Logic of Programs

Robert Keller
March 2009

Programs with Proofs

- For many reasons, it is desirable to accompany programs with a **proof** that the program meets a certain specification.

- One way to do this is to derive the proof along with the program.

Alan Turing

- Turing may have been the first to consider proving that a program is correct, in his 1950 paper:

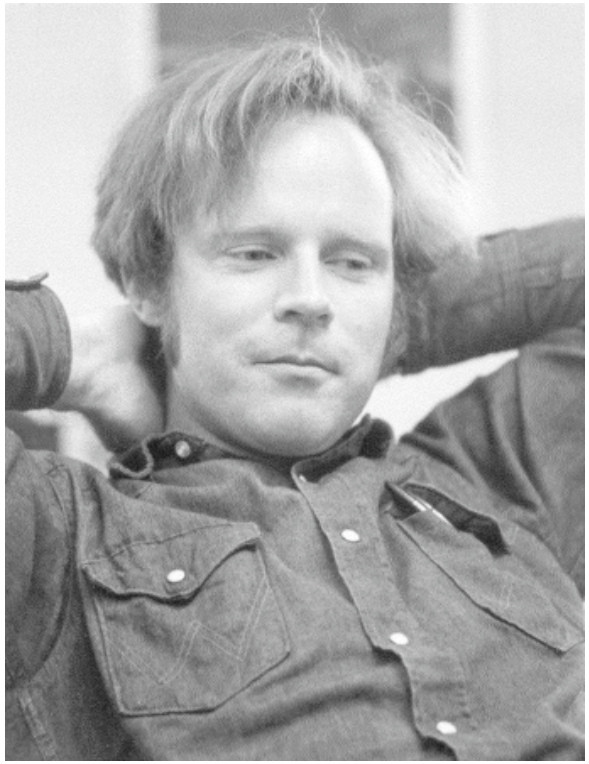
  "Checking a Large Routine"
  (3 pages)

  "How can one check a large routine in the sense that it's right?

  … make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily follows."

Robert W. Floyd

- “Assigning meanings to programs”, 1967

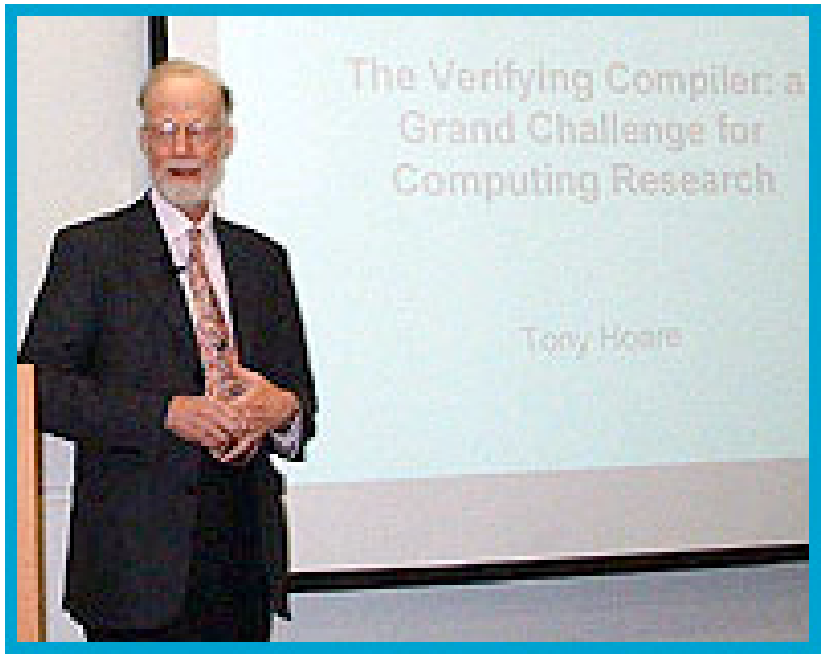


Hoare Logic

- C.A.R. ("Tony") Hoare was the first to express program construction along with proofs of correctness as a single **unified logic**.
- "An axiomatic basis for computer programming", 1969.

D2 Rule of Composition

If $\vdash P\{Q_1\}R_1$ and $\vdash R_1\{Q_2\}R$ then $\vdash P\{(Q_1\,;\,Q_2)\}R$



Sir Prof. Tony Hoare (FRS)
Microsoft Research Laboratory, Cambridge

Program "Dynamics"

- You may be accustomed to thinking of a program as something with "dynamic" behavior.

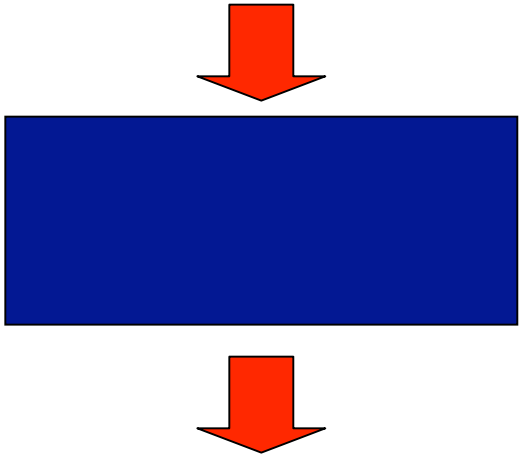
- The *mathematical* view is that a program's behavior is just one of many slices of a (generally-infinite) *static* structure, which can be analyzed with mathematics.

Programs States

- Programs work with **states**.
- Each state is a mapping from program variables into appropriate domains.
- A state is much like an *assignment* part of an interpretation our discussion of semantics of predicate calculus.

Program as a State Transformer

starting state



ending state

Note about I/O

- To deal with input streams and files, we will consider the entire file or stream, along with the current position of the reader or writer, to be part of the state.

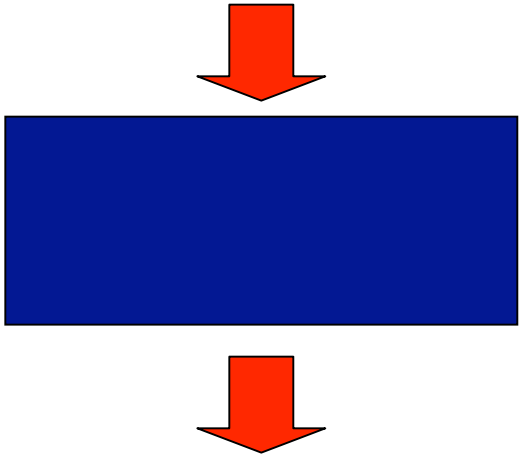
Programs with added Assertions

- An **assertion** is a predicate-logic expression about the variables in the program.

- Assertions can express two kinds of things:
  - An **assumption** about the state (also called the "pre-condition").
  - An **expectation** about the state (also called the "post-condition").

A **Program Specification** consists of

assumption about the starting state



expectation about the ending state

What ifs

- What if the assumption about the starting state doesn’t hold?
  - We don’t care about the result in this case.
  - However, the assumption can be made very stringent, e.g. T, in which case we will always care.

What ifs

- What if the assumption about the starting state holds, but the expectation doesn’t hold when the program terminates?
  - The program is incorrect.

Logic Triples

- Consider endowing a program to be designed with its assumption and expectation:

  {assumption} code {expectation}

- This is known as a “triple”, or “Hoare triple”.

- Originally Hoare put the braces around the code, instead of around the assertions.

Example of a Triple

{assumption} code {expectation}

$\{x \leq y \wedge x \leq z\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

Design then becomes the process of filling in the TBD.

Some triples are more stringent than others

{assumption} code {expectation}

$\{x \leq y \wedge x \leq z\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

$\{x \leq y\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

$\{T\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

$\{T\}$ *TBD* $\{x \leq y \wedge y \leq z \wedge z \leq w\}$

Ways of Using Logic

- **Formal Verification**: Create a program that is **proved** to meet its specification.
- **Model Checking**: Mechanically check that a program meets its specification (used for finite-state systems).
- **Static Analysis**: Symbolically check that no erroneous things are being done by the program (incomplete, but useful).
- **Program Synthesis**: Automate the construction of a program from a specification.

Composition of Triples

- Suppose we have a triple:
  {Assumption} Code {Expectation}

- To develop the code, we can break it into two parts:
  {Assumption 1} Code 1 {Expectation 1}
  {Assumption 2} Code 2 {Expectation 2}

  We want Code = Code 1; Code 2.

  What do we need for this to work?

Composition Rule

We need Expectation1 = Assumption2.

In the form of a natural deduction rule:

$$\frac{\{A\}\ S1\ \{B\} \qquad\qquad \{B\}\ S2\ \{C\}}{\{A\}\ S1;S2\ \{C\}}$$

Example of Composition Rule

1. $\{T\}\ S1\ \{x \leq y\}$
2. $\{x \leq y\}\ S2\ \{x \leq y \wedge y \leq z\}$
3. $\{T\}\ S1;\ S2\ \{x \leq y \wedge y \leq z\}$ Comp. 1, 2

What if Conditions don’t Match

- Sometimes we need to compose segments of code, but the expectation of the first doesn’t match the assumption of the second.
- The next best thing is for it the expectation to **imply** the assumption.
- We can express this by allowing the assumption to be strengthened or the expectation to be weakened, **without changing the code**.

Expectation Weakening Rule

{Assumption} Code {Expectation}

Expectation → Expectation’

{Assumption} Code {Expectation’}

Note: The weakest possible expectation is _____.

Assumption Strengthening Rule

{Assumption} Code {Expectation}

Assumption’ → Assumption (logical implication)

{Assumption’} Code {Expectation}

Note: The strongest possible assumption is _____.

Example of Weakening/Strengthening

1. $\{T\}\ S1\ \{x < y\}$

2. $\{x \leq y\}\ S2\ \{x \leq y \wedge y \leq z\}$

3. To compose these we can either use the assumption strengthening rule to get:

   $\{x < y\}\ S2\ \{x \leq y \wedge y \leq z\}$ since $x < y \rightarrow x \leq y$

4. or we could use the expectation weakening rule to get:

   $\{T\}\ S1\ \{x \leq y\}$

Generalized Composition Rule

$$\frac{\{A\}\ S1\ \{B\} \qquad B \rightarrow C \qquad \{C\}\ S2\ \{D\}}{\{A\}\ S1;S2\ \{D\}}$$

This avoids the introduction of extra steps by the strengthening and weakening rules.

Conditional Rule

$$\frac{\{A \wedge P\}\ S1\ \{B\} \qquad \{A \wedge \neg P\}\ S2\ \{B\}}{\{A\}\ \textbf{if(}\ P\ \textbf{)}\ S1\ \textbf{else}\ S2\ \{B\}}$$

There is a strong resemblance to $\vee$–Elimination.

Example of Conditional Rule

1. $\{x \leq y \wedge (y > z)\}$ *S1* $\{x \leq y \wedge y \leq z\}$
2. $\{x \leq y \wedge \neg(y > z)\}$ *S2* $\{x \leq y \wedge y \leq z\}$
3. $\{x \leq y\}$

   if(y > z) S1 **else** S2

   $\{x \leq y \wedge y \leq z\}$ Cond. 1, 2

One-Sided Conditional Rule

$\{A \wedge P\}\ S1\ \{B\} \qquad (A \wedge \neg P) \rightarrow B$

$\{A\}$ **if(** P **)** $S1\ \{B\}$

Example of One-Sided Conditional Rule

1. $\{x \leq y \wedge y > z\}\ S1\ \{x \leq y \wedge y \leq z\}$
2. $((x \leq y) \wedge \neg(y > z)) \rightarrow (x \leq y \wedge y \leq z)$
3. $\{x \leq y\}$

   if(y > z) S1

   $\{x \leq y \wedge y \leq z\}$ One-SidedCond. 1, 2

While Rule

$$\frac{\{I \wedge P\}\ S\ \{I\}}{\{I\}\ \textbf{while(P) S}\ \{I \wedge \neg P\}}$$

I is known as the “loop invariant”

Example of While Rule

1. $\{x \leq y \wedge y \geq z\}\ S\ \{x \leq y\}$

2. $\{x \leq y\}$

   while$(y \geq z)$ S

   $\{x \leq y \wedge \neg(y \geq z)\}$ While 1

Assignment Statement Rule

$$\frac{}{\{A[\varepsilon/\nu]\} \quad \nu := \varepsilon \quad \{A\}}$$

ν is a variable, an ε expression.

As in predicate logic, $A[\varepsilon/\nu]$ denotes the result of replacing free occurrences of variable ν in A with ε.

(This rule has an **empty** antecedent.)

"Assignment" here should not be confused with assignment as in the interpretation of logic formulas.

Those assignments are like program states.

Example of Assignment Rule

$$\{A[\varepsilon/\nu]\} \quad \nu := \varepsilon \quad \{A\}$$

1. $\{x \leq z\}$ **y := z** $\{x \leq y\}$ Assignment

Here ν is identified with y

ε is identified with z

It is easiest to "work backward" from the expectation.

More Examples of Assignment Rule

$$\{A[\varepsilon/\nu]\} \quad \nu := \varepsilon \quad \{A\}$$

1. $\{x \leq y+1\}$ **y := y+1** $\{x \leq y\}$ Assignment
2. $\{x*y \leq n\}$ **y := x*y** $\{y \leq n\}$ Assignment
3. $\{x+1 \leq n+1\}$ **x := x+1** $\{x \leq n+1\}$ Assignment

Examples of Derivations of Small Programs: **Exchange Program**

To derive: A program that exchanges the values in variables x and y.

$$\{x = x_0 \wedge y = y_0\}\ \textbf{z := x; x := y; y := z;}\ \{y = x_0 \wedge x = y_0\}$$

1. $\{z = x_0 \wedge x = y_0\}\ \textbf{y := z;}\ \{y = x_0 \wedge x = y_0\}$ Assignment
2. $\{z = x_0 \wedge y = y_0\}\ \textbf{x := y;}\ \{z = x_0 \wedge x = y_0\}$ Assignment
3. $\{x = x_0 \wedge y = y_0\}\ \textbf{z := x;}\ \{z = x_0 \wedge y = y_0\}$ Assignment
4. $\{z = x_0 \wedge y = y_0\}\ \textbf{x := y; y := z;}\ \{y = x_0 \wedge x = y_0\}$ Comp 2, 1
5. $\{x = x_0 \wedge y = y_0\}\ \textbf{z := x; x := y; y := z;}\ \{y = x_0 \wedge x = y_0\}$ Comp 3, 4

Examples of Derivations of Small Programs: Ordering two numbers

- $\{x = x_0 \wedge y = y_0\}$

  if(x > y) {z := x; x := y; y := z;}

  $\{x \leq y \wedge ((x = x_0 \wedge y = y_0) \vee (y = x_0 \wedge x = y_0))\}$

- We'll obviously be needing the 1-sided conditional rule.
- We'll assume some things about the < and $\leq$ predicates:
- $\neg(x > y) \rightarrow (x \leq y)$
- $(y > x) \rightarrow (x \leq y)$
- Similar to the derivation on the previous page, we can derive:
- $\{x > y \wedge x = x_0 \wedge y = y_0\}$
  z := x; x := y; y := z;
  $\{y > x \wedge y = x_0 \wedge x = y_0\}$
  and using expectation weakening, we can replace the expectation with
  $\{x \leq y \wedge y = x_0 \wedge x = y_0\}$
- Then identify P in the 1-sided cond rule as: x > y

Examples of Derivations of Small Programs

- $\{x \leq n\}$ **while(x < n) x := x+1** $\{x = n\}$

- We can use the while rule, provided that we can rely on properties of **integer** arithmetic such as:

$$(x < n) \rightarrow ((x+1) \leq n)$$

$$((x \leq n) \wedge \neg(x < n)) \equiv (x = n)$$

Examples of Derivations of Small Programs

1. $((x \leq n) \wedge \neg(x < n)) \equiv (x = n)$ Premise
2. $(x < n) \rightarrow ((x+1) \leq n)$ Premise
3. $\{x+1 \leq n\}$ **x := x+1** $\{x \leq n\}$ Assignment
4. $\{x < n\}$ **x := x+1** $\{x \leq n\}$ Assumption strengthening 3, 2
5. $\{x \leq n \wedge x < n\}$ **x := x+1** $\{x \leq n\}$ Assumption strengthening 4
6. $\{x \leq n\}$ while($x < n$) **x := x+1** $\{x \leq n \wedge \neg(x < n)\}$ While 4
7. $\{x \leq n\}$ while($x < n$) **x := x+1** $\{x = n\}$ Expectation weakening 6

Another Viewpoint: **Verification Conditions**

- An alternate, less formal, way to view a triple, such as:

  $\{x+1 \leq n\}$ **x := x+1** $\{x \leq n\}$

- Think of the assignment in terms of primed (after) and unprimed values:

  $x' = x + 1$ (mathematical equality)

  Then what we are proving is the following **verification condition**:

  $$(x+1 \leq n \wedge (x' = x + 1)) \rightarrow (x' \leq n)$$

  Proving the program reduces to proving a set of verification conditions, one for each transition in the program. Once the VC's are constructed, the program can be forgotten. This was **Floyd's method**.

Using JAPE

- JAPE's theory Hoare logic contains rules similar to what we have described, in addition to:
  - natural deduction
  - rules for dealing with equalities and inequalities.

- It is not complete, although very usable for instruction.

JAPE Hoare Logic Rules

Program

skip
tilt
sequence
Ntuple
variable-assignment
array-element-assignment
choice
while

consequence(L)
consequence(R)

Extra

A=A

A = ..
.. = B

obviously

boundedness from (in)equality

Array Indexing

FROM E=G INFER (A⊕E↦F)[G]=F

FROM E≠G INFER (A⊕E↦F)[G]=A[G]

A = … … = B

Comparison (bi-directional)

A=B≜B=A

A=B≜¬(A≠B)

A≠B≜B≠A

A≠B≜¬(A=B)

A<B≜B>A

A≤B≜A<B∨A=B

A≤B≜B≥A

A≤B≜¬(A>B)

A≤B≜A<B+1

A+1≤B≜A<B

A≥B≜¬(A<B)

A≥B≜A>B-1

A-1≥B≜A>B

A ≜ … … ≜ B

JAPE Hoare Logic ND Rules

Backward

↔ intro
∧ intro (all at once)
∧ intro (one step)
→ intro (makes assumption)
∨ intro (preserving left)
∨ intro (preserving right)
¬ intro (makes assumption A)
∀ intro (introduces variable)
∃ intro (needs formula)
truth

---

contra (classical; makes assumption ¬A)
contra (constructive)
¬ elim (invents formulae)

---

hyp

Forward

∧ elim (all at once)
∧ elim (preserving left)
∧ elim (preserving right)
→ elim
∨ elim (makes assumptions)
¬ elim
∀ elim (needs formula)
∃ elim (assumption & variable)
contra (constructive)

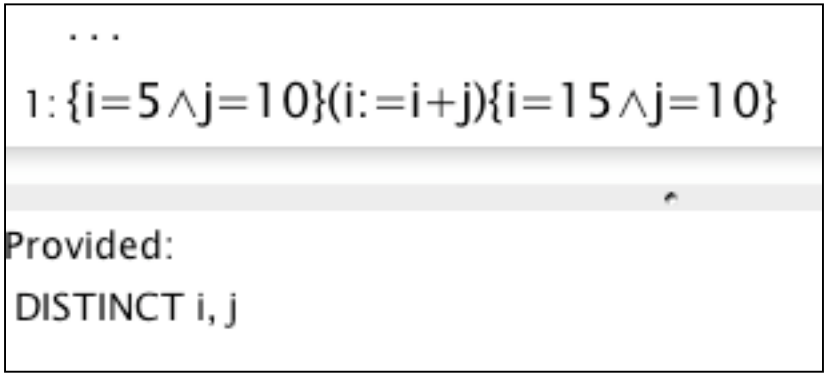
∧ intro
∨ intro (invents right)
∨ intro (invents left)

---

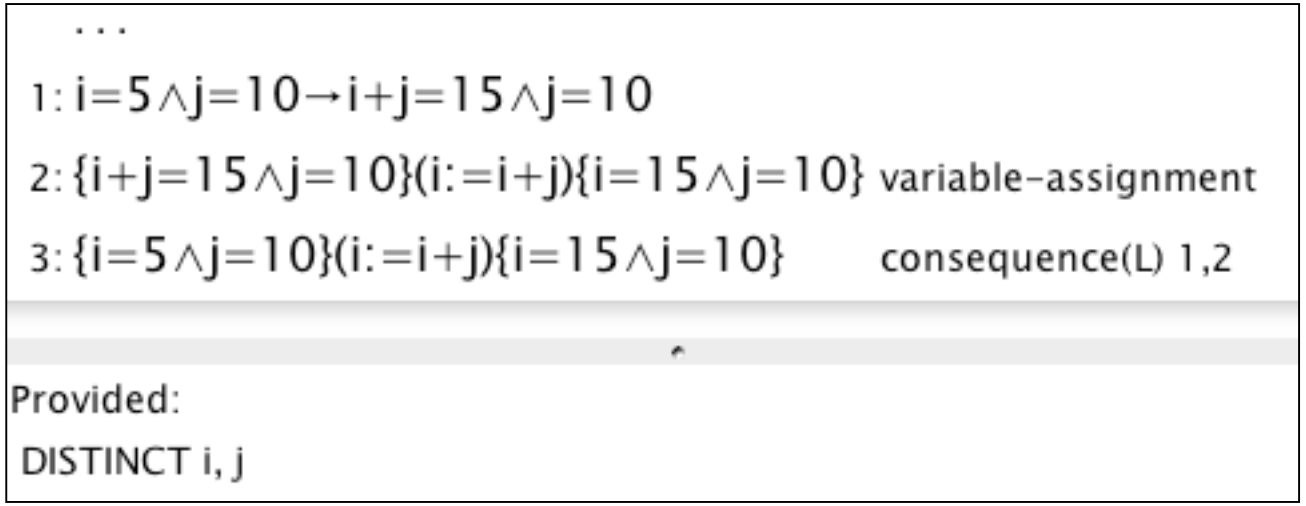
hyp

JAPE HL Examples

- Triple to be proved
  (We will discuss the DISTINCT issue in a bit.)

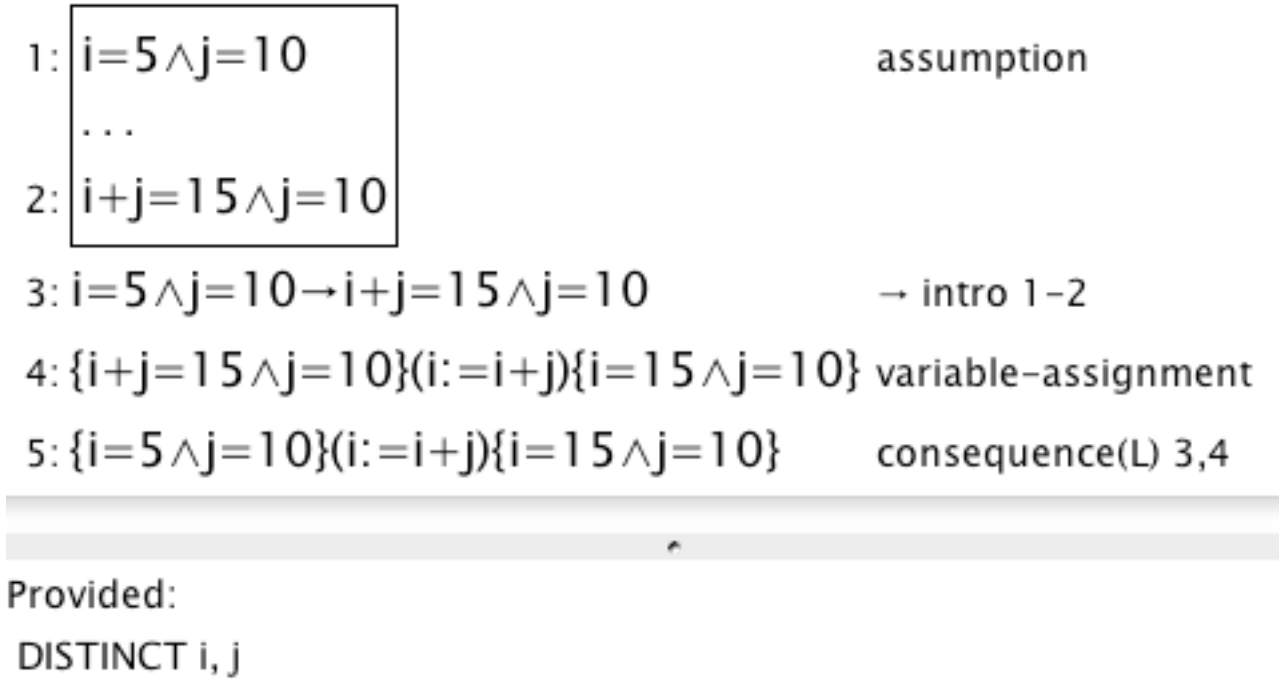


Applying the Variable-Assignment Rule

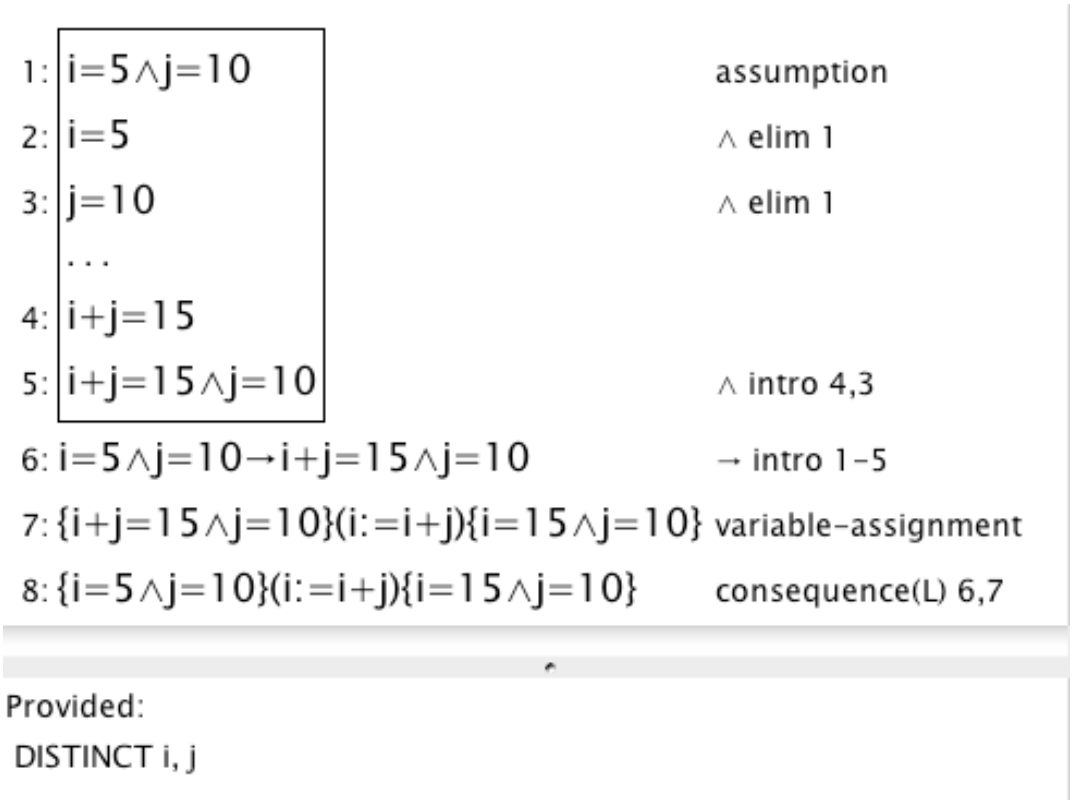


Now the program aspect is done; pure logic remains

- Using →E

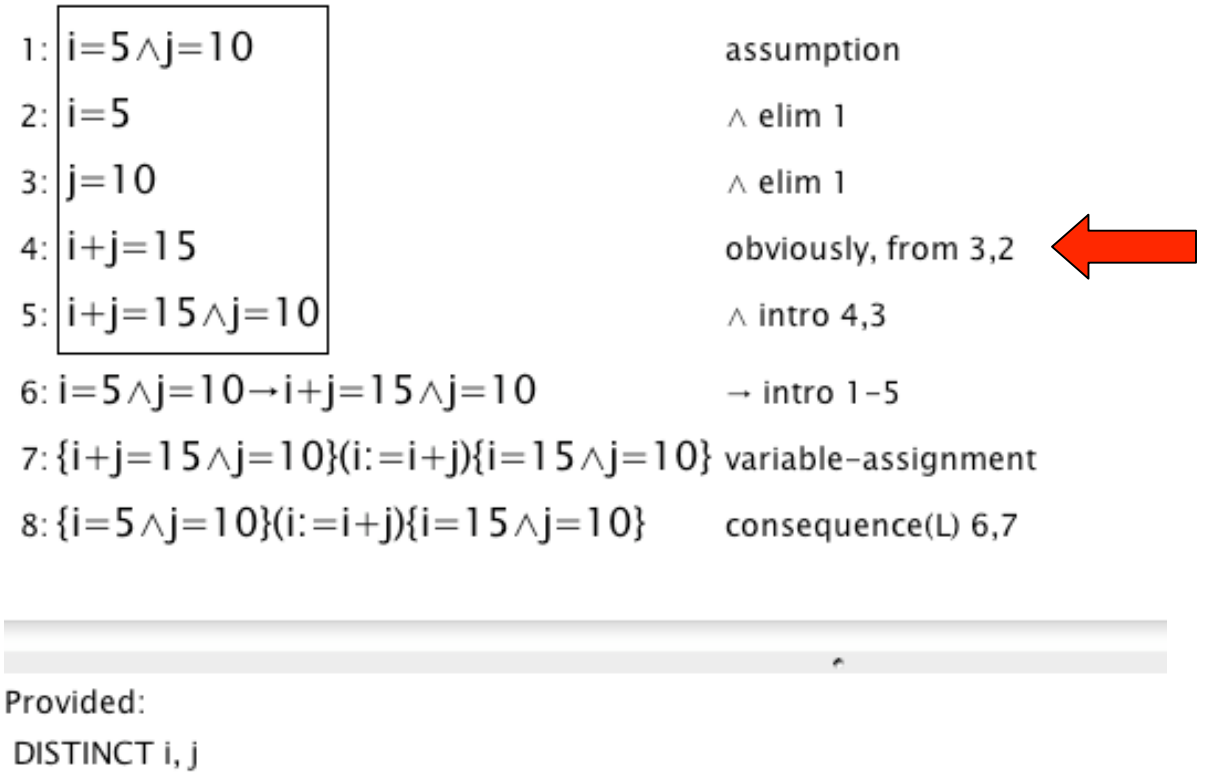


- Using $\wedge I$ and $\wedge E$



Full Arithmetic is Not Available

- So we "cheat"…

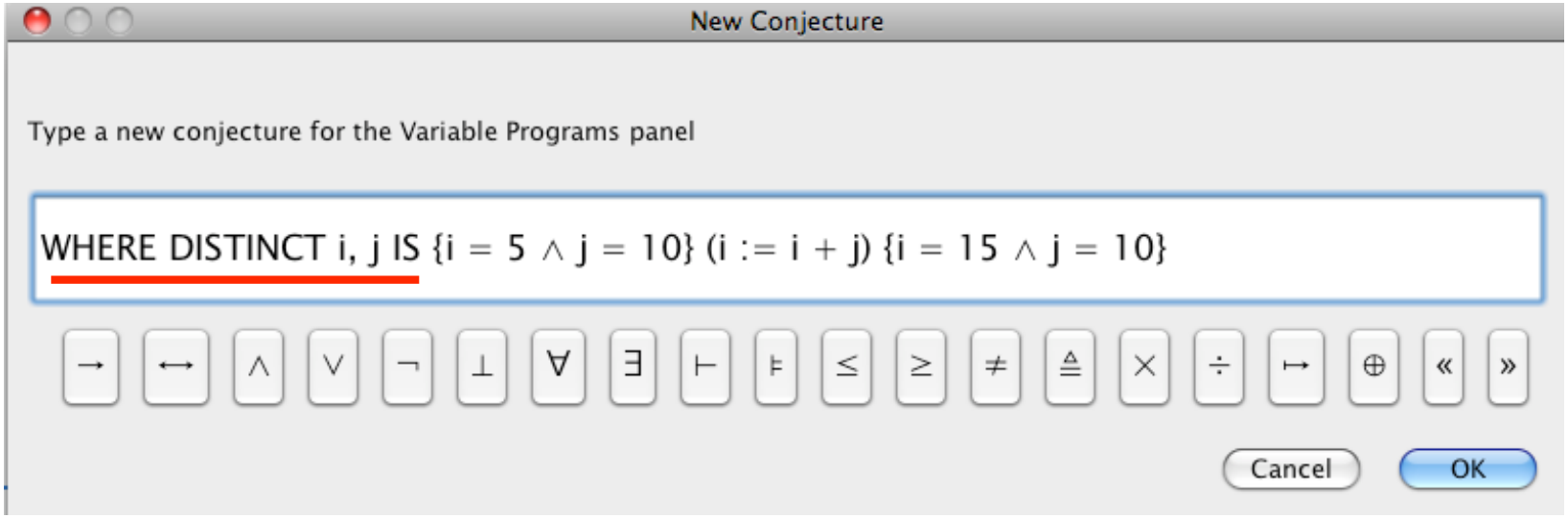


What is the PROVIDED … thing?

- HL rules are sound only if the LHS of an assignment statement is not aliased to another variable.
- JAPE observes this requirement.
- The proviso states this as an assumption.
- Without the proviso, substitutions will be messy.

How to add your own Provisos

- Prefix triple with the WHERE DISTINCT … IS at the time of creation:



Without Proviso

- You get a mess.
- Here « i+j / i » means the result of substituting i +j for free occurrences of i.

$$
\begin{array}{lll}
\ldots & & \\
1: & i=5 \wedge j=10 \rightarrow i+j=15 \wedge j\langle\langle i+j/i\rangle\rangle=10 & \\
2: & \{i+j=15 \wedge j\langle\langle i+j/i\rangle\rangle=10\} & \text{variable-assignment} \\
 & (i:=i+j)\{i=15 \wedge j=10\} & \\
3: & \{i=5 \wedge j=10\}(i:=i+j)\{i=15 \wedge j=10\} & \text{consequence(L) 1,2}
\end{array}
$$

Using the **while** rule

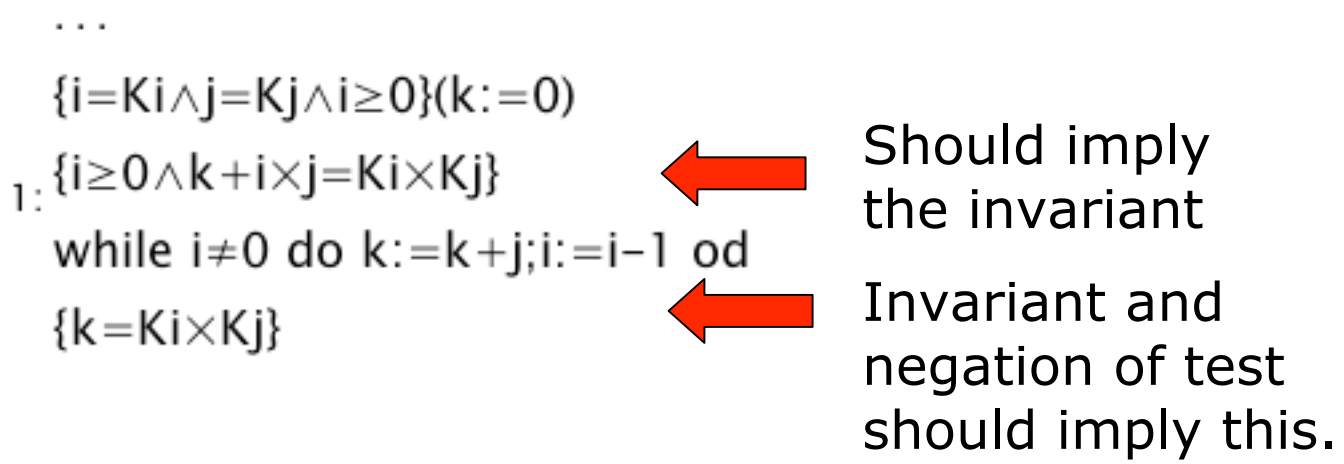
- In order to use the while rule, it is necessary to supply an invariant.

$$\{I \wedge P\}\ S\ \{I\}$$

$$\{I\}\ \textbf{while(P) S}\ \{I \wedge \neg P\}$$

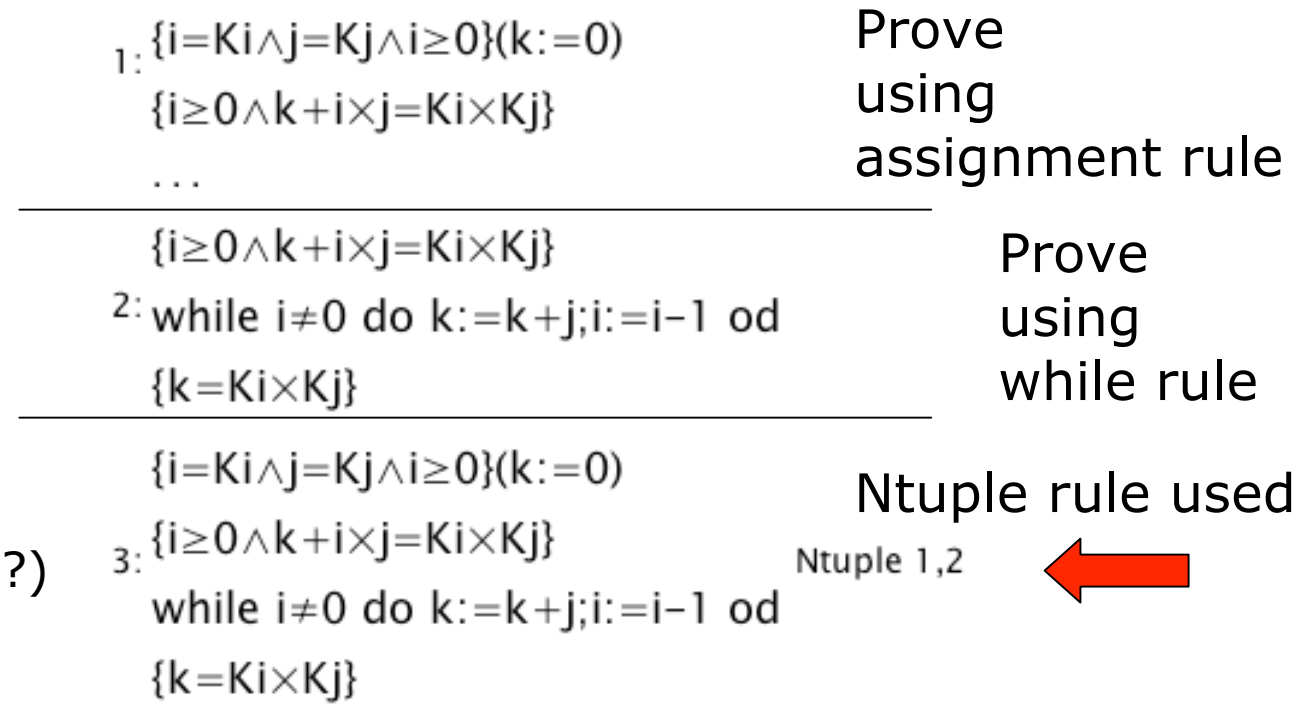
Using the **while** rule

- In JAPE, an assertion implying the invariant is included as an assertion before the while, and also serves as a post-condition for the preceding statement.



Using the **while** rule

- Before using Japе's while rule, this setup is decomposed using Japе's **Ntuple** rule.



(What does this code do?)

The actual while proof after Ntuple

- As Jape incorporates termination into such proofs, it introduces **two** antecedent triples, one for partial correctness and one for termination. First we show the one for partial correctness:

invariant test

2: {i≥0∧k+i×j=Ki×Kj∧i≠0}
(k:=k+j;i:=i−1){i≥0∧k+i×j=Ki×Kj}

invariant

Antecedent of the while rule.

Consequent of the while rule.

6: {i≥0∧k+i×j=Ki×Kj}while i≠0 do k:=k+j;i:=i−1 od while 2,3,4–5
{i≥0∧k+i×j=Ki×Kj∧¬(i≠0)}

invariant negation of test

Proof of the Antecedent (Body) Triple

Proved next page.

1:	$\{i=Ki \wedge j=Kj \wedge i \geq 0\}(k:=0)\{i \geq 0 \wedge k+i \times j=Ki \times Kj\}$	
2:	$i \geq 0 \wedge k+i \times j=Ki \times Kj \wedge i \neq 0$	assumption
3:	$i \geq 0$	$\wedge$ elim 2
4:	$k+i \times j=Ki \times Kj$	$\wedge$ elim 2
5:	$i \neq 0$	$\wedge$ elim 2
6:	$i-1 \geq 0$	obviously, from 5,3
7:	$k+j+(i-1) \times j=Ki \times Kj$	obviously, from 4
8:	$i-1 \geq 0 \wedge k+j+(i-1) \times j=Ki \times Kj$	$\wedge$ intro 6,7
9:	$i \geq 0 \wedge k+i \times j=Ki \times Kj \wedge i \neq 0 \rightarrow i-1 \geq 0 \wedge k+j+(i-1) \times j=Ki \times Kj$	$\rightarrow$ intro 2–8
10:	$\{i-1 \geq 0 \wedge k+j+(i-1) \times j=Ki \times Kj\}(k:=k+j)\{i-1 \geq 0 \wedge k+(i-1) \times j=Ki \times Kj\}$	variable-assignment
11:	$\{i \geq 0 \wedge k+i \times j=Ki \times Kj \wedge i \neq 0\}(k:=k+j)\{i-1 \geq 0 \wedge k+(i-1) \times j=Ki \times Kj\}$	consequence(L) 9,10
12:	$\{i-1 \geq 0 \wedge k+(i-1) \times j=Ki \times Kj\}(i:=i-1)\{i \geq 0 \wedge k+i \times j=Ki \times Kj\}$	variable-assignment
13:	$\{i \geq 0 \wedge k+i \times j=Ki \times Kj \wedge i \neq 0\}(k:=k+j;i:=i-1)\{i \geq 0 \wedge k+i \times j=Ki \times Kj\}$	sequence 11,12

Body triple

Proof that initialization and overall pre-condition implies the preceding pre-condition

1:	$i=Ki \wedge j=Kj \wedge i \geq 0$	assumption
2:	$i=Ki$	$\wedge$ elim 1
3:	$j=Kj$	$\wedge$ elim 1
4:	$i \geq 0$	$\wedge$ elim 1
5:	$0+i \times j=Ki \times Kj$	obviously, from 3,2
6:	$i \geq 0 \wedge 0+i \times j=Ki \times Kj$	$\wedge$ intro 4,5
7:	$i=Ki \wedge j=Kj \wedge i \geq 0 \rightarrow i \geq 0 \wedge 0+i \times j=Ki \times Kj$	$\rightarrow$ intro 1–6
8:	$\{i \geq 0 \wedge 0+i \times j=Ki \times Kj\}(k:=0)\{i \geq 0 \wedge k+i \times j=Ki \times Kj\}$	variable-assignment
9:	$\{i=Ki \wedge j=Kj \wedge i \geq 0\}(k:=0)\{i \geq 0 \wedge k+i \times j=Ki \times Kj\}$	consequence(L) 7,8

Top of previous page.

Partial vs. Total Correctness

- So far, only dealt with "partial correctness":
  - **If** the assumption is true **and** the program terminates, **then** the expectation will be true.
- Of greater interest is "total correctness":
  - **If** the assumption is true, **then** the program terminates with the expectation being true.

Partial vs. Total Correctness

- Total Correctness =

  Partial Correctness + Termination

How to Prove Termination?

- A program terminates if it progress inexorably to a final state.

- Identify a function μ of state:

  μ: States$\rightarrow$N (Natural Numbers)

  such that on every iteration μ decreases in value.

Handling termination in Jape

- The function μ is implicit.
- Its value is represented by an integer expression _M introduced by the **unify** edit.
- The expression is asserted to be **equal** to a fresh variable Km at the **start** of the of the loop body.
- The **invariant** conjoined with the truth of the loop test is conjoined to the start assertion.
- The expression is asserted to be **less than** that same variable Km at the **end** of the loop body.
- It is asserted that the **invariant** conjoined with truth of the loop test implies the value of _M is positive. (Rationale: Since _M is positive at the start of the loop, and always decreases, it cannot loop forever.)

Termination part of the while example

From while rule: implies loop invariant

For termination: If loop test true, then value must be positive.

3: $i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \rightarrow \_M > 0$ ⟵ _M (to be unified by user)

4: integer Km assumption

5: $\{i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \wedge \_M = Km\}$ (k:=k+j;i:=i−1)$\{\_M < Km\}$

invariant test anchor decrease

6: $\{i \geq 0 \wedge k+i \times j = Ki \times Kj\}$while i≠0 do k:=k+j;i:=i−1 od $\{i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge \neg(i \neq 0)\}$ while 2,3,4–5

22: $i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \rightarrow \_M > 0$ ⟵ See last bullet, previous page.

Proof with all parts completed

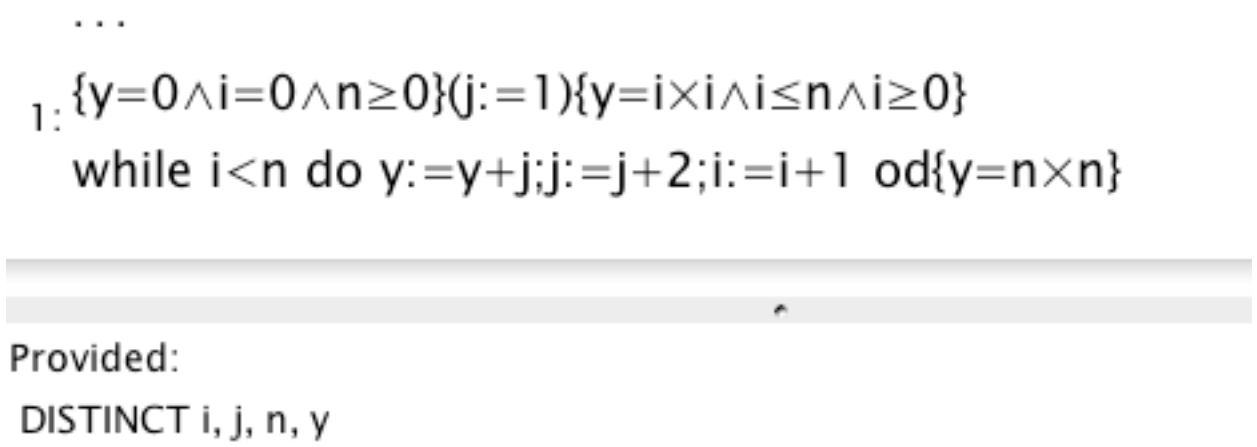
26:	$i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \rightarrow i > 0$	→ intro 22–25
27:	integer Km	assumption
28:	$i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \wedge i = Km$	assumption
29:	$i \geq 0$	∧ elim 28
30:	$i = Km$	∧ elim 28
31:	$i-1 < Km$	obviously, from 30,29
32:	$i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \wedge i = Km \rightarrow i-1 < Km$	→ intro 28–31
33:	$\{i-1 < Km\}(k := k+j)\{i-1 < Km\}$	variable-assignment
34:	$\{i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \wedge i = Km\}(k := k+j)\{i-1 < Km\}$	consequence(L) 32,33
35:	$\{i-1 < Km\}(i := i-1)\{i < Km\}$	variable-assignment
36:	$\{i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge i \neq 0 \wedge i = Km\}(k := k+j; i := i-1)\{i < Km\}$	sequence 34,35
37:	$\{i \geq 0 \wedge k+i \times j = Ki \times Kj\}$while $i \neq 0$ do $k := k+j; i := i-1$ od$\{i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge \neg(i \neq 0)\}$	while 21,26,27–36
38:	$i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge \neg(i \neq 0)$	assumption
39:	$k+i \times j = Ki \times Kj$	∧ elim 38
40:	$\neg(i \neq 0)$	∧ elim 38
41:	$i = 0$	$A=B \triangleq \neg(A \neq B)$ 40
42:	$k = Ki \times Kj$	obviously, from 41,39
43:	$i \geq 0 \wedge k+i \times j = Ki \times Kj \wedge \neg(i \neq 0) \rightarrow k = Ki \times Kj$	→ intro 38–42
44:	$\{i \geq 0 \wedge k+i \times j = Ki \times Kj\}$while $i \neq 0$ do $k := k+j; i := i-1$ od$\{k = Ki \times Kj\}$	consequence(R) 37,43
45:	$\{i = Ki \wedge j = Kj \wedge i \geq 0\}(k := 0)\{i \geq 0 \wedge k+i \times j = Ki \times Kj\}$while $i \neq 0$ do $k := k+j; i := i-1$ od$\{k = Ki \times Kj\}$	Ntuple 9,44

A proof of a simpler program

1:	$i=10 \wedge i>0$	assumption
2:	$i-1=10$	obviously
3:	$i=10 \wedge i>0 \rightarrow i-1=10$	→ intro 1-2
4:	$\{i-1=10\}(i:=i-1)\{i=10\}$	variable-assignment
5:	$\{i=10 \wedge i>0\}(i:=i-1)\{i=10\}$	consequence(L) 3,4
6:	$i=10 \wedge i>0$	assumption
7:	$i>0$	∧ elim 6
8:	$i=10 \wedge i>0 \rightarrow i>0$	→ intro 6-7
9:	integer Km	assumption
10:	$i=10 \wedge i>0 \wedge i=Km$	assumption
11:	$i-1<Km$	obviously
12:	$i=10 \wedge i>0 \wedge i=Km \rightarrow i-1<Km$	→ intro 10-11
13:	$\{i-1<Km\}(i:=i-1)\{i<Km\}$	variable-assignment
14:	$\{i=10 \wedge i>0 \wedge i=Km\}(i:=i-1)\{i<Km\}$	consequence(L) 12,13
15:	$\{i=10\}$while $i>0$ do $i:=i-1$ od$\{i=10 \wedge \neg(i>0)\}$	while 5,8,9-14
16:	$i=10 \wedge \neg(i>0) \rightarrow i=0$	obviously
17:	$\{i=10\}$(while $i>0$ do $i:=i-1$ od)$\{i=0\}$	consequence(R) 15,16

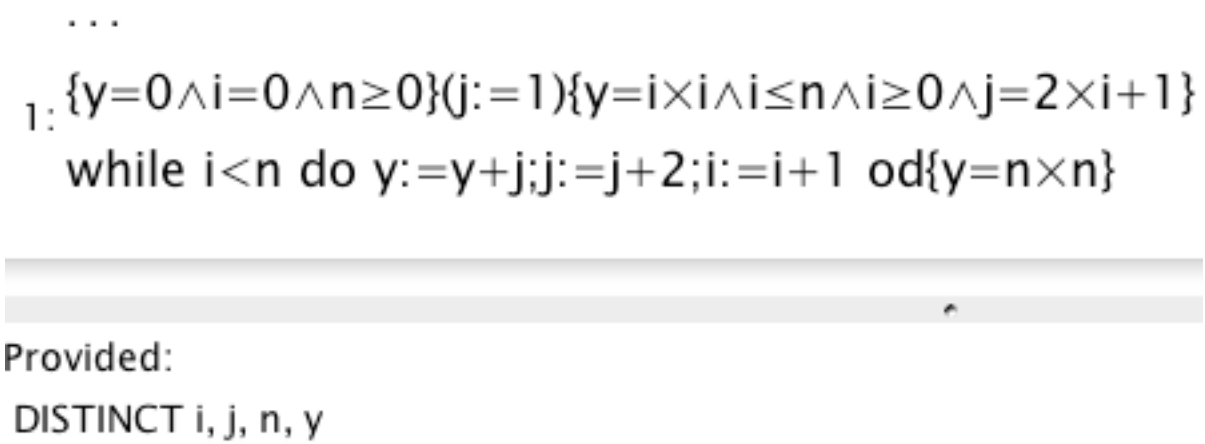
Subtleties about loop invariants

- Can the following be proved?



The loop invariant is not strong enough to enable induction

- This is more likely provable.
- What is _M for termination?



The Completed Proof (lines 1-24)

1:	$y=0 \wedge i=0 \wedge n \geq 0$	assumption
2:	$y=0$	$\wedge$ elim 1
3:	$i=0$	$\wedge$ elim 1
4:	$n \geq 0$	$\wedge$ elim 1
5:	$y=i \times i$	obviously, from 3,2
6:	$i \leq n$	obviously, from 4,3
7:	$i \geq 0$	obviously, from 3
8:	$1=2 \times i+1$	obviously, from 3
9:	$y=i \times i \wedge i \leq n \wedge i \geq 0 \wedge 1=2 \times i+1$	$\wedge$ intro 5,6,7,8
10:	$y=0 \wedge i=0 \wedge n \geq 0 \rightarrow y=i \times i \wedge i \leq n \wedge i \geq 0 \wedge 1=2 \times i+1$	$\rightarrow$ intro 1–9
11:	$\{y=i \times i \wedge i \leq n \wedge i \geq 0 \wedge 1=2 \times i+1\}(j:=1)\{y=i \times i \wedge i \leq n \wedge i \geq 0 \wedge j=2 \times i+1\}$	variable-assignment
12:	$\{y=0 \wedge i=0 \wedge n \geq 0\}(j:=1)\{y=i \times i \wedge i \leq n \wedge i \geq 0 \wedge j=2 \times i+1\}$	consequence(L) 10,11
13:	$y=i \times i \wedge i \leq n \wedge i \geq 0 \wedge j=2 \times i+1 \wedge i<n$	assumption
14:	$y=i \times i$	$\wedge$ elim 13
15:	$i \geq 0$	$\wedge$ 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:	$i+1 \leq n$	obviously, from 17
20:	$i+1 \geq 0$	obviously, from 15
21:	$j+2=2 \times (i+1)+1$	obviously, from 16
22:	$y+j=(i+1) \times (i+1) \wedge i+1 \leq n \wedge i+1 \geq 0 \wedge j+2=2 \times (i+1)+1$	$\wedge$ intro 18,19,20,21
23:	$y=i \times i \wedge i \leq n \wedge i \geq 0 \wedge j=2 \times i+1 \wedge i<n \rightarrow y+j=(i+1) \times (i+1) \wedge i+1 \leq n \wedge i+1 \geq 0 \wedge j+2=2 \times (i+1)+1$	$\rightarrow$ intro 13–22
24:	$\{y+j=(i+1) \times (i+1) \wedge i+1 \leq n \wedge i+1 \geq 0 \wedge j+2=2 \times (i+1)+1\}(y:=y+j)\{y=(i+1) \times (i+1) \wedge i+1 \leq n \wedge i+1 \geq 0 \wedge j+2=2 \times (i+1)+1\}$	variable-assignment

Initialization

First assignment in loop body

The Completed Proof (lines 25-51)

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
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
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
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

Other assignments in loop body

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 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
- Because several indices can have the same value, arrays are more susceptible to variable aliasing, e.g.

i := 5; j = 6-1; a[j] = a[i]+1 ds

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

. . .

1: {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}

Provided:

DISTINCT a, i, j, n

Read-Only Example (lines 1-15)

1:	$n \geq 0 \wedge \text{length}(a) = n$	assumption
2:	$n \geq 0$	$\wedge$ elim 1
3:	$\text{length}(a) = n$	$\wedge$ elim 1
4:	$0 \leq n$	$A \leq B \triangleq B \geq A$ 2
5:	$0 \geq 0$	obviously
6:	$n < n$	assumption
7:	$\bot$	obviously, from 6
8:	$a[n] = 0$	contra (constructive) 7
9:	$n < n \rightarrow a[n] = 0$	$\rightarrow$ intro 6–8
10:	$0 \leq n \wedge 0 \geq 0 \wedge \text{length}(a) = n \wedge (n < n \rightarrow a[n] = 0)$	$\wedge$ intro 4,5,3,9
11:	$n \geq 0 \wedge \text{length}(a) = n \rightarrow 0 \leq n \wedge 0 \geq 0 \wedge \text{length}(a) = n \wedge (n < n \rightarrow a[n] = 0)$	$\rightarrow$ intro 1–10
12:	$\{0 \leq n \wedge 0 \geq 0 \wedge \text{length}(a) = n \wedge (n < n \rightarrow a[n] = 0)\}(i := 0)\{i \leq n \wedge i \geq 0 \wedge \text{length}(a) = n \wedge (n < n \rightarrow a[n] = 0)\}$	variable-assignment
13:	$\{n \geq 0 \wedge \text{length}(a) = n\}(i := 0)\{i \leq n \wedge i \geq 0 \wedge \text{length}(a) = n \wedge (n < n \rightarrow a[n] = 0)\}$	consequence(L) 11,12
14:	$\{i \leq n \wedge i \geq 0 \wedge \text{length}(a) = n \wedge (n < n \rightarrow a[n] = 0)\}(j := n)\{i \leq n \wedge i \geq 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)\}$	variable-assignment
15:	$\{n \geq 0 \wedge \text{length}(a) = n\}(i := 0; j := n)\{i \leq n \wedge i \geq 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)\}$	sequence 13,14

Read-Only Example (lines 16-37)

16:	$i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge i < n$	assumption
17:	$i \ge 0$	$\wedge$ elim 16
18:	$\text{length}(a) = n$	$\wedge$ elim 16
19:	$j < n \rightarrow a[j] = 0$	$\wedge$ elim 16
20:	$i < n$	$\wedge$ elim 16
21:	$a[i] = 0$	assumption
22:	$i+1 \le n$	obviously, from 20
23:	$i+1 \ge 0$	obviously, from 17
24:	$i < n$	assumption
25:	$a[i] = 0$	hyp 21
26:	$i < n \rightarrow a[i] = 0$	$\rightarrow$ intro 24–25
27:	$i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (i < n \rightarrow a[i] = 0)$	$\wedge$ intro 22,23,18,26
28:	$a[i] = 0 \rightarrow i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (i < n \rightarrow a[i] = 0)$	$\rightarrow$ intro 21–27
29:	$\neg(a[i] = 0)$	assumption
30:	$i+1 \le n$	obviously, from 20
31:	$i+1 \ge 0$	obviously, from 17
32:	$i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)$	$\wedge$ intro 30,31,18,19
33:	$\neg(a[i] = 0) \rightarrow i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)$	$\rightarrow$ intro 29–32
34:	$0 \le i$	obviously, from 17
35:	$i < \text{length}(a)$	obviously, from 20,18
36:	$(a[i] = 0 \rightarrow i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (i < n \rightarrow a[i] = 0)) \wedge (\neg(a[i] = 0) \rightarrow i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)) \wedge 0 \le i \wedge i < \text{length}(a)$	$\wedge$ intro 28,33,34,35
37:	$i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge i < n$ $\rightarrow (a[i] = 0 \rightarrow i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (i < n \rightarrow a[i] = 0)) \wedge (\neg(a[i] = 0) \rightarrow i+1 \le n \wedge i+1 \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)) \wedge 0 \le i \wedge i < \text{length}(a)$	$\rightarrow$ intro 16–36

Read-Only Example (lines 38-47)

38:	$\{i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(i<n\rightarrow a[i]=0)\}(j:=i)\{i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}$	variable-assignment
39:	$\{i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}\text{skip}\{i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}$	skip
40:	$\{(a[i]=0\rightarrow i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(i<n\rightarrow a[i]=0))\wedge(\neg(a[i]=0)\rightarrow i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0))\wedge 0\le i\wedge i<\text{length}(a)\}$ if a[i]=0 then j:=i else skip fi$\{i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}$	choice 38,39
41:	$\{i\le n\wedge i\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\wedge i<n\}$if a[i]=0 then j:=i else skip fi$\{i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}$	consequence(L) 37,40
42:	$\{i+1\le n\wedge i+1\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}(i:=i+1)\{i\le n\wedge i\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}$	variable-assignment
43:	$\{i\le n\wedge i\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\wedge i<n\}$(if a[i]=0 then j:=i else skip fi;i:=i+1)$\{i\le n\wedge i\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\}$	sequence 41,42
44:	$i\le n\wedge i\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\wedge i<n$	assumption
45:	$i<n$	$\wedge$ elim 44
46:	$n-i>0$	obviously, from 45
47:	$i\le n\wedge i\ge 0\wedge \text{length}(a)=n\wedge(j<n\rightarrow a[j]=0)\wedge i<n\rightarrow n-i>0$	$\rightarrow$ intro 44–46

Read-Only Example (lines 48-74)

48:	integer Km	assumption
49:	$i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge i < n \wedge n - i = Km$	assumption
50:	$i \ge 0$	$\wedge$ elim 49
51:	$i < n$	$\wedge$ elim 49
52:	$n - i = Km$	$\wedge$ elim 49
53:	$a[i] = 0$	assumption
54:	$n - (i+1) < Km$	obviously, from 52
55:	$a[i] = 0 \rightarrow n - (i+1) < Km$	$\rightarrow$ intro 53–54
56:	$\neg(a[i] = 0)$	assumption
57:	$n - (i+1) < Km$	obviously, from 52
58:	$\neg(a[i] = 0) \rightarrow n - (i+1) < Km$	$\rightarrow$ intro 56–57
59:	$0 \le i$	$A \le B \triangleq B \ge A$ 50
60:	$i < \text{length}(a)$	obviously, from 51
61:	$(a[i] = 0 \rightarrow n - (i+1) < Km) \wedge (\neg(a[i] = 0) \rightarrow n - (i+1) < Km) \wedge 0 \le i \wedge i < \text{length}(a)$	$\wedge$ intro 55,58,59,60
62:	$i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge i < n \wedge n - i = Km \rightarrow (a[i] = 0 \rightarrow n - (i+1) < Km) \wedge (\neg(a[i] = 0) \rightarrow n - (i+1) < Km) \wedge 0 \le i \wedge i < \text{length}(a)$	$\rightarrow$ intro 49–61
63:	$\{n - (i+1) < Km\}(j := i)\{n - (i+1) < Km\}$	variable-assignment
64:	$\{n - (i+1) < Km\}\text{skip}\{n - (i+1) < Km\}$	skip
65:	$\{(a[i] = 0 \rightarrow n - (i+1) < Km) \wedge (\neg(a[i] = 0) \rightarrow n - (i+1) < Km) \wedge 0 \le i \wedge i < \text{length}(a)\}\text{if } a[i] = 0 \text{ then } j := i \text{ else skip fi}\{n - (i+1) < Km\}$	choice 63,64
66:	$\{i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge i < n \wedge n - i = Km\}\text{if } a[i] = 0 \text{ then } j := i \text{ else skip fi}\{n - (i+1) < Km\}$	consequence(L) 62,65
67:	$\{n - (i+1) < Km\}(i := i+1)\{n - i < Km\}$	variable-assignment
68:	$\{i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge i < n \wedge n - i = Km\}(\text{if } a[i] = 0 \text{ then } j := i \text{ else skip fi}; i := i+1)\{n - i < Km\}$	sequence 66,67
69:	$\{i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)\}\text{while } i < n \text{ do if } a[i] = 0 \text{ then } j := i \text{ else skip fi}; i := i+1 \text{ od}\{i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge \neg(i < n)\}$	while 43,47,48–68
70:	$i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge \neg(i < n)$	assumption
71:	$j < n \rightarrow a[j] = 0$	$\wedge$ elim 70
72:	$i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0) \wedge \neg(i < n) \rightarrow j < n \rightarrow a[j] = 0$	$\rightarrow$ intro 70–71
73:	$\{i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)\}\text{while } i < n \text{ do if } a[i] = 0 \text{ then } j := i \text{ else skip fi}; i := i+1 \text{ od}\{j < n \rightarrow a[j] = 0\}$	consequence(R) 69,72
74:	$\{n \ge 0 \wedge \text{length}(a) = n\}(i := 0; j := n)\{i \le n \wedge i \ge 0 \wedge \text{length}(a) = n \wedge (j < n \rightarrow a[j] = 0)\}\text{while } i < n \text{ do if } a[i] = 0 \text{ then } j := i \text{ else skip fi}; i := i+1 \text{ od}\{j < n \rightarrow a[j] = 0\}$	Ntuple 15,73

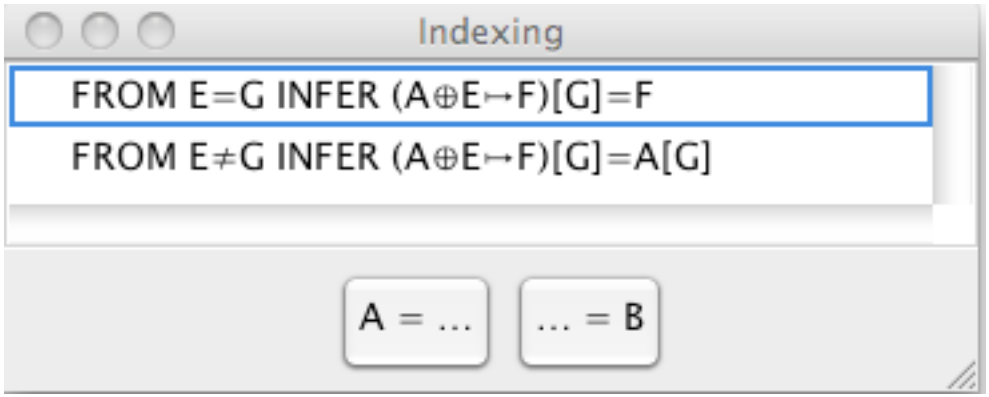
Modifiable Arrays

- 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 Notation: $a \oplus i \rightarrow v$ is the array that is like a except that the value of a[i] is v.
- So $(a \oplus i \rightarrow v)[i] = v$, and
  $(a \oplus i \rightarrow v)[j] = a[j]$ if $j \neq i$.

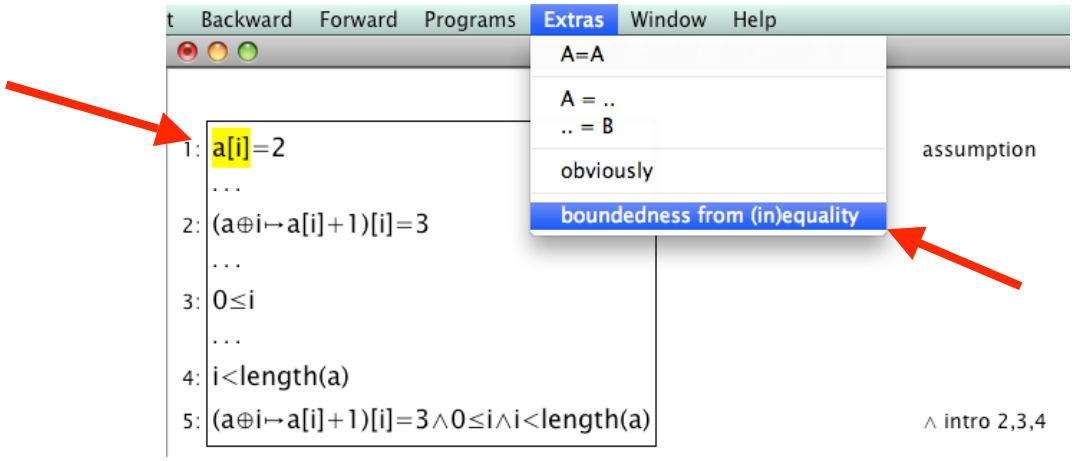
Jape's Indexing rules

- $(a \oplus i \rightarrow v)[i] = v$, and
  $(a \oplus i \rightarrow v)[j] = a[j]$ if $j \neq i$.
- Two rules below capture the two cases preceding.
  - The first rule simplifies an array modification expression when the index of the new array is provably **the same** to which assignment was done.
  - The second rule simplifies in the case of a **different** index.
- The buttons indicate the direction of substitution.

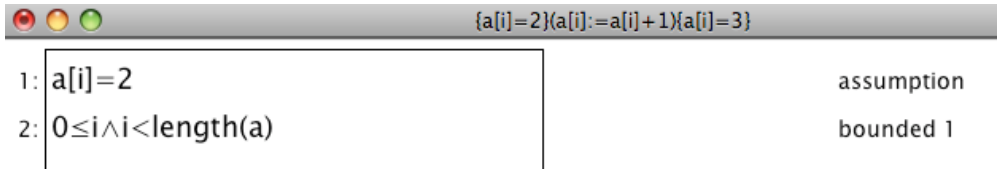


Array Bounds Guarantees

- If an array index value is used in an assumption, the same index value can be used later on without requiring a bounds check.
- Sub-formula select a hypothesis using the desired index.



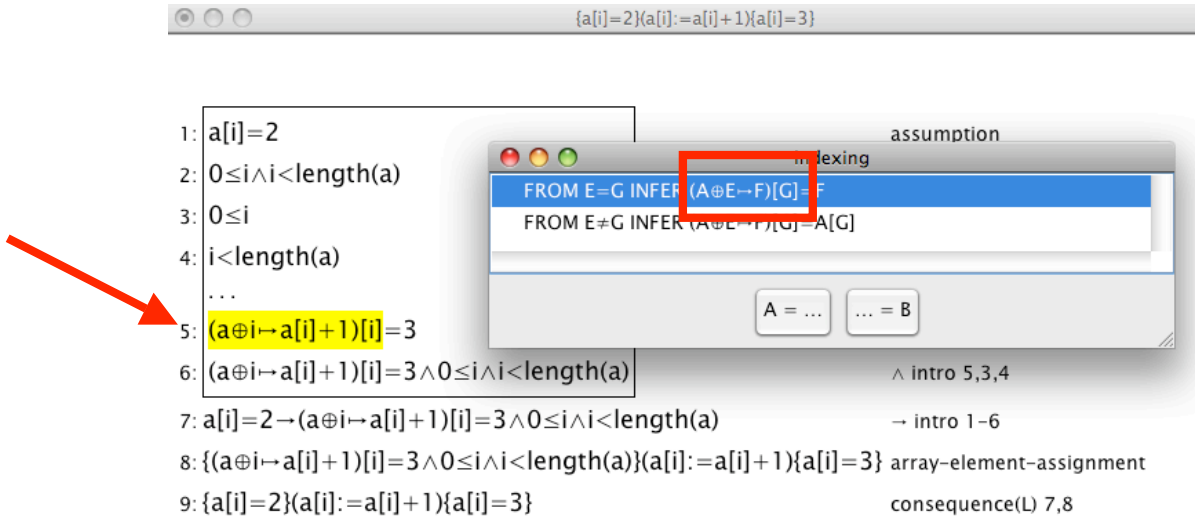
Result:



Using the Array Rule

- Make sure the entire array sub-expression is sub-formula selected.

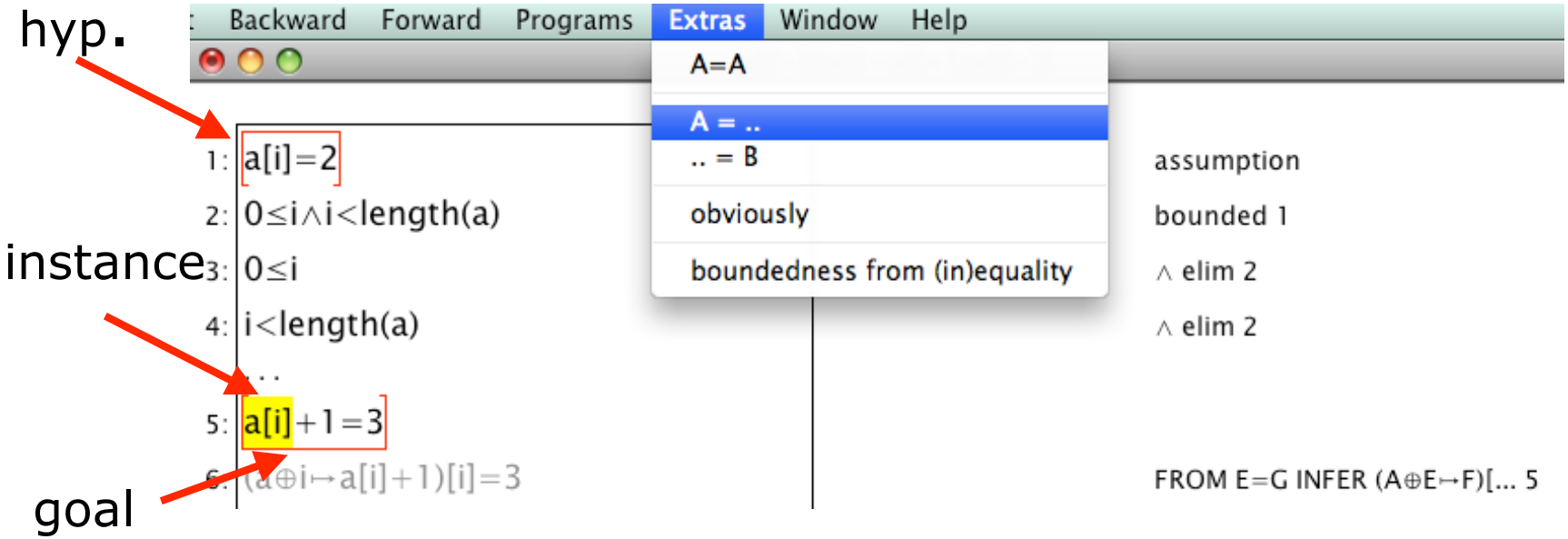
- It should match the form in the rule in the menu:



Here we identify:
A with a
E with i
F with a[i]+1
[G] with [i]
(so E = G).

Using the Equality Rule

- Two selections and a sub-formula selection are needed:
  - Selection an equality hypothesis and a goal.
  - Sub-formula select an instance of the LHS of the equality.

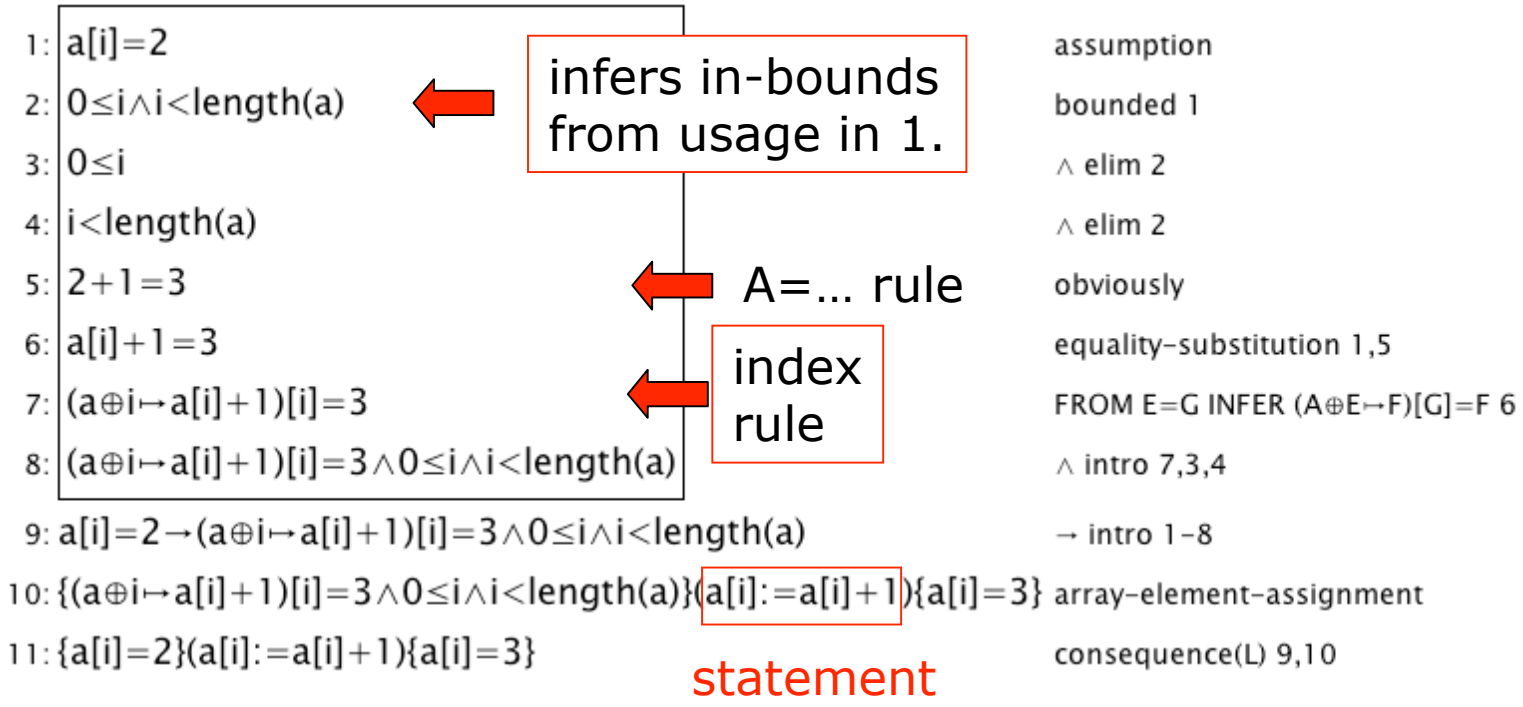


Result of the Equality Rule

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

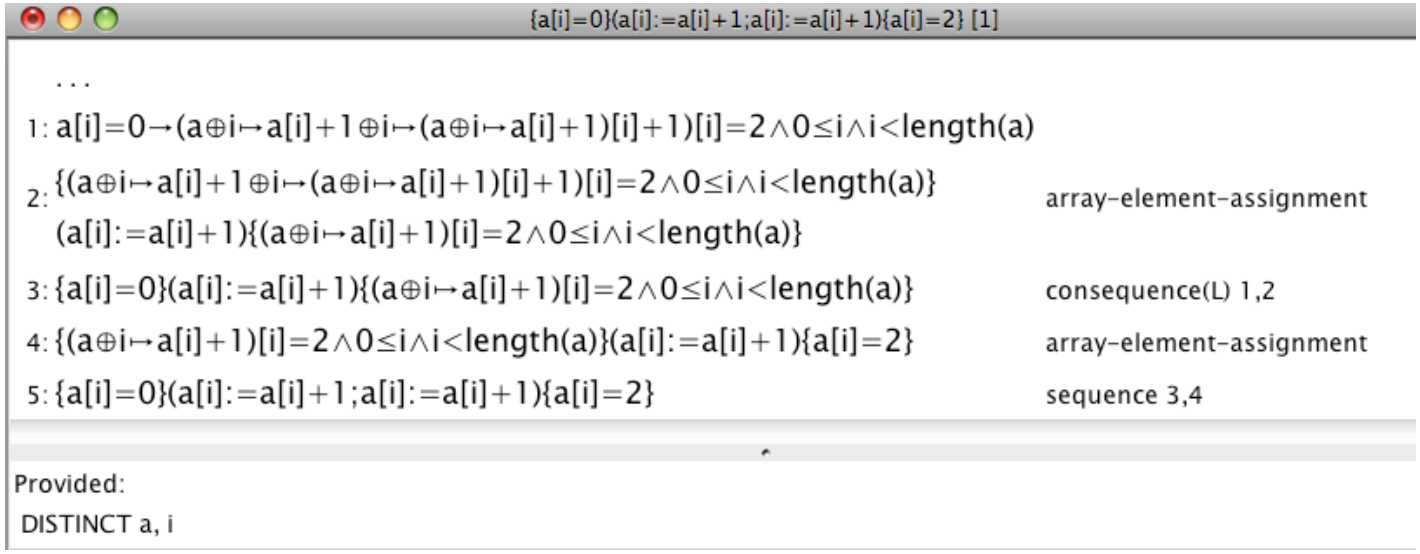
The top simple equation
can be justified by “obviously”.

Summary: Jape proof with array modification



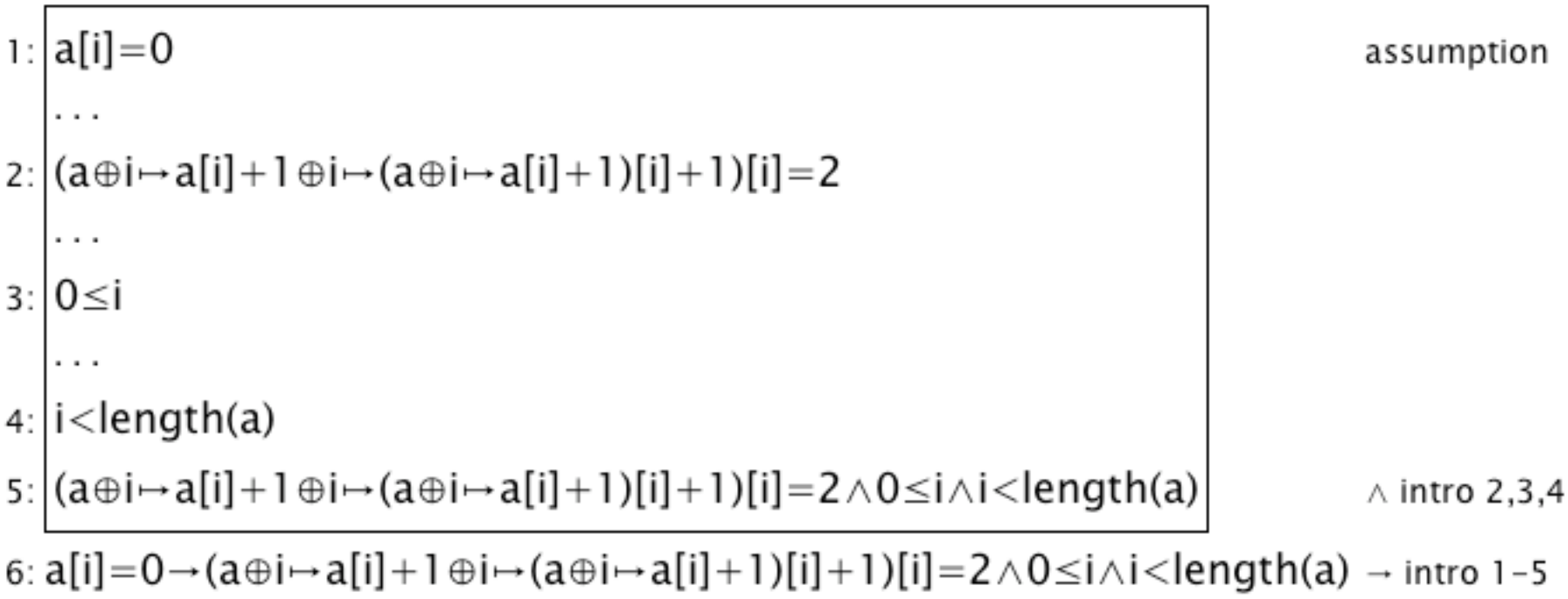
How to get these rules to work in the GUI (It isn't so obvious.)

- Looking at the 2nd provided array program example, we use sequence, then array-assignment twice (from the bottom up) to get to this point:



How to use GUI (continued)

- The top line is pure logic, so we expand using →Introduction and ∧Introduction:



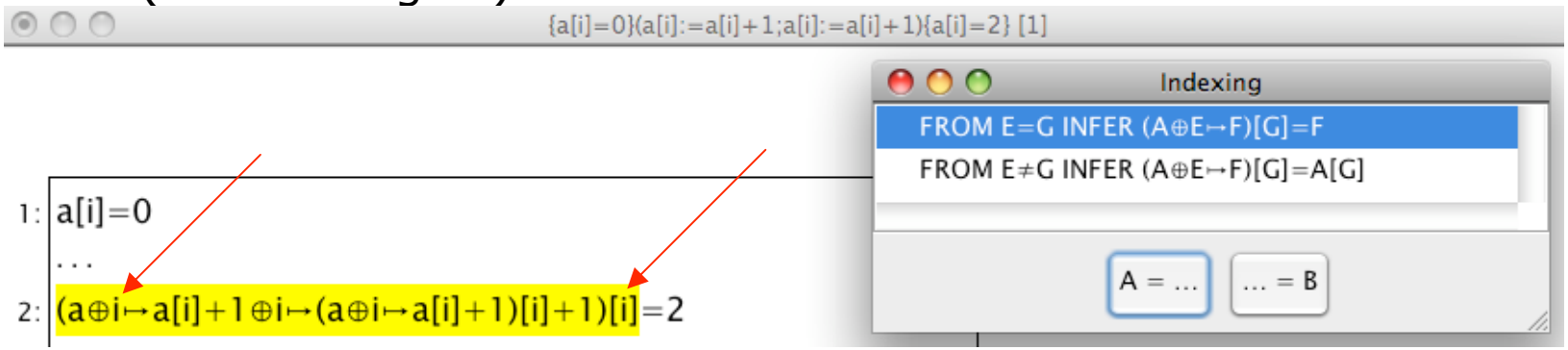
How to use GUI (continued)

- We then conclude the two array index bounds (lines 4, 5) by ∧ Elimination, giving:

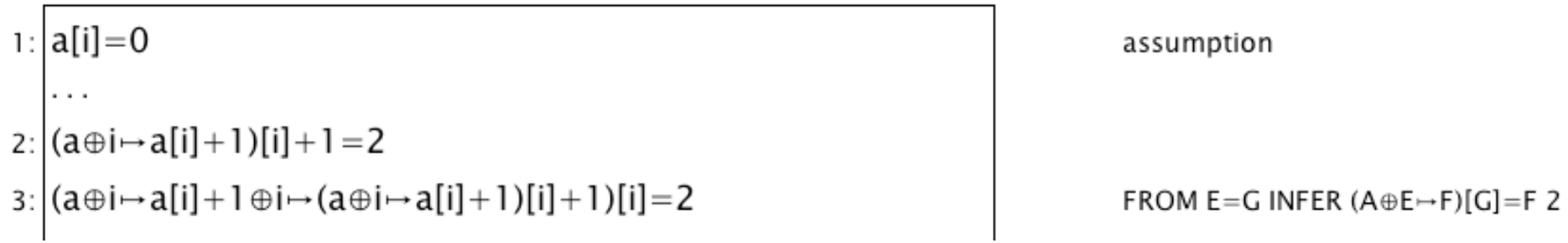
1:	$a[i]=0$	assumption
	...	
2:	$(a \oplus i \mapsto a[i]+1 \oplus i \mapsto (a \oplus i \mapsto a[i]+1)[i]+1)[i]=2$	
3:	$0 \leq i \wedge i < \text{length}(a)$	bounded 1
4:	$0 \leq i$	∧ elim 3
5:	$i < \text{length}(a)$	∧ elim 3
6:	$(a \oplus i \mapsto a[i]+1 \oplus i \mapsto (a \oplus i \mapsto a[i]+1)[i]+1)[i]=2 \wedge 0 \leq i \wedge i < \text{length}(a)$	∧ intro 2,4,5

How to use GUI (continued)

- We are left with a nested array-modification expression. *Carefully* sub-formula select the outer array-modification and apply the rule shown (since we have a⊕i→ …[i]). Do not have anything else (such as a goal) selected.

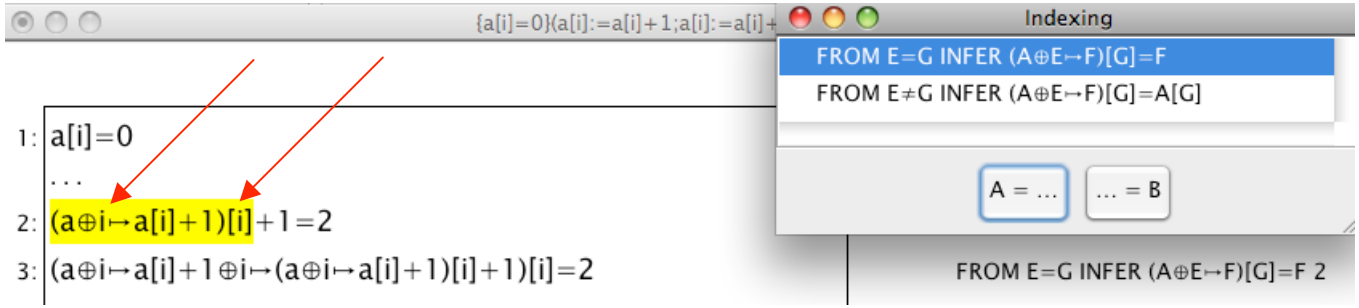


giving:

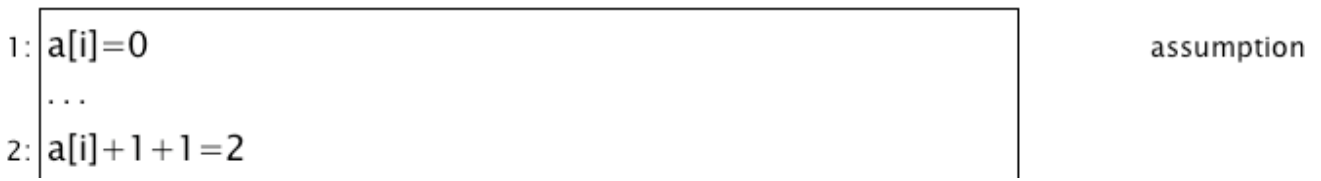


How to use GUI (continued)

- Repeat the preceding process on the new formula:

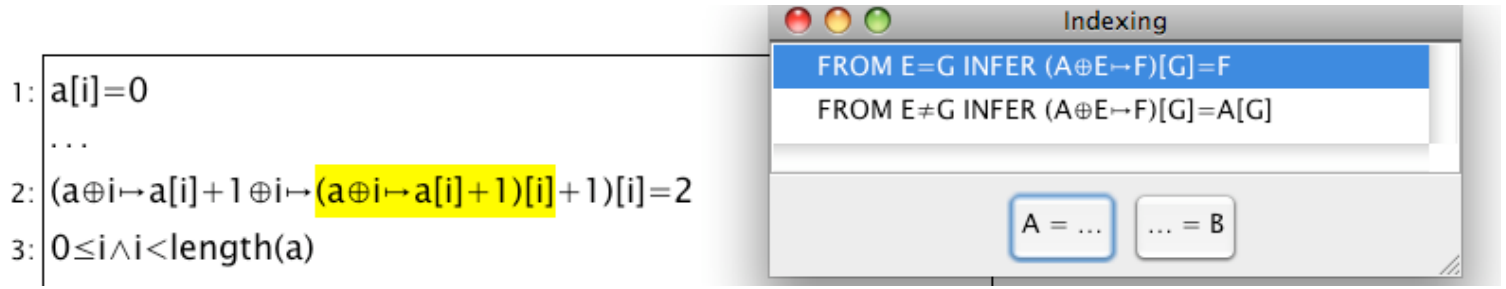


giving:

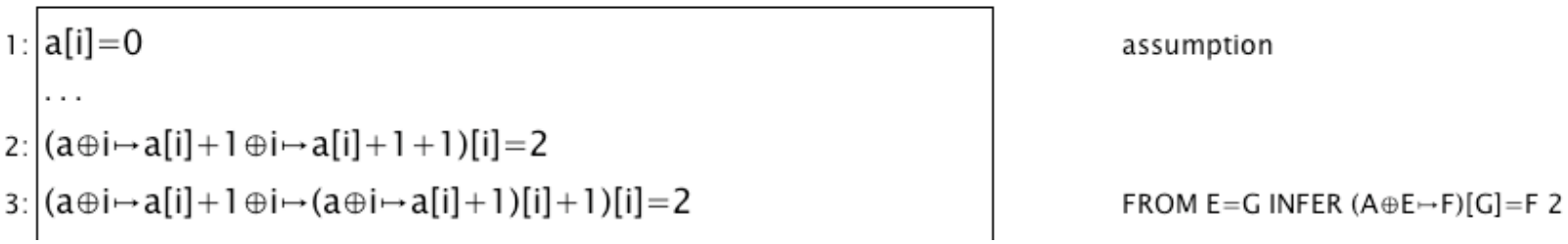


How to use GUI (continued)

- Alternatively we could have selected the *inner* modification expression first:

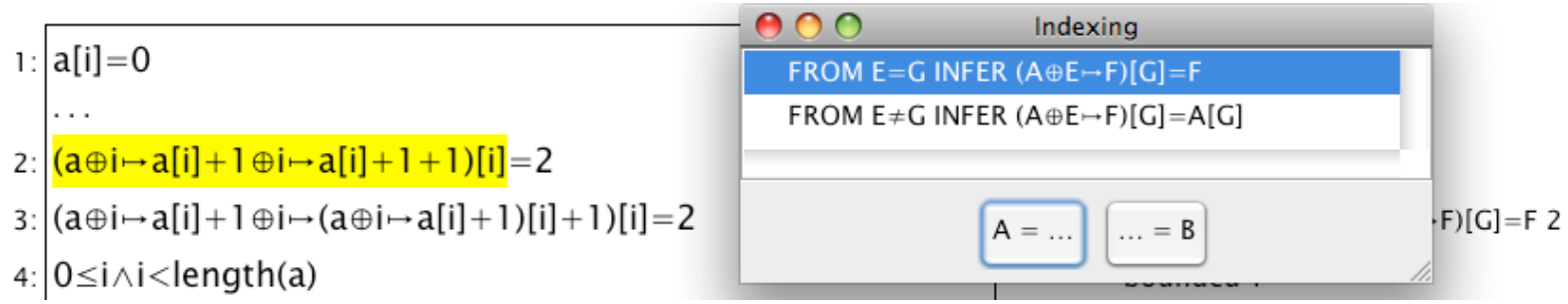


giving:

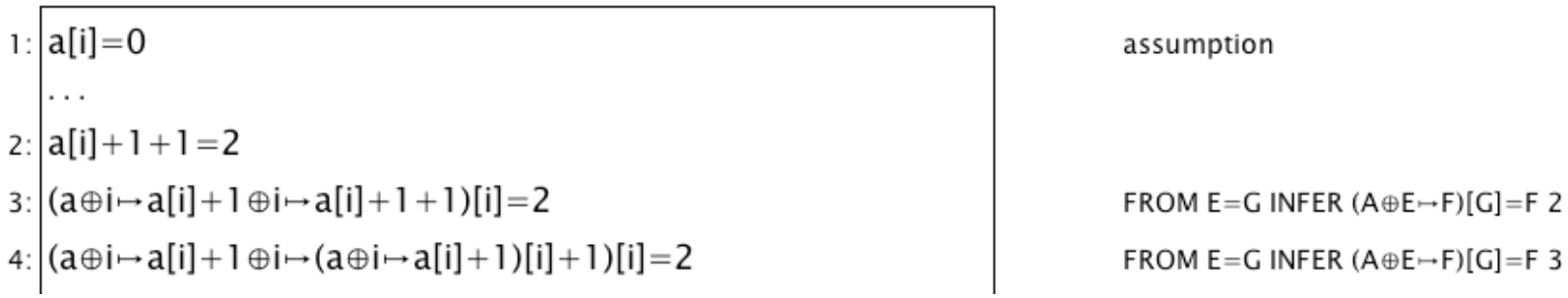


How to use GUI (continued)

- (Note that this is different from two slides ago).
  Then simplify that result:

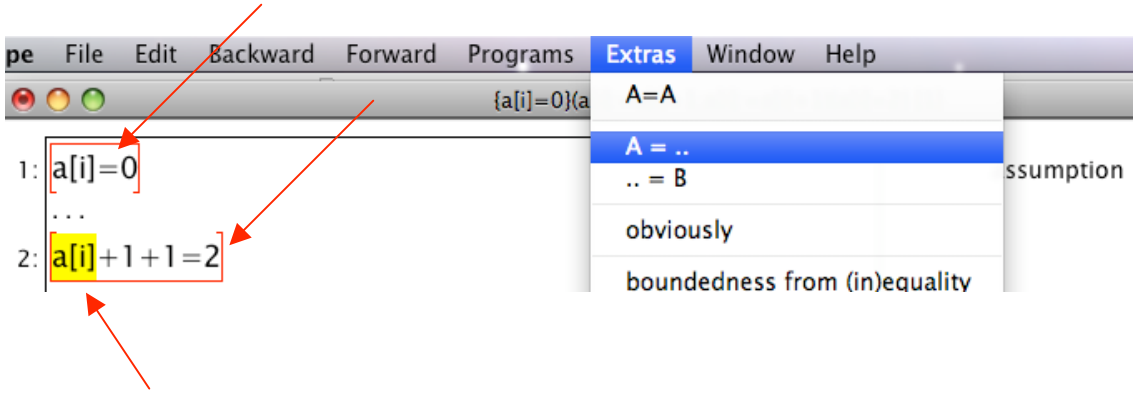


giving (as before):

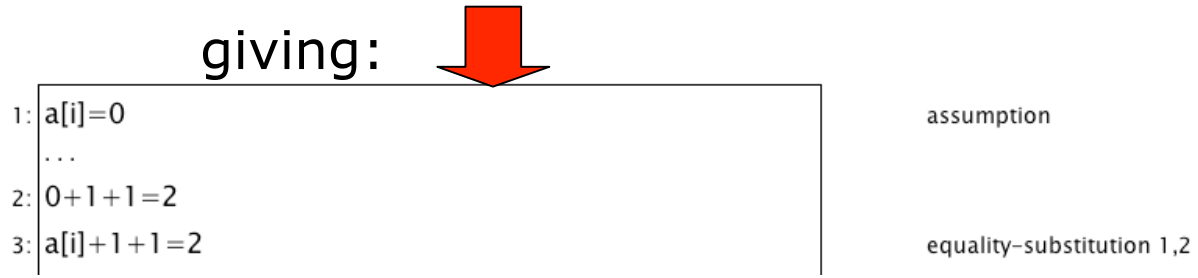


How to use GUI (continued)

- Use plain equality substitution to simplify the new goal. Note that both the goal and the equation defining the substitution are selected, and the sub-formula for which substitution is to occur is sub-formula selected (3 selections).



giving:



Consecutive Array Modification

1:	$a[i]=0$	assumption
2:	$0+1+1=2$	obviously
3:	$a[i]+1+1=2$	equality-substitution 1,2
4:	$(a\oplus i\mapsto a[i]+1)[i]+1=2$	FROM E=G INFER (A⊕E↦F)[G]=F 3
5:	$(a\oplus i\mapsto a[i]+1\oplus i\mapsto(a\oplus i\mapsto a[i]+1)[i]+1)[i]=2$	FROM E=G INFER (A⊕E↦F)[G]=F 4
6:	$0\le i\wedge i<\text{length}(a)$	bounded 1
7:	$0\le i$	∧ elim 6
8:	$i<\text{length}(a)$	∧ elim 6
9:	$(a\oplus i\mapsto a[i]+1\oplus i\mapsto(a\oplus i\mapsto a[i]+1)[i]+1)[i]=2\wedge 0\le i\wedge i<\text{length}(a)$	∧ intro 5,7,8
10:	$a[i]=0\rightarrow(a\oplus i\mapsto a[i]+1\oplus i\mapsto(a\oplus i\mapsto a[i]+1)[i]+1)[i]=2\wedge 0\le i\wedge i<\text{length}(a)$	→ intro 1-9
11:	$\{(a\oplus i\mapsto a[i]+1\oplus i\mapsto(a\oplus i\mapsto a[i]+1)[i]+1)[i]=2\wedge 0\le i\wedge i<\text{length}(a)\}$ $(a[i]:=a[i]+1)\{(a\oplus i\mapsto a[i]+1)[i]=2\wedge 0\le i\wedge i<\text{length}(a)\}$	array-element-assignment
12:	$\{a[i]=0\}(a[i]:=a[i]+1)\{(a\oplus i\mapsto a[i]+1)[i]=2\wedge 0\le i\wedge i<\text{length}(a)\}$	consequence(L) 10,11
13:	$\{(a\oplus i\mapsto a[i]+1)[i]=2\wedge 0\le i\wedge i<\text{length}(a)\}(a[i]:=a[i]+1)\{a[i]=2\}$	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

Subtleties with Array Programs

$\{\exists x.(0 \le x \wedge x < \text{length}(a) \wedge a[x]=0)\}(i:=0)$

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

- Look at part of the invariant here.
- Note that the lower bound on x is a function of the index i.
- This is important, because it says that the element such that **a[x] = 0 is yet to be found**.
- We need this invariant to prove termination.
- The loop test will stop when a[i] = 0.

- The expansion order is tricky.

Key Step #1: Split i ≤ i1

2: $0 \le i \wedge i < \text{length}(a) \wedge \exists x.(i \le x \wedge x < \text{length}(a) \wedge a[x]=0) \rightarrow 0 \le i \wedge i < \text{length}(a)$

3: $0 \le i \wedge i < \text{length}(a) \wedge \exists x.(i \le x \wedge x < \text{length}(a) \wedge a[x]=0) \wedge a[i] \neq 0$ — assumption

4: $0 \le i$ — ∧ elim 3

5: $i < \text{length}(a)$ — ∧ elim 3

6: $\exists x.(i \le x \wedge x < \text{length}(a) \wedge a[x]=0)$ — ∧ elim 3

7: $a[i] \neq 0$ — ∧ elim 3

8: integer i1 — assumption

9: $i \le i1 \wedge i1 < \text{length}(a) \wedge a[i1]=0$ — assumption

10: $i \le i1$ — ∧ elim 9

11: $i < i1 \vee i = i1$ — $A \le B \triangleq A < B \vee A = B$ 10

12: $i1 < \text{length}(a)$ — ∧ elim 9

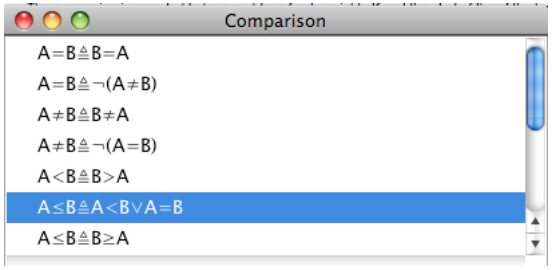
13: $a[i1]=0$ — ∧ elim 9

...

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

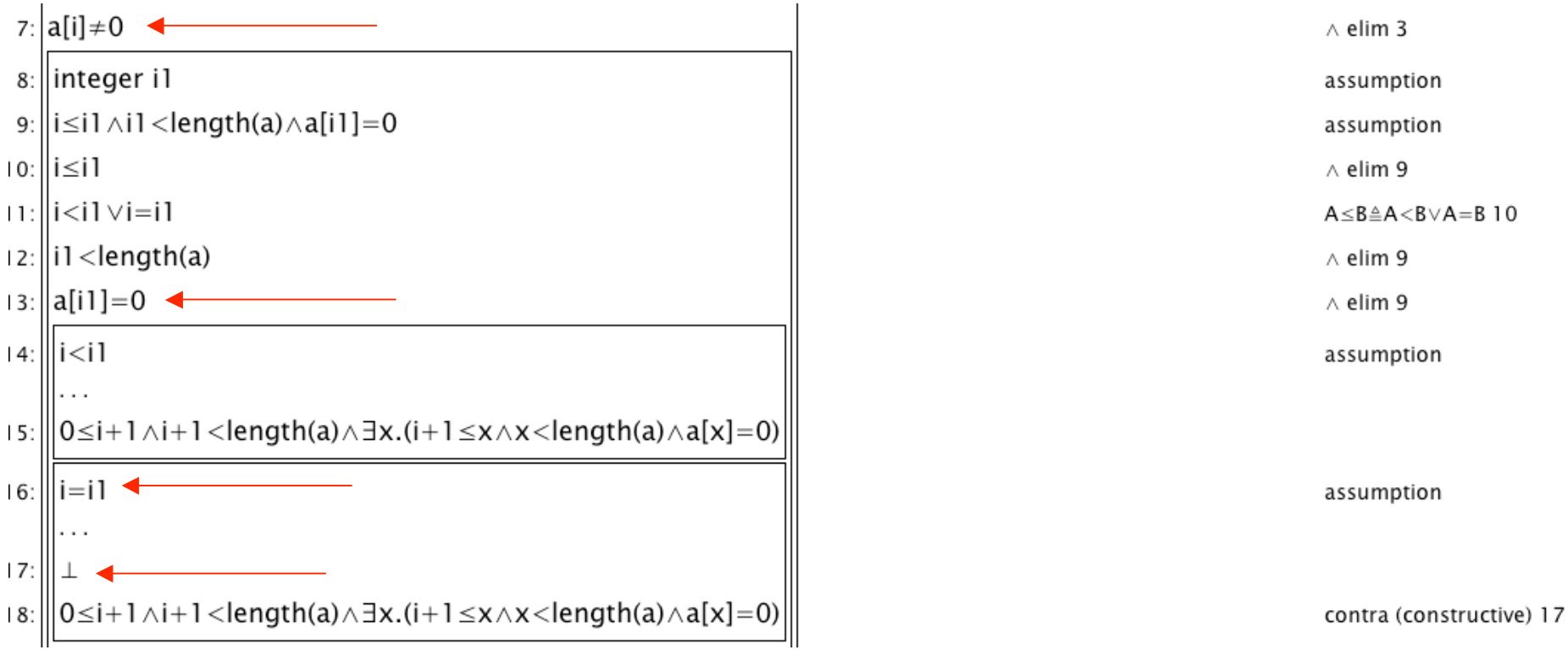
15: $0 \le i+1 \wedge i+1 < \text{length}(a) \wedge \exists x.(i+1 \le x \wedge x < \text{length}(a) \wedge a[x]=0)$ — ∃ elim 6,8–14

16: $0 \le i \wedge i < \text{length}(a) \wedge \exists x.(i \le x \wedge x < \text{length}(a) \wedge a[x]=0) \wedge a[i] \neq 0 \rightarrow 0 \le i+1 \wedge i+1 < \text{length}(a) \wedge \exists x.(i+1 \le x \wedge x < \text{length}(a) \wedge a[x]=0)$ — → intro 3–15



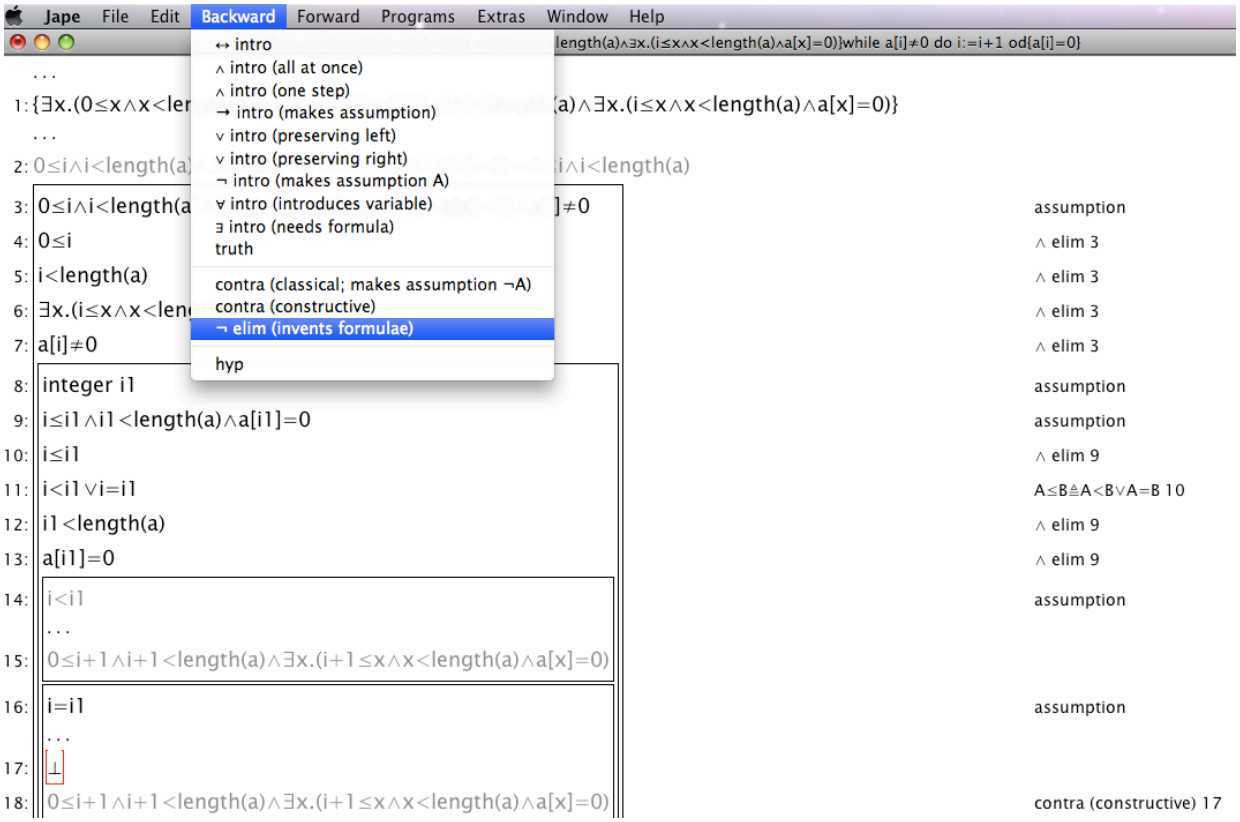
Then use ∨Elimination.

Key Step #2:
Aim for a contradiction in the i = i1 case.

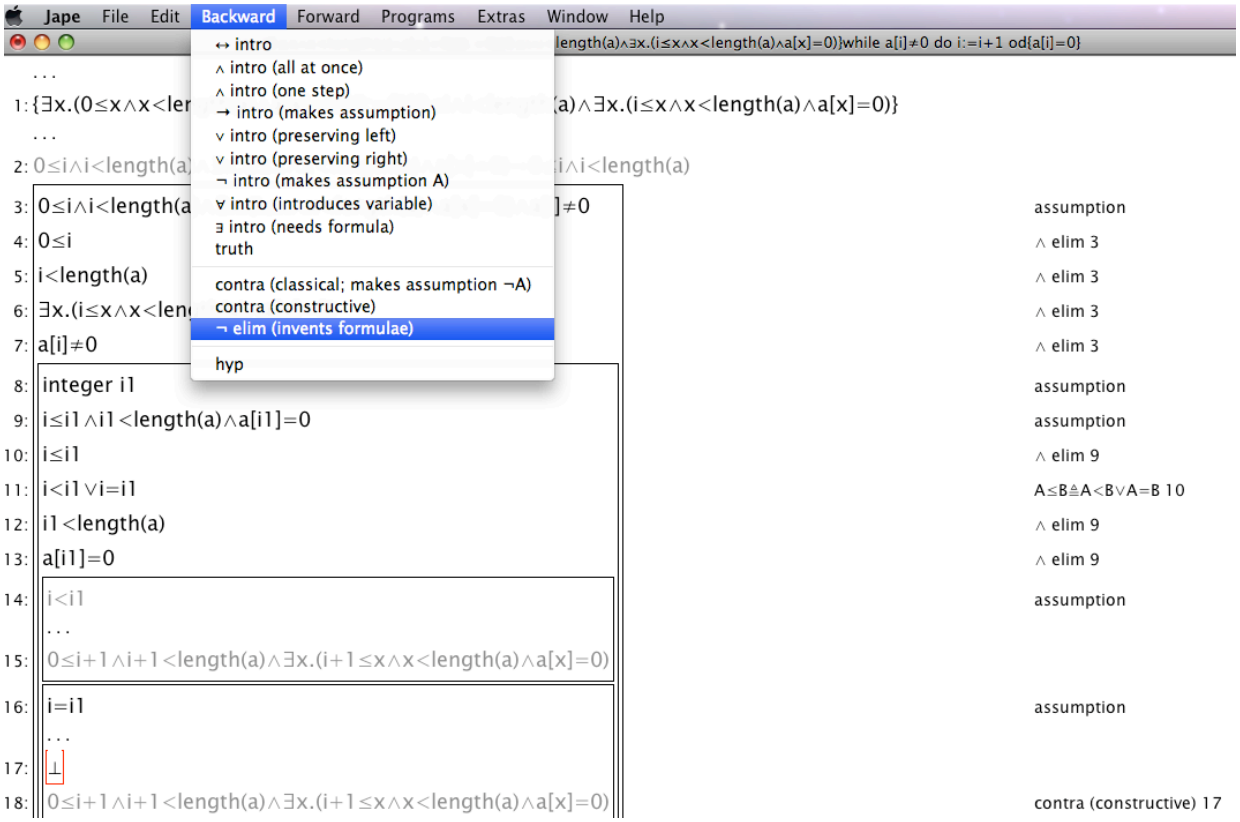


Now introduce a backward ¬Elimination.

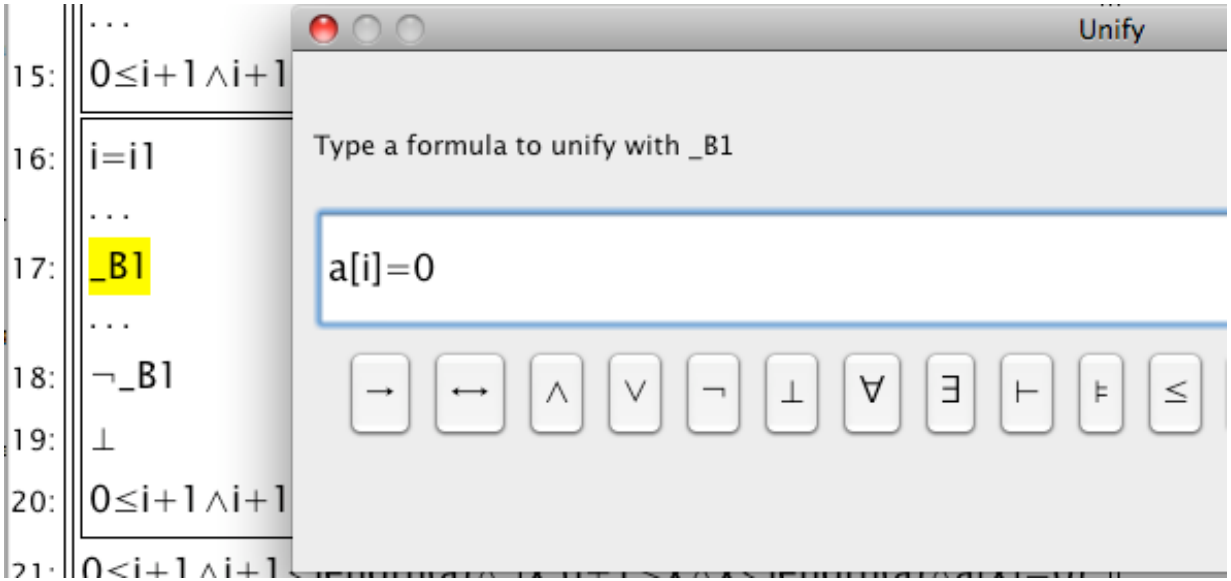
Key Step #2, continued:



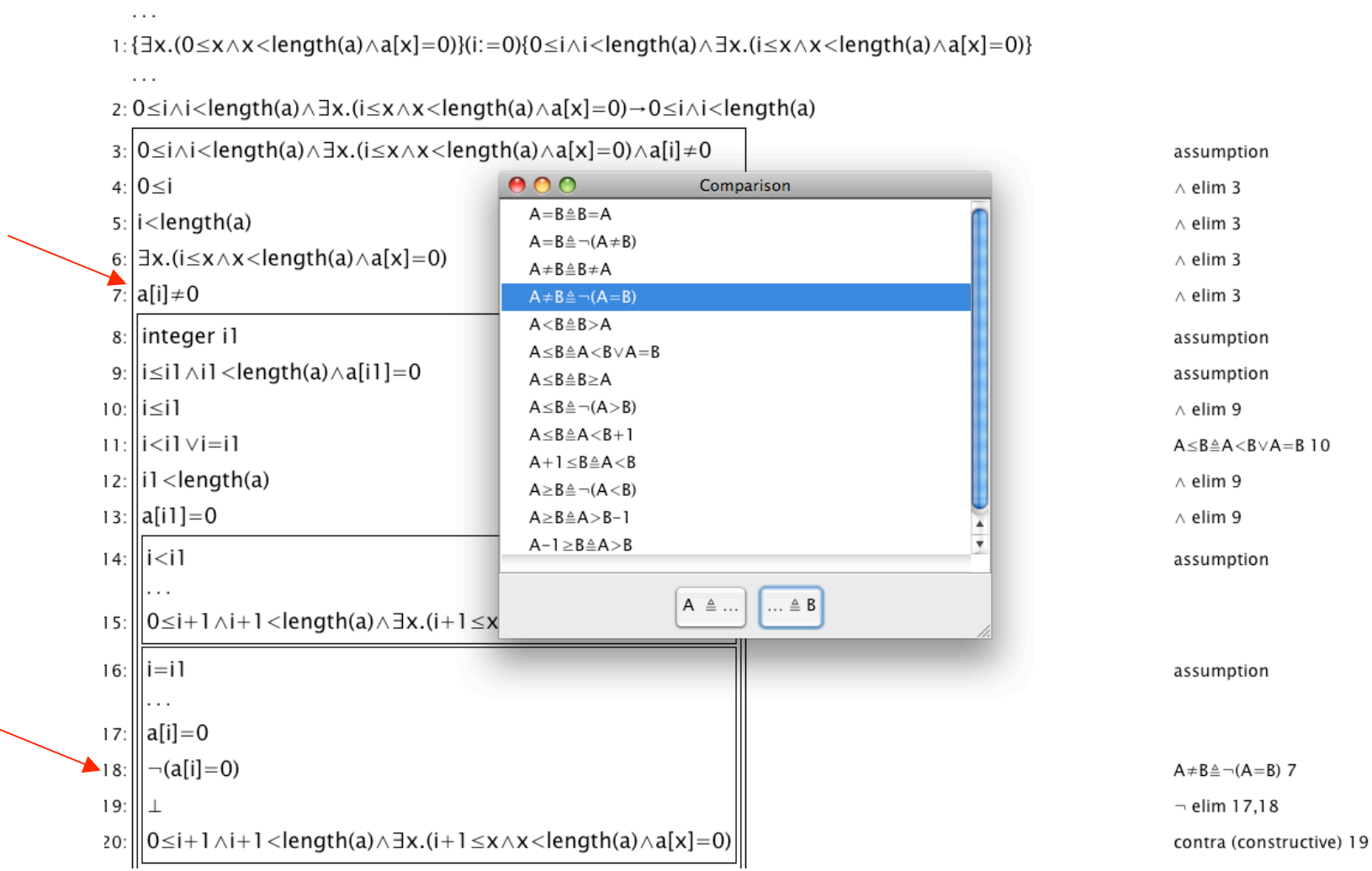
Key Step #2, continued:



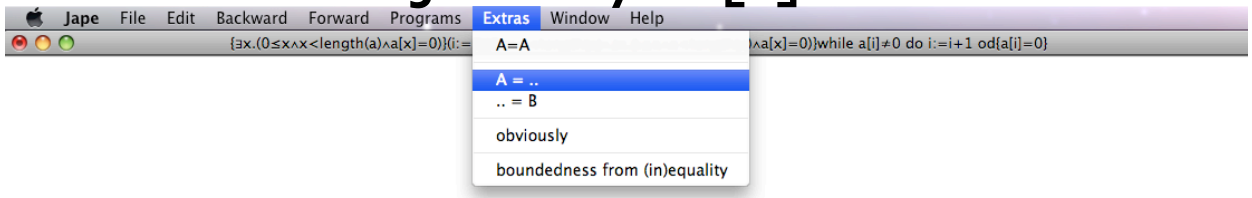
Key Step #2, continued: unify



Key Step #2, continued: use comparison menu to justify ¬(a[i] = 0)



Key Step #2, concluded: substitute to justify a[i] = 0



. . .

1: {∃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)}

. . .

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

3:	0≤i∧i<length(a)∧∃x.(i≤x∧x<length(a)∧a[x]=0)∧a[i]≠0	assumption
4:	0≤i	∧ elim 3
5:	i<length(a)	∧ elim 3
6:	∃x.(i≤x∧x<length(a)∧a[x]=0)	∧ elim 3
7:	a[i]≠0	∧ elim 3
8:	integer i1	assumption
9:	i≤i1∧i1<length(a)∧a[i1]=0	assumption
10:	i≤i1	∧ elim 9
11:	i<i1∨i=i1	A≤B≙A<B∨A=B 10
12:	i1<length(a)	∧ elim 9
13:	a[i1]=0	∧ elim 9
14:	i<i1	assumption
	. . .	
15:	0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	
16:	i=i1	assumption
	. . .	
17:	a[i]=0	
18:	¬(a[i]=0)	A≠B≙¬(A=B) 7
19:	⊥	¬ elim 17,18
20:	0≤i+1∧i+1<length(a)∧∃x.(i+1≤x∧x<length(a)∧a[x]=0)	contra (constructive) 19

Status following key step #2

1:	$\{\exists x.(0\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}(i:=0)\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}$	
	...	
2:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\rightarrow 0\le i\wedge i<\text{length}(a)$	
3:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0$	assumption
4:	$0\le i$	$\wedge$ elim 3
5:	$i<\text{length}(a)$	$\wedge$ elim 3
6:	$\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)$	$\wedge$ elim 3
7:	$a[i]\neq 0$	$\wedge$ elim 3
8:	integer $i1$	assumption
9:	$i\le i1\wedge i1<\text{length}(a)\wedge a[i1]=0$	assumption
10:	$i\le i1$	$\wedge$ elim 9
11:	$i<i1\vee i=i1$	$A\le B\triangleq A<B\vee A=B$ 10
12:	$i1<\text{length}(a)$	$\wedge$ elim 9
13:	$a[i1]=0$	$\wedge$ elim 9
14:	$i<i1$	assumption
	...	
15:	$0\le i+1\wedge i+1<\text{length}(a)\wedge\exists x.(i+1\le x\wedge x<\text{length}(a)\wedge a[x]=0)$	
16:	$i=i1$	assumption
17:	$a[i]=0$	equality-substitution 16,13
18:	$\neg(a[i]=0)$	$A\neq B\triangleq\neg(A=B)$ 7
19:	$\bot$	$\neg$ elim 17,18
20:	$0\le i+1\wedge i+1<\text{length}(a)\wedge\exists x.(i+1\le x\wedge x<\text{length}(a)\wedge a[x]=0)$	contra (constructive) 19
21:	$0\le i+1\wedge i+1<\text{length}(a)\wedge\exists x.(i+1\le x\wedge x<\text{length}(a)\wedge a[x]=0)$	$\vee$ elim 11,14-15,16-20
22:	$0\le i+1\wedge i+1<\text{length}(a)\wedge\exists x.(i+1\le x\wedge x<\text{length}(a)\wedge a[x]=0)$	$\exists$ elim 6,8-21

Completed Proof (lines 1-15)

1:	$\exists x.(0 \leq x \wedge x < \text{length}(a) \wedge a[x]=0)$	assumption
2:	$0 \leq 0$	obviously
3:	integer i2	assumption
4:	$0 \leq i2 \wedge i2 < \text{length}(a) \wedge a[i2]=0$	assumption
5:	$0 \leq i2$	$\wedge$ elim 4
6:	$i2 < \text{length}(a)$	$\wedge$ elim 4
7:	$0 < \text{length}(a)$	obviously, from 6,5
8:	$0 < \text{length}(a)$	$\exists$ elim 1,3–7
9:	$0 \leq 0 \wedge 0 < \text{length}(a) \wedge \exists x.(0 \leq x \wedge x < \text{length}(a) \wedge a[x]=0)$	$\wedge$ intro 2,8,1
10:	$\exists x.(0 \leq x \wedge x < \text{length}(a) \wedge a[x]=0) \rightarrow 0 \leq 0 \wedge 0 < \text{length}(a) \wedge \exists x.(0 \leq x \wedge x < \text{length}(a) \wedge a[x]=0)$	$\rightarrow$ intro 1–9
11:	$\{0 \leq 0 \wedge 0 < \text{length}(a) \wedge \exists x.(0 \leq x \wedge x < \text{length}(a) \wedge a[x]=0)\}(i:=0)\{0 \leq i \wedge i < \text{length}(a) \wedge \exists x.(i \leq x \wedge x < \text{length}(a) \wedge a[x]=0)\}$	variable-assignment
12:	$\{\exists x.(0 \leq x \wedge x < \text{length}(a) \wedge a[x]=0)\}(i:=0)\{0 \leq i \wedge i < \text{length}(a) \wedge \exists x.(i \leq x \wedge x < \text{length}(a) \wedge a[x]=0)\}$	consequence(L) 10,11
13:	$0 \leq i \wedge i < \text{length}(a) \wedge \exists x.(i \leq x \wedge x < \text{length}(a) \wedge a[x]=0)$	assumption
14:	$0 \leq i \wedge i < \text{length}(a)$	$\wedge$ elim(L) 13
15:	$0 \leq i \wedge i < \text{length}(a) \wedge \exists x.(i \leq x \wedge x < \text{length}(a) \wedge a[x]=0) \rightarrow 0 \leq i \wedge i < \text{length}(a)$	$\rightarrow$ intro 13–14

Completed Proof (lines 16-39)

16:	$0 \le i \wedge i < \text{length}(a) \wedge \exists x.(i \le x \wedge x < \text{length}(a) \wedge a[x]=0) \wedge a[i] \neq 0$	assumption
17:	$0 \le i$	$\wedge$ elim 16
18:	$\exists x.(i \le x \wedge x < \text{length}(a) \wedge a[x]=0)$	$\wedge$ elim 16
19:	$a[i] \neq 0$	$\wedge$ elim 16
20:	integer $i1$	assumption
21:	$i \le i1 \wedge i1 < \text{length}(a) \wedge a[i1]=0$	assumption
22:	$i \le i1$	$\wedge$ elim 21
23:	$i < i1 \vee i = i1$	$A \le B \triangleq A < B \vee A = B$ 22
24:	$i1 < \text{length}(a)$	$\wedge$ elim 21
25:	$a[i1]=0$	$\wedge$ elim 21
26:	$i < i1$	assumption
27:	$0 \le i+1$	obviously, from 17
28:	$i+1 < \text{length}(a)$	obviously, from 26,24
29:	$i+1 \le i1$	obviously, from 26
30:	$i+1 \le i1 \wedge i1 < \text{length}(a) \wedge a[i1]=0$	$\wedge$ intro 29,24,25
31:	$\exists x.(i+1 \le x \wedge x < \text{length}(a) \wedge a[x]=0)$	$\exists$ intro 30
32:	$0 \le i+1 \wedge i+1 < \text{length}(a) \wedge \exists x.(i+1 \le x \wedge x < \text{length}(a) \wedge a[x]=0)$	$\wedge$ intro 27,28,31
33:	$i = i1$	assumption
34:	$a[i]=0$	equality-substitution 33,25
35:	$\neg(a[i]=0)$	$A \neq B \triangleq \neg(A=B)$ 19
36:	$\bot$	$\neg$ elim 34,35
37:	$0 \le i+1 \wedge i+1 < \text{length}(a) \wedge \exists x.(i+1 \le x \wedge x < \text{length}(a) \wedge a[x]=0)$	contra (constructive) 36
38:	$0 \le i+1 \wedge i+1 < \text{length}(a) \wedge \exists x.(i+1 \le x \wedge x < \text{length}(a) \wedge a[x]=0)$	$\vee$ elim 23,26-32,33-37
39:	$0 \le i+1 \wedge i+1 < \text{length}(a) \wedge \exists x.(i+1 \le x \wedge x < \text{length}(a) \wedge a[x]=0)$	$\exists$ elim 18,20-38

Completed Proof (lines 40-60)

40:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0\rightarrow 0\le i+1\wedge i+1<\text{length}(a)\wedge\exists x.(i+1\le x\wedge x<\text{length}(a)\wedge a[x]=0)$	→ intro 16–39
41:	$\{0\le i+1\wedge i+1<\text{length}(a)\wedge\exists x.(i+1\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}(i:=i+1)\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}$	variable-assignment
42:	$\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0\}(i:=i+1)\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}$	consequence(L) 40,41
43:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0$	assumption
44:	$i<\text{length}(a)$	∧ elim 43
45:	$\text{length}(a)-i>0$	obviously, from 44
46:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0\rightarrow\text{length}(a)-i>0$	→ intro 43–45
47:	integer Km	assumption
48:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0\wedge\text{length}(a)-i=Km$	assumption
49:	$\text{length}(a)-i=Km$	∧ elim 48
50:	$\text{length}(a)-(i+1)<Km$	obviously, from 49
51:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0\wedge\text{length}(a)-i=Km\rightarrow\text{length}(a)-(i+1)<Km$	→ intro 48–50
52:	$\{\text{length}(a)-(i+1)<Km\}(i:=i+1)\{\text{length}(a)-i<Km\}$	variable-assignment
53:	$\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge a[i]\neq 0\wedge\text{length}(a)-i=Km\}(i:=i+1)\{\text{length}(a)-i<Km\}$	consequence(L) 51,52
54:	$\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}$while a[i]≠0 do i:=i+1 od $\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge\neg(a[i]\neq 0)\}$	while 15,42,46,47–53
55:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge\neg(a[i]\neq 0)$	assumption
56:	$\neg(a[i]\neq 0)$	∧ elim 55
57:	$a[i]=0$	A=B≙¬(A≠B) 56
58:	$0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\wedge\neg(a[i]\neq 0)\rightarrow a[i]=0$	→ intro 55–57
59:	$\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}$while a[i]≠0 do i:=i+1 od$\{a[i]=0\}$	consequence(R) 54,58
60:	$\{\exists x.(0\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}(i:=0)\{0\le i\wedge i<\text{length}(a)\wedge\exists x.(i\le x\wedge x<\text{length}(a)\wedge a[x]=0)\}$while a[i]≠0 do i:=i+1 od$\{a[i]=0\}$	Ntuple 12,59