Course 141
– Reasoning about Programs –

Loop Invariants and Natural Deduction

- We look at some imperative code (binary chop), how it meets its specification, and how to reason rigorously to show this.

- We reason using the postcondition to show properties of binary chop.

LOOP INVARIANT PROOF STRATEGY (REVISION)

We want to show:
"**if** Pre holds, **and** the code is executed **and** the code terminates, **then** Post will hold".

We don't discuss termination itself here.
We assume we've shown it (by reasoning about the variant)!

Must show:

Precondition for loop
{Loop Initialisation code}
{Loop code}
{Loop Finalisation code}
Postcondition for loop

To do this, show:

1. *Pre* {Loop Initialisation code} *Invariant*
2. *Invariant & (while condition)* {Loop code} *Invariant*
3. *Invariant & ¬(while condition)* {Loop Finalisation code} *Post*

A natural deduction view

Pre
{Loop Initialisation code}
Loop Invariant

Loop Invariant (before iteration)
(While condition)
{Loop statements}
Loop Invariant (after iteration)

Invariant ∧ not (While condition)
{Loop Finalisation code}
Post

Pre {Init code}{Loop statements}
{Final code} Post

EXAMPLE: Binary Chop (again assume a=a0 throughout)

```
int search(int [] a,  int x) {
//
//Pre:        Sorted(a): i.e.,
//            ∀i,j:int (0≤i≤j< a.length→a[i]≤a[j])
//Post:       0≤ r≤ a.length
//            ∧ ∀i:int (0≤ i<r → a[i]<x)
//            ∧ ∀i:int (r ≤ i < a.length → a[i]≥x)
//
//Loop Invariant :   0≤left<right≤a.length
//                   ∧ ∀i:int(0≤i≤left → a[i]<x)
//                   ∧  ∀i:int(right≤i< a.length →a[i]≥x)
//
//Loop Variant:  right – left-1
```

THE CODE FOR SEARCH (BINARY CHOP)

```
int search (int [] a, int x) {
// Pre:  Sorted(a)
  int left, mid;
  int right =a.length;
  if ((a.length==0||(a[0]>=x)) return 0; //a[0], …, a[a.length-1]≥x
  left = 0;      //a[left]<x & left<right=a.length
  while  (right-left>1) {
    mid= (left+right) / 2;     // left < middle < right
    if (a[mid]< x)             //Array access is legal
    left = mid;                //a[0], ..., a[left]<x
    else right = mid;          //a[right], ..., a[a.length-1]≥x
    }
  return right;
  }
```


GENERAL CONVENTIONS USED IN MY PART OF COURSE

Let v be a variable used in the code.

v0 is value of v at beginning of method

v is value of v now
Meaning of "now" depends on our current concern.
E.g. Now = at end of method.
Now = at start of method.
Now = just after an iteration of the loop.
// You should say what it means!

v1, v2 are names for value of v before an iteration, or at some other intermediate stage. Allows to refer to these values.
// You should say what they mean too!

In this problem (binary chop), we'll also assume:
All variables are of type **int**.

(Informal) proof that invariant is re-established

Let left1 and right1 be the values of the variables left and right at the start of an iteration.

We must show that if at the start of an iteration we have

0≤left1<right1≤a.length — (2) Inv.
∀i(0≤i≤left1→a[i]<x) — (3) Inv.
∀i(right1≤i<a.length→a[i]≥x) — (4) Inv.
left1<right1-1 — while condition is true

then at the end of the iteration we shall have

0≤left<right≤a.length — ie "now" is at end
∀i(0≤i≤left→a[i]<x)
∀i(right≤i<a.length→a[i]≥x) — invariant still true
right-left-1 < right1-left1-1 — variant decreased

The code ensures left1 < mid < right1 — (5) code
We are also given that a is sorted and as no assignments are made it remains sorted. i.e. ∀i,j(0≤i≤j<a.length→a[i]≤a[j]) —(1) pre

Informal proof (continued)

Case 1: a[mid]<x. (left = mid, right=right1)
1st conjunct: 0 ≤ left < right ≤ a.length <==> 0≤mid<right1≤a.length
True since 0≤left1 (2) <mid (5) <right1(5) ≤a.length (2).
2nd conjunct:∀i(0≤i≤left→a[i]<x) <==> ∀i(0≤i≤mid→a[i]<x)
True. ∀i(0≤i≤mid→a[i]≤a[mid]<x) (a is sorted by (1), and case)
3rd conjunct:∀i(right≤i<a.length→a[i]≥x) <==>
∀i(right1≤i<a.length→a[i]≥x) <==> true by (4).
So the invariant is re-established.

Case 2: a[mid]≥x (right = mid, left=left1)
0 ≤ left < right ≤ a.length <==> 0≤left1<mid≤a.length.
True since 0 ≤ left1 (2) <mid (5) <right1 (5) ≤ a.length (2).
∀i(0≤i≤left→a[i]<x) <==> ∀i(0≤i≤left1→a[i]<x) <==> true (3).
∀i(right≤i<a.length→a[i]≥x) <==> ∀i(mid≤i<a.length→a[i]≥x),
which is true by the case if ∀i(mid≤i<a.length→a[i]≥a[mid])
<==> true by (1). **So the invariant is re-established.**

Natural deduction proof that invariant is re-established.
Assumptions (as on slide 7):

Sorted(a),
i.e., ∀i,j(0≤i≤j<a.length→a[i]≤a[j]) —(1) pre
0≤left1<right1≤a.length —(2) Inv.
∀i(0≤i≤left1→a[i]<x) —(3) Inv.
∀i(right1≤i<a.length→a[i]≥x) —(4) Inv.
left1<mid<right1 —(5) code

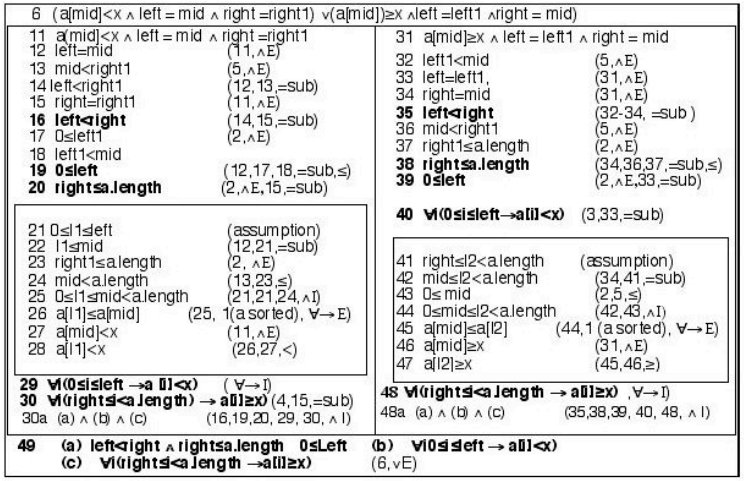
(5) implies the while condition is true, so we can now drop that.
We also write down the effect of the loop code in Logic:

(a[mid]<x ∧ left = mid ∧ right = right1)
or (a[mid]≥x ∧ left = left1 ∧ right = mid) —(6) code

Use natural deduction to prove the invariant after the iteration:

0≤left < right≤a.length —(a)
∀i(0≤i≤left→a[i]<x) —(b)
∀i(right≤i<a.length→a[i]≥x) —(c)

The full proof is on slide 10. The proof on slide 8 is sufficient.



(www.doc.ic.ac.uk/pandora/newraptor)

THE INFORMAL PROOF IN NATURAL DEDUCTION STYLE

Givens are (1) - (5) and (6):
(a[mid]<x∧left=mid∧right=right1) or (a[mid]≥x∧left=left1∧right=mid)
To show (a), (b) and (c) we'll make a case analysis by (∨E) (using (6)) and then for each case we'll show all of (a), (b) and (c) and then use (∧I). Here, we'll just show (b) for the first case. The outline structure is:

6 (a[mid]<x∧left=mid∧right =right1)∨(a[mid])≥x∧left =left1∧right= mid)

7 a[mid]<x∧left = mid∧ right =right1	a[mid])≥x ∧left =left1 ∧right= mid
8 sk1 (∀I) 0≤sk1≤left <fill in this part ...> 9 a[sk1]<x	
10 ∀i(0≤i≤left→a[i]<x) (∀→I) (b)	(a) ∧ (b) ∧ (c) (∧ I)
11 (a) ∧ (b) ∧ (c) (∧ I)	
12 (a) ∧ (b) ∧ (c) (∨E, 6)	

proof continued

7 a[mid]<x∧left = mid∧ right =right1

8 sk1 (∀I)	0≤sk1≤left
13	left=mid (∧E, 7)
14	0≤sk1≤mid (=sub, 13, 8)
15	a[sk1]≤a[mid] (∀→E, 14,1– i.e. a is sorted)
16	a[mid]<x (∧E, 7)
9	a[sk1]<x (15,16, transitivity of < and ≤

10 ∀i(0≤i≤left→a[i]<x) (∀→I)

The main difference between this more formal proof and the one on slide 8 is that in this one the quantifiers have to be removed in order to show the main implication.

Note also the reason for line 9 - it is still fairly informal (but OK).
Other cases can be completed in a similar way (see proof on slide 10).

NOTES ON STYLE

The informal proof is similar to the natural deduction proof, but has "obvious" bits omitted (e.g. lines 13 and 16 on slide 12).

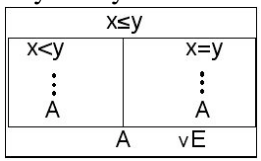
In this course some informality is OK in natural deduction too. Eg:

— Quoting properties of <, ≤, + (e.g., line 9 on slide 12).
Label with "≤" as justification. Anything reasonable goes.

— Implicitly using types of things
(e.g., don't bother to keep saying v is **int**).

— x≤y≤z can be used to abbreviate x≤y ∧ y≤z.

— x≤y is equivalent to x<y ∨ x=y. So the following is OK:



PROS AND CONS OF NATURAL DEDUCTION

\+ You (should) know how to do it.
\+ You (should) know what to write down.
\+ Maximises rigour, and minimises writing.
\+ Encourages working backwards.

\- Can be long and tedious (unless you use "trust me" often — but then it's easy to make mistakes).

Conclusion:
You should be able to use natural deduction *and* do informal proofs.

PROVING THINGS FROM POSTCONDITIONS

A program's specification is expressed as "pre ==> post".
When we use a program, assuming that pre holds, then what happens should depend solely on post.

Example: let's prove from the postcondition of Search that
(∀i(0≤i<a.length→a[i]≥x)) → **r** = 0.

The postcondition (*Post*) was: 0≤**r**≤a.length ∧
∀i:int (0≤i<**r** → a[i]<x) ∧ ∀i:int (**r**≤i<a.length → a[i]≥x).

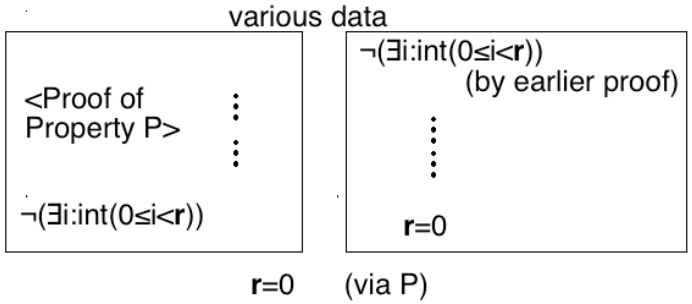
(Very) informal proof (a natural deduction proof follows):

Assume ∀i(0≤i<a.length→a[i]≥x) ==> all entries in a are ≥ x (a).
Post ==> all entries in a before **r** are <x (b).
(a) and (b) ==> there are no entries in a before **r** (c).
(Any such entry would be both <x and ≥x, impossible.) Hence **r** = 0.
(**r**≥0 by *Post*; **r**>0 ==> a[0] is before **r** contradicting (c), so **r** = 0.)

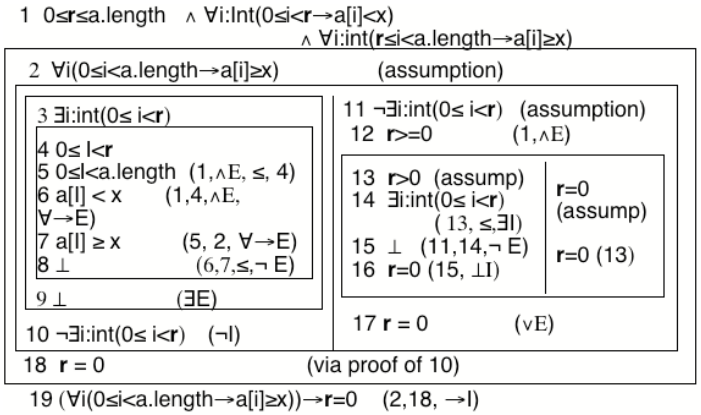
To show r=0, we use the following strategy:

(i) show an intermediate property P and
(ii) use P to show r=0

Here, the property P is ¬(∃i:int(0≤i<r))



Here is the corresponding formal natural deduction proof:



Second (Simpler) Proof

Show (∀i(0≤i<a.length→a[i]≥x)) → **r** = 0:

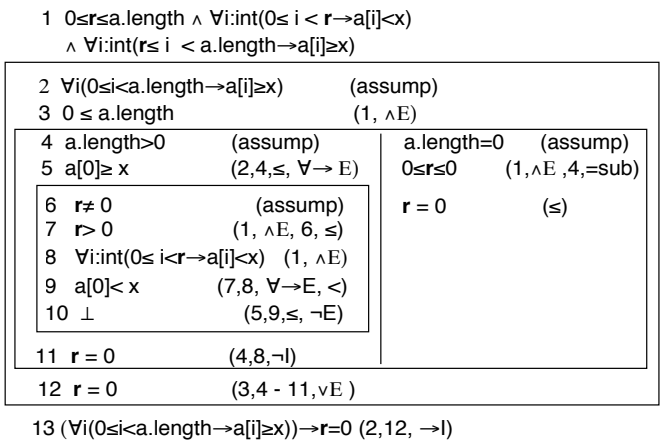
By the assumption, for every i, a[i]≥x.

Case 1: a.length>0. Hence a[0]≥x. Suppose, for contradiction, that **r**≠0, then by *Post* **r**>0 and a[0]<x. A contradiction, so **r**=0.

Case 2: a.length=0. Hence by *Post* 0≤**r**≤0 ==> **r**=0.
Since a.length≥0, either way **r**=0.

The corresponding formal natural deduction proof is on slide 19.

Exercise: prove that the post-condition of Search implies that (∀i(0≤i<a.length→a[i]<x)) → **r** = a.length.



Example: Show ∀x,y:Nats [x≤y → search(a,x)≤ search(a,y)].

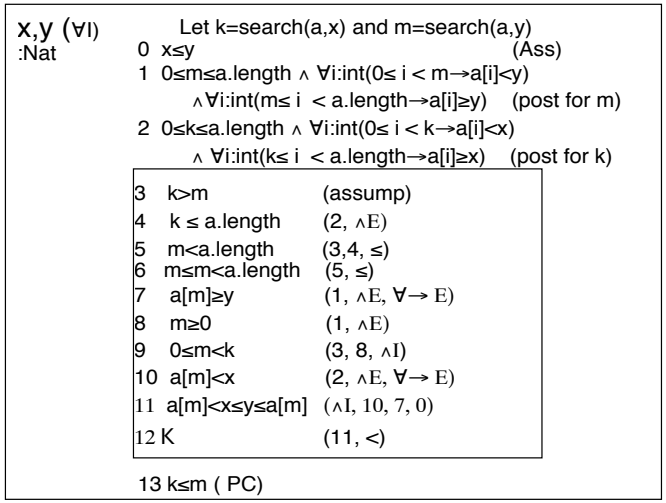
Informal proof: (A formal natural deduction proof is on slide 21.)

Let x≤y for arbitrary Nats x and y. We're required to show search(a,x)≤ search(a,y). We'll call k=search(a,x) and m=search(a,y), so we have to show k≤m.

Suppose, for contradiction, that k>m.
The postcondition of search gives lots of properties about m and k:
0≤m≤a.length∧∀i:int(0≤i<m→a[i]<y)∧∀i:int(m≤i<a.length→a[i]≥y)
and
0≤k≤a.length∧∀i:int(0≤i<k→a[i]<x)∧∀i:int(k≤i<a.length→a[i]≥x).

From the "k postcondition", and because 0≤m<k, a[m]<x, and from the "m postcondition", and because m<k≤a.length, a[m]≥y. Hence a[m]<x≤y≤a[m], or a[m]<a[m], which is impossible, so k≤m.

Exercise: Call the "m properties" (1), (2), (3) and the "k properties" (4), (5), (6). State which properties are used where in the proof.
The exercise sheet suggests other properties to prove about binchop.



∀x,y:Nat [x≤y → search(a,x) ≤ search(a,y)] (∀→E)