)∧∃x.(0≤x∧x<length(a)∧a[x]=0)} (i:=0){0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}	variable-assignment
12:	{∃x.(0≤x∧x<length(a)∧a[x]=0)}(i:=0){0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}	consequence(L) 10,11
13:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)	assumption
14:	0≤i∧i<length(a)	∧ elim(L) 13
15:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)→0≤i∧i<length(a)	→ intro 13–14

2/3

16:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0	assumption
17:	0≤i	∧ elim 16
18:	∃x.(i≤x∧x<length(a)∧a[x]=0)	∧ elim 16
19:	a[i]≠0	∧ elim 16
20:	integer i2	assumption
21:	i≤i2∧i2<length(a)∧a[i2]=0	assumption
22:	i≤i2	∧ elim 21
23:	i<i2∨i=i2	A≤B≜A<B∨A=B 22
24:	i2<length(a)	∧ elim 21
25:	a[i2]=0	∧ elim 21
26:	i<i2	assumption
27:	0≤i+1	A≤B ⊢ A≤B+1 17
28:	i+1<length(a)	i<i2, i2<length(a) ... 26,24
29:	i+1≤i2	A<N ⊢ A+1≤N 26
30:	i+1≤i2∧i2<length(a)∧a[i2]=0	∧ intro 29,24,25
31:	∃x.(i+1≤x∧x<length(a)∧a[x]=0)	∃ intro 30
32:	0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	∧ intro 27,28,31
33:	i=i2	assumption
34:	a[i]=0	equality-substitution 33,25
35:	¬(a[i]=0)	A≠B≜¬(A=B) 19
36:	⊥	¬ elim 34,35
37:	0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	contra (constructive) 36
38:	0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	∨ elim 23,26–32,33–37
39:	0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	∃ elim 18,20–38

3/3

40:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0 →0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	→ intro 16–39
41:	{0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)} (i:=i+1){0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}	variable-assignment
42:	{0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0} (i:=i+1){0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}	consequence(L) 40,41
43:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0	assumption
44:	i<length(a)	∧ elim 43
45:	length(a)−i>0	i<length(a) ⊢ length(a... 44
46:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0→length(a)−i>0	→ intro 43–45
47:	integer Km	assumption
48:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0∧length(a)−i=Km	assumption
49:	length(a)−i=Km	∧ elim 48
50:	length(a)−(i+1)<Km	length(a)−i=X ⊢ lengt... 49
51:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0∧length(a)−i=Km →length(a)−(i+1)<Km	→ intro 48–50
52:	{length(a)−(i+1)<Km}(i:=i+1){length(a)−i<Km}	variable-assignment
53:	{0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0∧length(a)−i=Km} (i:=i+1){length(a)−i<Km}	consequence(L) 51,52
54:	{0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}while a[i]≠0 do i:=i+1 od {0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧¬(a[i]≠0)}	while 15,42,46,47–53
55:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧¬(a[i]≠0)	assumption
56:	¬(a[i]≠0)	∧ elim 55
57:	a[i]=0	A=B≜¬(A≠B) 56
58:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧¬(a[i]≠0)→a[i]=0	→ intro 55–57
59:	{0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}while a[i]≠0 do i:=i+1 od{a[i]=0}	consequence(R) 54,58
60:	{∃x.(0≤x∧x<length(a)∧a[x]=0)}(i:=0) {0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)}while a[i]≠0 do i:=i+1 od{a[i]=0}	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⊕i→v** is the array that is like **a** except that the value of **a[i]** is **v**.
- (a⊕i→v)[i] = v — same index, new value
- (a⊕i→v)[j] = a[j] if j ≠ i — different index, old value

Array Element Assignment

- Consider a triple
  {??}
  a[i] := E
  {B}
- If this is viewed mathematically as an array replacement:
  {??}
  a := (a⊕i→E)
  {B}
- 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 ∧ a[1] > 50}
- {(a⊕1→99)[0] = 98 ∧ (a⊕1→99)[1] > 50}
  a := (a⊕1→99)
  {a[0] = 98 ∧ a[1] > 50}

JAPE Example

...
1: {_Unknown}(a[1]:=99){a[0]=98∧a[1]>50}

- {??}
  a[1] := 99
  {a[0] = 98 ∧ a[1] > 50}
- {(a⊕1→99)[0] = 98 ∧ (a⊕1→99)[1] > 50}
  a := (a⊕1→99)
  {a[0] = 98 ∧ a[1] > 50}

1: {(a⊕1↦99)[0]=98∧(a⊕1↦99)[1]>50∧0≤1∧1<length(a)} (a[1]:=99){a[0]=98∧a[1]>50} array-element-assignment

Bornat's Array Assignment Rule in JAPE

assignment

$$\{\ (a[E]\ computes) \wedge (F\ computes)\ \wedge B^{a}_{a\oplus E\mapsto F}\ \}\ a[E] := F\ \{B\}$$

Formula B with each free a replaced with a⊕E→F

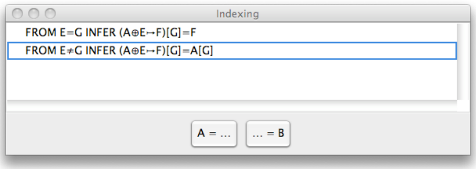
B

What *a[E] computes* means:

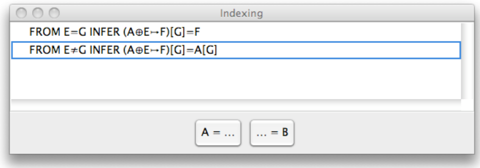
$$\frac{\begin{matrix}\vdots \\ E\ computes\end{matrix} \qquad \begin{matrix}\vdots \\ 0 \le E < \mathrm{length}(a)\end{matrix}}{a[E]\ computes}$$

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:



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

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

...
1: $\{a[0]=98 \wedge a[1]=5\}(a[1]:=99)$
$\{a[0]=98 \wedge a[1]>50\}$

Candidates for Simplification

...
1: $a[0]=98 \wedge a[1]=5$
$\rightarrow \boxed{(a\oplus 1\mapsto 99)[0]}=98 \wedge \boxed{(a\oplus 1\mapsto 99)[1]}>50 \wedge 0 \le 1 \wedge 1 < length(a)$

2: $\{(a\oplus 1\mapsto 99)[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)\}$ array-element-assignment
$(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$

3: $\{a[0]=98 \wedge a[1]=5\}(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$ consequence(L) 1,2

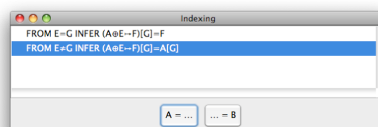
Using Indexing Rule where E≠G

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

...
1: $a[0]=98 \wedge a[1]=5$
$\rightarrow (a\oplus 1\mapsto 99)[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)$

2: $\{(a\oplus 1\mapsto 99)[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)\}$ array-element-assignment
$(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$

3: $\{a[0]=98 \wedge a[1]=5\}(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$ consequence(L) 1,2



Result

when $0 \neq 1$, $(a\oplus 1\rightarrow 99)[0]$ is same as $a[0]$ whatever that happens to be.

...
1: $1 \neq 0$
...
2: $a[0]=98 \wedge a[1]=5 \rightarrow \boxed{a[0]}=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)$

3: $a[0]=98 \wedge a[1]=5$ FROM E≠G INFER... 1,2
$\rightarrow (a\oplus 1\mapsto 99)[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)$

4: $\{(a\oplus 1\mapsto 99)[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)\}$ array-element-assi...
$(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$

5: $\{a[0]=98 \wedge a[1]=5\}(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$ consequence(L) 3,4

Using Indexing Rule where E=G

...
1: $1 \neq 0$
...
2: $a[0]=98 \wedge a[1]=5 \rightarrow a[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)$

3: $a[0]=98 \wedge a[1]=5$ FROM E≠G INFER... 1,2
$\rightarrow (a\oplus 1\mapsto 99)[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)$

4: $\{(a\oplus 1\mapsto 99)[0]=98 \wedge (a\oplus 1\mapsto 99)[1]>50 \wedge 0 \le 1 \wedge 1 < length(a)\}$ array-element-assi...
$(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$

5: $\{a[0]=98 \wedge a[1]=5\}(a[1]:=99)\{a[0]=98 \wedge a[1]>50\}$ consequence(L) 3,4

Result

where 1=1, (a⊕1→99)[1] is same as RHS of assignment

...
1: 1≠0
...
2: a[0]=98∧a[1]=5→a[0]=98∧99>50∧0≤1∧1<length(a)
3: a[0]=98∧a[1]=5→a[0]=98∧(a⊕1↦99)[1]>50∧0≤1∧1<length(a) FROM E=G INFER (... 2
4: a[0]=98∧a[1]=5 →(a⊕1↦99)[0]=98∧(a⊕1↦99)[1]>50∧0≤1∧1<length(a) FROM E≠G INFER... 1,3
5: {(a⊕1↦99)[0]=98∧(a⊕1↦99)[1]>50∧0≤1∧1<length(a)} (a[1]:=99){a[0]=98∧a[1]>50} array-element-assi...
6: {a[0]=98∧a[1]=5}(a[1]:=99){a[0]=98∧a[1]>50} consequence(L) 4,5

Using Bounds Inference

Once an index is used, it is assumed to be valid.

Select only indexed expression, not entire statement

Extras Window Help
A=A
A = ..
.. = B
obviously
boundedness from (in)equality

Rule

1: a[i]=2
...
2: a[i]+1=3∧0≤i∧i<length(a)
3: a[i]=2→a[i]+1=3∧0≤i∧i<length(a) → intro 1-2
4: a[i]=2→(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a) FROM E=G INFER (A⊕E↦F)[G]=F 3
5: {(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=3} array-element-assignment
6: {a[i]=2}(a[i]:=a[i]+1){a[i]=3} consequence(L) 4,5

Result

1: a[i]=2 assumption
2: 0≤i∧i<length(a) bounded 1
...
3: a[i]+1=3∧0≤i∧i<length(a)
4: a[i]=2→a[i]+1=3∧0≤i∧i<length(a) → intro 1-3
5: a[i]=2→(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a) FROM E=G INFER (A⊕E↦F)[G]=F 4
6: {(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<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)

{a[i]=2}(a[i]:=a[i]+1){a[i]=3}

Extras Window Help
A=A
A = ..
.. = B
obviously
boundedness from (in)equality

Rule

Equation

Goal selected, and expression selected

1: a[i]=2
2: 0≤i∧i<length(a)
3: 0≤i
4: i<length(a) ∧ elim 2
...
5: a[i]+1=3
6: a[i]+1=3∧0≤i∧i<length(a) ∧ intro 5,3,4
7: a[i]=2→a[i]+1=3∧0≤i∧i<length(a) → intro 1-6
8: a[i]=2→(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a) FROM E=G INFER (A⊕E↦F)[G]=F 7
9: {(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=3} array-element-assignment
10: {a[i]=2}(a[i]:=a[i]+1){a[i]=3} consequence(L) 8,9

Result

1: a[i]=2 assumption
2: 0≤i∧i<length(a) bounded 1
3: 0≤i ∧ elim 2
4: i<length(a) ∧ elim 2
...
5: 2+1=3

was a[i]

6: a[i]+1=3 equality-substitution 1,5
7: a[i]+1=3∧0≤i∧i<length(a) ∧ intro 6,3,4
8: a[i]=2→a[i]+1=3∧0≤i∧i<length(a) → intro 1-7
9: a[i]=2→(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a) FROM E=G INFER (A⊕E↦F)[G]=F 8
10: {(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=3} array-element-assignment
11: {a[i]=2}(a[i]:=a[i]+1){a[i]=3} consequence(L) 9,10

Completed Proof

1: a[i]=2 assumption
2: 0≤i∧i<length(a) bounded 1
3: 0≤i ∧ elim 2
4: i<length(a) ∧ elim 2
5: 2+1=3 obviously
6: a[i]+1=3 equality-substitution 1,5
7: a[i]+1=3∧0≤i∧i<length(a) ∧ intro 6,3,4
8: a[i]=2→a[i]+1=3∧0≤i∧i<length(a) → intro 1-7
9: a[i]=2→(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a) FROM E=G INFER (A⊕E↦F)[G]=F 8
10: {(a⊕i↦a[i]+1)[i]=3∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=3} array-element-assignment
11: {a[i]=2}(a[i]:=a[i]+1){a[i]=3} 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}
```

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

```
...
1: {a[i]=0}(a[i]:=a[i]+1){_B2}
...
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∧0≤i∧i<length(a)}
2: {(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=2}   array-element-assignment
3: {a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}                  sequence 1,2
```

```
...
1: {a[i]=0}(a[i]:=a[i]+1){a[i]+1=2∧0≤i∧i<length(a)}
2: {a[i]=0}(a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}   FROM E=G INFER (A⊕E↦F)[G]=F 1
3: {(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=2}   array-element-assignment
4: {a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}                  sequence 2,3
```

Early Simplification, step 2

```
...
1: a[i]=0→(a⊕i↦a[i]+1)[i]+1=2∧0≤i∧i<length(a)
2: {(a⊕i↦a[i]+1)[i]+1=2∧0≤i∧i<length(a)}
   (a[i]:=a[i]+1){a[i]+1=2∧0≤i∧i<length(a)}                      array-element-assignment
3: {a[i]=0}(a[i]:=a[i]+1){a[i]+1=2∧0≤i∧i<length(a)}             consequence(L) 1,2
4: {a[i]=0}(a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}   FROM E=G INFER (A⊕E↦F)[G]=F 3
5: {(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=2}   array-element-assignment
6: {a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}                  sequence 4,5
```

```
...
1: a[i]=0→a[i]+1+1=2∧0≤i∧i<length(a)
2: a[i]=0→(a⊕i↦a[i]+1)[i]+1=2∧0≤i∧i<length(a)                   FROM E=G INFER (A⊕E↦F)[G]=F 1
3: {(a⊕i↦a[i]+1)[i]+1=2∧0≤i∧i<length(a)}
   (a[i]:=a[i]+1){a[i]+1=2∧0≤i∧i<length(a)}                      array-element-assignment
4: {a[i]=0}(a[i]:=a[i]+1){a[i]+1=2∧0≤i∧i<length(a)}             consequence(L) 2,3
5: {a[i]=0}(a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}   FROM E=G INFER (A⊕E↦F)[G]=F 4
6: {(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=2}   array-element-assignment
7: {a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}                  sequence 5,6
```

Completed Proof

```
 1: a[i]=0                                                        assumption
 2: 0≤i∧i<length(a)                                               bounded 1
 3: 0≤i                                                           ∧ elim 2
 4: i<length(a)                                                   ∧ elim 2
 5: 0+1+1=2                                                       obviously
 6: a[i]+1+1=2                                                    equality-substitution 1,5
 7: a[i]+1+1=2∧0≤i∧i<length(a)                                    ∧ intro 6,3,4
 8: a[i]=0→a[i]+1+1=2∧0≤i∧i<length(a)                             → intro 1-7
 9: a[i]=0→(a⊕i↦a[i]+1)[i]+1=2∧0≤i∧i<length(a)                    FROM E=G INFER (A⊕E↦F)[G]=F 8
10: {(a⊕i↦a[i]+1)[i]+1=2∧0≤i∧i<length(a)}
    (a[i]:=a[i]+1){a[i]+1=2∧0≤i∧i<length(a)}                      array-element-assignment
11: {a[i]=0}(a[i]:=a[i]+1){a[i]+1=2∧0≤i∧i<length(a)}             consequence(L) 9,10
12: {a[i]=0}(a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}   FROM E=G INFER (A⊕E↦F)[G]=F 11
13: {(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=2}   array-element-assignment
14: {a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}                  sequence 12,13
```

Deferred Simplification Alternative

- Use sequence, then array-assignment twice (from the bottom up) to get to this point:

```
{a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2} [1]
...
                messy
1: a[i]=0→(a⊕i↦a[i]+1⊕i↦(a⊕i↦a[i]+1)[i]+1)[i]=2∧0≤i∧i<length(a)
2: {(a⊕i↦a[i]+1⊕i↦(a⊕i↦a[i]+1)[i]+1)[i]=2∧0≤i∧i<length(a)}
   (a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}           array-element-assignment
3: {a[i]=0}(a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}   consequence(L) 1,2
4: {(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=2}   array-element-assignment
5: {a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}                  sequence 3,4
Provided:
DISTINCT a, i
```

Completed Proof, using Deferred Simplification

```
1: a[i]=0                                                   assumption
2: 0+1+1=2                                                  obviously
3: a[i]+1+1=2                                               equality-substitution 1,2
4: (a⊕i↦a[i]+1)[i]+1=2                                      FROM E=G INFER (A⊕E↦F)[G]=F 3
5: (a⊕i↦a[i]+1⊕i↦(a⊕i↦a[i]+1)[i]+1)[i]=2                    FROM E=G INFER (A⊕E↦F)[G]=F 4
6: 0≤i∧i<length(a)                                          bounded 1
7: 0≤i                                                      ∧ elim 6
8: i<length(a)                                              ∧ elim 6
9: (a⊕i↦a[i]+1⊕i↦(a⊕i↦a[i]+1)[i]+1)[i]=2∧0≤i∧i<length(a)    ∧ intro 5,7,8
10: a[i]=0→(a⊕i↦a[i]+1⊕i↦(a⊕i↦a[i]+1)[i]+1)[i]=2∧0≤i∧i<length(a)   → intro 1-9
11: {(a⊕i↦a[i]+1⊕i↦(a⊕i↦a[i]+1)[i]+1)[i]=2∧0≤i∧i<length(a)}        array-element-assignment
    (a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}
12: {a[i]=0}(a[i]:=a[i]+1){(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}      consequence(L) 10,11
13: {(a⊕i↦a[i]+1)[i]=2∧0≤i∧i<length(a)}(a[i]:=a[i]+1){a[i]=2}      array-element-assignment
14: {a[i]=0}(a[i]:=a[i]+1;a[i]:=a[i]+1){a[i]=2}                    sequence 12,13
```

original program

Provided:
DISTINCT a, i

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=length(a)∧n=length(b)∧0≤n}

… array-copy program …

{∀j.((0≤j∧j<n)→(b[j]=a[j]))}

assuming indices run 0, … n-1

Proposed Program

```
{n=length(a)∧n=length(b)∧0≤n}
(i := 0)
while i < n
   do
   b[i] := a[i];
   i := i+1
   od
{∀j.((0≤j∧j<n)→(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)∧n=length(b)∧0≤n}

(i := 0)

{n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n ∧∀j.((0≤j∧j<i)→(b[j]=a[j]))}

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

{∀j.((0≤j∧j<n)→(b[j]=a[j]))}

Invariant Part of Loop Body

```
18: n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n∧∀j.((0≤j∧j<i)→(b[j]=a[j]))∧i<n      assumption
19: n=length(a)∧n=length(b)∧0≤n∧0≤i+1∧i+1≤n∧∀j.((0≤j∧j<i+1)→((b⊕i↦a[i])[j]=a[j]))∧0≤i∧i<length(b)∧i<length(a)      (cut)
20: n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n∧∀j.((0≤j∧j<i)→(b[j]=a[j]))∧i<n      → intro 18-19
    →n=length(a)∧n=length(b)∧0≤n∧0≤i+1∧i+1≤n∧∀j.((0≤j∧j<i+1)→((b⊕i↦a[i])[j]=a[j]))∧0≤i∧i<length(b)∧i<length(a)
21: {n=length(a)∧n=length(b)∧0≤n∧0≤i+1∧i+1≤n∧∀j.((0≤j∧j<i+1)→((b⊕i↦a[i])[j]=a[j]))∧0≤i∧i<length(b)∧i<length(a)}      array-element-assignment
    (b[i]:=a[i]){n=length(a)∧n=length(b)∧0≤n∧0≤i+1∧i+1≤n∧∀j.((0≤j∧j<i+1)→(b[j]=a[j]))}
22: {n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n∧∀j.((0≤j∧j<i)→(b[j]=a[j]))∧i<n}(b[i]:=a[i])      consequence(L) 20,21
    {n=length(a)∧n=length(b)∧0≤n∧0≤i+1∧i+1≤n∧∀j.((0≤j∧j<i+1)→(b[j]=a[j]))}
23: {n=length(a)∧n=length(b)∧0≤n∧0≤i+1∧i+1≤n∧∀j.((0≤j∧j<i+1)→(b[j]=a[j]))}      variable-assignment
    (i:=i+1){n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n∧∀j.((0≤j∧j<i)→(b[j]=a[j]))}
24: {n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n∧∀j.((0≤j∧j<i)→(b[j]=a[j]))∧i<n}      sequence 22,23
    (b[i]:=a[i];i:=i+1){n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n∧∀j.((0≤j∧j<i)→(b[j]=a[j]))}
```

Details Collapsed on Previous Slide

18:	n=length(a)∧n=length(b)∧0≤n∧0≤i∧i≤n∧∀j.((0≤j∧j<i)→(b[j]=a[j]))∧i<n	assumption
19:	n=length(a)	∧ elim 18
20:	n=length(b)	∧ elim 18
21:	0≤n	∧ elim 18
22:	0≤i	∧ elim 18
23:	∀j.((0≤j∧j<i)→(b[j]=a[j]))	∧ elim 18
24:	i<n	∧ elim 18
25:	0≤i+1	A≤B ⊢ A≤B+1 22
26:	i+1≤n	A+1≤B≜A<B 24
27:	integer i3	assumption
28:	0≤i3∧i3<i+1	assumption
29:	0≤i3	∧ elim 28
30:	i3<i+1	∧ elim 28
31:	i3<i∨i3=i	A<B+1 ⊢ A<B∨A=B 30
32:	i3<i	assumption
33:	0≤i3∧i3<i	∧ intro 29,32
34:	(0≤i3∧i3<i)→(b[i3]=a[i3])	∀ elim 23
35:	b[i3]=a[i3]	→ elim 34,33
36:	i3≠i	A<B ⊢ A≠B 32
37:	i≠i3	A≠B≜B≠A 36
38:	(b⊕i↦a[i])[i3]=a[i3]	FROM E≠G INFER (A⊕E↦F)[G]=... 37,35
39:	i3=i	assumption
40:	i=i3	A=B≜B=A 39
41:	a[i]=a[i3]	i=j ⊢ a[i]=a[j] 40
42:	(b⊕i↦a[i])[i3]=a[i3]	FROM E=G INFER (A⊕E↦F)[G]=F 40,41
43:	(b⊕i↦a[i])[i3]=a[i3]	∨ elim 31,32-38,39-42
44:	(0≤i3∧i3<i+1)→((b⊕i↦a[i])[i3]=a[i3])	→ intro 28-43
45:	∀j.((0≤j∧j<i+1)→((b⊕i↦a[i])[j]=a[j]))	∀ intro 27-44
46:	i<length(b)	i<n, n=length(a) ⊢ i<length(a) 24,20
47:	i<length(a)	i<n, n=length(a) ⊢ i<length(a) 24,19
48:	n=length(a)∧n=length(b)∧0≤n∧0≤i+1∧i+1≤n∧∀j.((0≤j∧j<i+1)→((b⊕i↦a[i])[j]=a[j]))∧0≤i∧i<length(b)∧i<length(a)	∧ intro 19,20,21,25,26,45,22,46,47

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).