CSC236 winter 2020, week 7: Iterative correctness

Recommended supplementary reading: Chapter 2 Vassos course notes

Colin Morris

colin@cs.toronto.edu

http://www.cs.toronto.edu/~colin/236/W20/

- A1 results on MarkUs

- A2 [i - 1] posted

February 24, 2020

Outline

- Correctness proof pitfalls
  - Actual vs. expected behaviour
  - Level of detail / justification
- Finishing recursive correctness
- Iterative correctness

isuniform (quiz 6, version 2)

```
def isuniform(A):
  """Pre: A is a list
  Post: Return True if and only if every element in A is the same.
  """
  if len(A) <= 1:
    return True
  return A[0] == A[1] and isuniform(A[1:])
```

What's wrong with this proof?

```python
1 def isuniform(A):
2   """Pre: A is a list
3   Post: Return True if and only if every element in A is the same.
4   """
5   if len(A) <= 1:
6     return True
7   return A[0] == A[1] and isuniform(A[1:])
```

$P(n)$: for all lists A of length n $\text{Pre}(A) \implies$ `isuniform` terminates and satisfies $\text{Post}(A)$.

<u>Basis:</u> Let A be a list of length 0 or 1. By lines 5-6, `isuniform` returns True, so $P(0) \wedge P(1)$.

<u>IS:</u> Let $n \in \mathbb{N}^+$ and assume $P(n)$. WTS: $P(n+1)$. Let A be a list of length $n+1$, and assume $\text{Pre}(A)$.

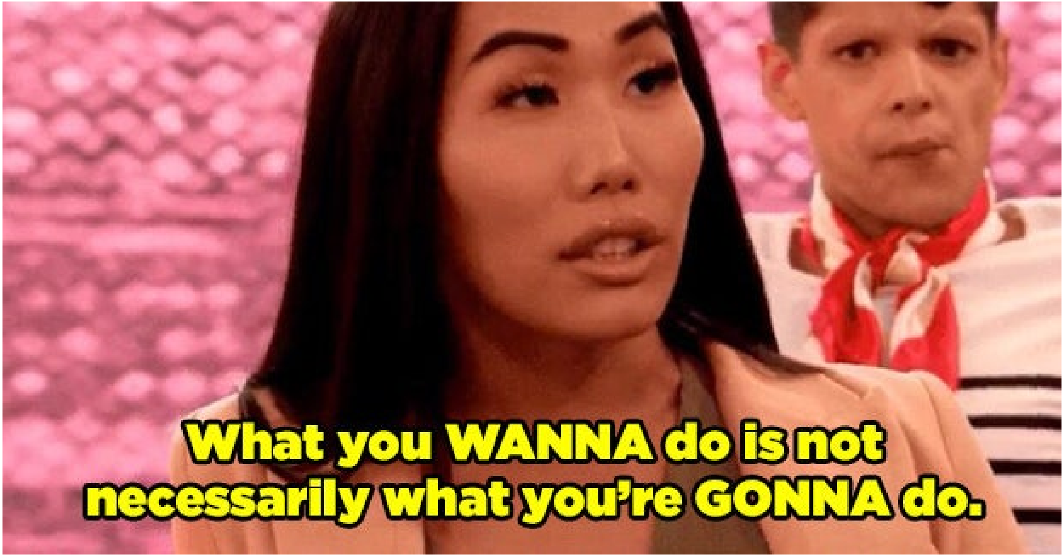
Case 1: $A[0] \neq A[1]$. Then by line 7, we return False, so $\text{Post}(A)$.

Case 2: $A[0] = A[1]$. By IH, `isuniform(A[1:])` returns True or False.

Case 2a: `isuniform(A[1:])` is True. Then line 7 returns `(True and True)`, which evaluates to `True`, so $\text{Post}(A)$ holds.

Case 2b: `isuniform(A[1:])` is False. Then line 7 returns `(True and False)`, which evaluates to `False`, so $\text{Post}(A)$ holds.

$\text{Post}(A)$ holds in all cases, so $P(n+1)$. $\blacksquare$


What you WANNA do is not
necessarily what you're GONNA do.

Actual vs. expected

For any input, your proof should address:

1. What a function *should* return on that input in order to satisfy the postcondition
2. What our function *actually* returns given that input

Aren't these the same thing?

- This is exactly what you need to prove.

Actual vs. expected

Cases are commonly patterned on one or the other.

- Case 1: My function returns X
  - here's why X is the right answer in this case
- Case 2: My function returns Y
  - here's why Y is the right answer in this case
- ...

Or

- Case 1: The correct answer is X
  - here's why my function actually returns X on these inputs
- Case 2: The correct answer is Y
- ...

Actual $\rightarrow$ Expected

```
1 def isuniform(A):
2   """Pre: A is a list
3   Post: Return True if and only if every element in A is the same.
4   """
5   if len(A) <= 1:
6     return True
7   return A[0] == A[1] and isuniform(A[1:])
```

Assume our function is correct for inputs of size n for some $n \in \mathbb{N}^+$. Let A be a list of length $n+1$, and assume Pre(A). Then `isuniform(A)` reaches line 7 and returns `(A[0] = A[1] and isuniform(A[1:]))`. By the IH, `isuniform(A[1:])` $\iff$ $A[1:]$ is uniform.

Case 1: We return True. Then, every element in $A[1:]$ is equal to $A[1]$. Since $A[0] = A[1]$, every element in A is the same.
Case 2: We return False. Then by line 7, at least one of the following is true:

- $A[0] \neq A[1]$
- $A[1:]$ is non-uniform

In either case, this means A is not uniform.

Expected $\rightarrow$ Actual

```python
1 def isuniform(A):
2   """Pre: A is a list
3   Post: Return True if and only if every element in A is the same.
4   """
5   if len(A) <= 1:
6     return True
7   return A[0] == A[1] and isuniform(A[1:])
```

Assume our function is correct for inputs of size n for some $n \in \mathbb{N}^+$. Let A be a list of length $n+1$, and assume Pre(A). Then `isuniform(A)` reaches line 7 and returns `(A[0] = A[1] and isuniform(A[1:]))`. By the IH, `isuniform(A[1:])` $\iff$ $A[1:]$ is uniform.

Case 1: A is uniform. Then it follows that $A[0] = A[1]$ and that any sublist of A, including $A[1:]$ is uniform. Thus we return True.

Case 2: A is not uniform. Then let $i \in \mathbb{N}$ be the smallest index such that $A[i] \neq 0$ (such an i must exist, otherwise A would uniformly consist of the element $A[0]$).

Case 2a: $i = 1$. Then $A[0] \neq A[1]$ and we return False.

Case 2b: $i > 1$. Then $A[1:]$ is non-uniform, since it contains $A[i]$, and $A[1:][0] = A[0] \neq A[i]$. So we return False.

In either case, we return False, as required.

Level of detail / justification

At this stage, your induction proofs can be a bit less formal. e.g.

- Don't *need* to define a predicate
- Can omit justification of some 'obvious' facts
  - "A is a list of natural numbers, therefore $A[1:]$ is also a list of natural numbers"
- Don't need to specify domain of variables *if it's clear from context*
  - "Let $i \in \mathbb{N}$ be the index of..." $\rightarrow$ "Let i be the index of..."
  - For convenience, we can use the notation $\mathbb{N}^*$ to denote the set of lists of natural numbers (and similarly for $\mathbb{Z}^*$, $\mathbb{R}^*$, etc.).

(But note that taking off the training wheels $\rightarrow$ more speed, but easier to wipe out)

isuniform sample solution

The bare minimum.

```
1 def isuniform(A):
2   """Pre: A is a list
3   Post: Return True if and only if every element in A is the same.
4   """
5   if len(A) <= 1:
6     return True
7   return A[0] == A[1] and isuniform(A[1:])
```

Basis: any list of length 0 or 1 is uniform, and our function returns True for such lists
IS: assume that our function is correct on inputs of size n , $n > 0$.
Let A be a list of length $n+1$.
Case 1: A is uniform. It follows that $A[0] = A[1]$ and that $A[1:]$ is uniform. So our function returns True.
Case 2: A is not uniform. Then by definition, there exists an index i such that $A[0] \neq A[i]$. Let i' be the smallest such index. If $i' = 1$, then we return False . If $i' > 1$, then it follows that $A[1:]$ is not uniform . Thus we return False on line 7.

■

`isuniform` sample solution

The bare minimum.

Basis: any list of length 0 or 1 is uniform, and our function returns True for such lists

IS: assume that our function is correct on inputs of size n , $n > 0$.

Let A be a list of length $n + 1$.

Case 1: A is uniform. It follows that $A[0] = A[1]$ and that $A[1 :]$ is uniform. So our function returns True.

Case 2: A is not uniform. Then by definition, there exists an index i such that $A[0] \neq A[i]$

. Let i' be the smallest such index. If $i' = 1$, then we return False .

If $i' > 1$, then it follows that $A[1 :]$ is not uniform

. Thus

we return

False on line 7.

$\blacksquare$

isuniform sample solution

Recommended, but not required

Basis: any list of length 0 or 1 is uniform, and our function returns True for such lists , by lines 5-6.

IS: Let $n \in \mathbb{N}^+$, and assume that our function is correct on inputs of size n .

Let A be a list of length $n+1$ satisfying the precondition .

Because $n+1 \geq 2$, we reach line 7 in the code.

Case 1: A is uniform. It follows that $A[0] = A[1]$ and that $A[1:]$ is uniform. So our function returns True.

Case 2: A is not uniform. Then by definition, there exists an index i such that $A[0] \neq A[i]$

. Let i' be the smallest such index. If $i' = 1$, then we return False , by the first condition of line 7 .

If $i' > 1$, then it follows that $A[1:]$ is not uniform , because the sublist contains an element not equal to $A[0]$, and the first element of the sublist ($A[1]$) is equal to $A[0]$. Thus , by the IH, because $A[1:]$ is not uniform, the recursive call returns False, meaning that we return False on line 7.

In either case, our function matches the postcondition for an arbitrary input of size $n+1$. ■

`isuniform` sample solution

Recommended, but not required   Nice-to-have

Basis: any list of length 0 or 1 is uniform, and our function returns True for such lists , by lines 5-6.

IS: Let $n \in \mathbb{N}^+$, and assume that our function is correct on inputs of size n .

Let A be a list of length $n+1$ satisfying the precondition .

Because $n+1 \geq 2$, we reach line 7 in the code.

Since len($A[1:] = n$, by the IH, `isuniform(A[1:])` is True iff $A[1:]$ is uniform.

Case 1: A is uniform. It follows that $A[0] = A[1]$ and that $A[1:]$ (indeed, any sublist of A) is uniform. So our function returns True.

Case 2: A is not uniform. Then by definition, there exists an index i such that $A[0] \neq A[i]$ (it is easy to see that the negation of this statement entails that A uniformly consists of instances of $A[0]$) . Let i' be the smallest such index. If $i' = 1$, then we return False , by the first condition of line 7 .

If $i' > 1$, then it follows that $A[1:]$ is not uniform , because the sublist contains an element not equal to $A[0]$, and the first element of the sublist ($A[1]$) is equal to $A[0]$. Thus , by the IH, because $A[1:]$ is not uniform, the recursive call returns False, meaning that we return False on line 7.

In either case, our function matches the postcondition for an arbitrary input of size $n+1$. ■

`isuniform` sample solution (bonkers level of detail)

Not recommended!

$P(n)$: for any input A having $\text{len}(A) = n$, if A satisfies the precondition, then `isuniform(A)` terminates and satisfies the postcondition.
Basis: Let A be a list satisfying the precondition such that $\text{len}(A) \leq 1$. By lines 5-6, `isuniform(A)` returns True. I will show that this matches the postcondition, i.e. A is uniform.
Case 1: $\text{len}(A) = 0$, i.e. $A = [\,]$. Since A has no elements, it is vacuously true that they are all equal.
Case 2: $\text{len}(A) = 1$. It follows that for all valid pairs of indices i, j that $A[i] = A[j]$, since there is only one valid index, 0.
So $P(0) \wedge P(1)$.
IS: Let $n \in \mathbb{N}^+$, and assume $P(n)$. WTS: $P(n+1)$.
Let A be a list of length $n+1$ satisfying the precondition. Because $n + 1 \geq 2$, we reach line 7 in the code and return `(A[0] == A[1] and isuniform(A[1:]))`. Note that:

- Since $\text{len}(A) = n + 1 \geq 2$, $A[0]$ and $A[1]$ are legal index expressions (i.e. they do not raise an `IndexError`).
- By the IH, the recursive call to `isuniform(A[1:])` terminates and returns True iff $A[1:]$ is uniform. We are justified in applying $P(n)$ to draw this conclusion because...
  - Since A satisfies the precondition, $A[1:]$ does as well, since a slice of a list is also a list.
  - $\text{len}(A[1:]) = n$

Case 1: A is uniform. Since all elements in A are equal, it follows that $A[0] = A[1]$ and that $A[1:]$ (indeed, any sublist of A) is uniform. So our function returns `(True and True)` which evaluates to True, as required by the postcondition.
Case 2: A is not uniform. Claim: there exists an index i such that $A[0] \neq A[i]$. Suppose for the sake of contradiction that no such index exists. Then $\forall j \in \mathbb{N}, j < \text{len}(A) \implies A[0] = A[j]$. But this would mean that A is uniform, contradicting the assumption of our case. Therefore such an index does exist. Let i' be the smallest such index (i' is guaranteed to exist by the Principle of Well-ordering, since the set $S = \{i \in \mathbb{N} \mid A[0] \neq A[i]\}$ is a non-empty subset of $\mathbb{N}$).
Case 2a:$i' = 1$. Then $A[0] \neq A[1]$, and we return False by the first condition of line 7.
Case 2b: $i' > 1$, then it follows that $A[1:]$ is not uniform, because the sublist contains an element not equal to $A[0]$, and the first element of the sublist ($A[1]$) is equal to $A[0]$. Thus, by the IH, because $A[1:]$ is not uniform, the recursive call returns False, meaning that we return False on line 7.
Note that $i' > 0$, since $i' = 0$ would imply $A[0] \neq A[0]$, a contradiction. Therefore cases 2a and 2b are exhaustive. In both subcases, our function returns False, as required.
In each outer case, our function matches the postcondition for an arbitrary input of size $n+1$ meeting the precondition. Thus $P(n+1)$.

$P(0) \wedge P(1) \wedge \left(\forall n \in \mathbb{N}^+, P(n) \implies P(n+1)\right)$, so by the principle of induction, $\forall n \in \mathbb{N}, P(n)$. ■

Return of silly sum

$\mathbb{N}^*$ = set of all sequences of nats

```python
def sum(A):
  """Pre: A is a list containing
  only natural numbers.
  Post: return the sum of the
  numbers in A."""
  if len(A) == 0:
    return 0
  first = A[0]
  if first == 0:
    return sum(A[1:])
  else:
    A[0] = A[0] - 1
    return 1 + sum(A)
```

Proof sketch:

- Lemma 1: For all non-empty $A \in \mathbb{N}^*$, `sum(A)` returns $A[0]+$ `sum(A[1:])`.
  - Prove by induction on $A[0]$
  - Note: this doesn't imply termination!
- Theorem: `sum` is correct
  - Prove by induction on length of A, using Lemma 1

(Breaking a tricky proof into intermediate lemmas is an important skill, especially for correctness proofs, which can have many interacting parts. This comes up in a big way in A2 question 3.)

Lemma 1: `sum(A)` returns `A[0] + sum(A[1:])` for non-empty A

By induction on the head

$i = i+1 \Rightarrow \leq$

```
1  def sum(A):
2    """Pre: A is a list containing
3    only natural numbers.
4    Post: return the sum of the
5    numbers in A."""
6    if len(A) == 0:
7      return 0
8    first = A[0]
9    if first == 0:
10     return sum(A[1:])
11   else:
12     A[0] = A[0] - 1
13     return 1 + sum(A)
```

$P(k)$: for all lists A where $A[0] = k$, `sum(A)` returns $k+$ `sum(A[1:])`.

Basis: Let A be a list of form $[0, \ldots]$
by l 8-10, we return sum(A[1:]) + 0

IS: Assume $P(k)$ for some k. Let A be a list starting with $k+1$. `sum(A)` reaches lines 12-13... Let A_0 be A at the start of the fn

$A_0 = [k+1, \ldots]$ Let A_1 be A after l12

after l12 $A_1 = [k, \ldots]$

l13 returns $1 + \text{sum}(A_1)$

$= 1 + (k + \text{sum}(A_1[1:]))$ # by IH

$= 1 + k + \text{sum}(A_1[1:])$

$\Downarrow$

note $A_1[1:] = A_0[1:]$

$\therefore P(k+1)$

Main course: correctness of sum

By induction on length of A, and a little help from Lemma 1

```
1   def sum(A):
2     """Pre: A is a list containing
3     only natural numbers.
4     Post: return the sum of the
5     numbers in A."""
6     if len(A) == 0:
7       return 0
8     first = A[0]
9     if first == 0:
10      return sum(A[1:])
11    else:
12      A[0] = A[0] - 1
13      return 1 + sum(A)
```

IS Assume $Q(n)$ for $n \in \mathbb{N}$

Let A be list of size $n+1$

By lemma 1, sum(A) returns...

$A[0] + \text{sum}(A[1:])$

$= A[0] + [\text{sum of } A[1:]]$

$=$ sum of A,

so $P(n+1)$

$Q(n)$: For all lists A of size n, $\texttt{sum(A)} = \sum_{x \in A} x$.

Basis: $Q(0)$ by l 6-7

Sometimes code has loops

```
def imax(A):
  """Pre: A is non-empty and contains comparable items.
  Post: return the maximum element in A
  """
  curr = A[0]
  i = 1
  while i < len(A):
    if A[i] > curr:
      curr = A[i]
    i += 1
  return curr
```

Loop invariants

- A loop invariant is a statement involving the program's variables which is true at the end of each iteration of a loop.
  - Important convention: "the end of the 0th iteration" $\equiv$ the state of the program immediately before the first iteration
- There are lots of candidates. Which should we prove? Whichever ones we need to prove the program correct.
  - For correctness proofs, loop invariant will often be a conjunction of several (unrelated) facts needed for different reasons. (See A2 Q3 starter.)

What about imax?

```
curr = A[0]
i = 1
while i < len(A):
  if A[i] > curr:
    curr = A[i]
  i += 1
```

At the end of iteration j...

- $i_j = j+1$
- $curr_j$ is the largest elt in the list so far, i.e., up to $A[i_j]$

but not including

Formalizing imax loop invariant

```python
def imax(A):
  """Pre: A is non-empty and contains comparable items.
  Post: return the maximum element in A
  """
  curr = A[0]
  i = 1
  while i < len(A):
    if A[i] > curr:
      curr = A[i]
    i += 1
  return curr
```

Inv(j) : at the end of the jth iteration, if one occurs, $curr_j$ is $\geq$ every element in $A[:i_j]$

- x_j denotes the value of variable x at the end of the jth iteration.
  - for simplicity, we can drop subscripts for variables like A whose values never change during execution

Suppose we've proven $\forall j \in \mathbb{N}, \mathrm{Inv}(j)$. Is that enough to show that `imax` is correct?

TODO list

```python
def imax(A):
  curr = A[0]
  i = 1
  while i < len(A):
    if A[i] > curr:
      curr = A[i]
    i += 1
  return curr
```

a)
- Prove that at the end of every iteration j...
  -

b) Prove that imax terminates

c) a + b $\Rightarrow$ postcondition

Lemma 1: loop invariant

```
def imax(A):
  curr = A[0]
  i = 1
  while i < len(A):
    if A[i] > curr:
      curr = A[i]
    i += 1
  return curr
```

$\forall j \in \mathbb{N}$, at the end of the jth iteration, if it exists:

(a) $i_j = j+1$

(b) $curr_j$ is the max of $A[:i_j]$

Base case

$curr_0 = A[0]$

$i_0 = 1 = 0+1$

$A[:i_0] = A[:1] = [A[0]]$

So inv holds.

IS Assume inv holds after j^{th} iter.

Assume we do another iter.

$* \quad i_{j+1} = i_j + 1$

$curr_{j+1} = \max(A[i_j], curr_j)$

$= \max(A[i_j], \max(A[:i_j]))$ # by IH

$= \max(A[:i_j + 1])$

$= \max(A[:i_{j+1}])$ by $*$

by IH $i_j = j+1$

$i_{j+1} = j+1+1$ (a)

Lemma 2: partial correctness

For any valid input, *if the program terminates*, the postcondition is satisfied

$A = [1, 2]$

$A[:100]$

```
1  def imax(A):
2    curr = A[0]
3    i = 1
4    while i < len(A):
5      if A[i] > curr:
6        curr = A[i]
7      i += 1
8    return curr
```

Assume loop exits after some iteration, call it j.

By l4, $i_j \geq \text{len}(A)$

Inv (b) says $curr_j$ is max of $A[:i_j] = A$

Lemma 3: termination

`itermax` terminates on all valid inputs. (We'll leave this to next week.)

```python
def imax(A):
  curr = A[0]
  i = 1
  while i < len(A):
    if A[i] > curr:
      curr = A[i]
    i += 1
  return curr
```

Later

Corollary: `itermax` is correct

(overly pedantic)

Let A be a list satisfying the precondition.

Lemma 2 says that if `itermax(A)` terminates, it returns the right answer.

Lemma 3 says that `itermax(A)` terminates.

Something something *modus ponens*. . . ■

Iterative correctness proofs recipe

1. Prove **loop invariant** by induction. $\forall j \in \mathbb{N}$, if a jth iteration occurs, then at the end of that iteration: ____________________
   1.1 Basis: show that invariant holds *before* entering loop
   1.2 Inductive step: if the invariant holds at the end of iteration j, it also holds at the end of $j+1$ (after another pass through the loop)
   - How to choose what statements to prove? Look ahead to 2.
2. Prove **partial correctness** - if the program terminates, then the postcondition is satisfied. Typical proof pattern:
   2.1 Assume loop terminates after k iterations
   2.2 Therefore, we know the `while` loop condition Q is false.
   2.3 $\neg Q$ tells us something about state of variables at the end of iteration k. Combine with loop invariant from 1 and postcondition follows.
3. Prove **termination**.
   - We'll learn how to do this next week

Return of mergesort

Exercise: brainstorm invariants for merge

[ ] [1, 5, 9] [3, 4, 11]

[1] [1, 5, 9] [3, 4, 11]

[1, 3] [1, 5, 9] [3, 4, 11]

```python
def mergesort(A):
  if len(A) <= 1:
    return A
  m = len(A) // 2
  L1 = mergesort(A[:m])
  L2 = mergesort(A[m:])
  return merge(L1, L2)
```

```python
def merge(A, B):
  """Pre: A and B are sorted lists of numbers.
  Post: return a sorted permutation of A+B
  """
  i = j = 0
  C = []
  while i < len(A) and j < len(B):
    if A[i] <= B[j]:
      C.append(A[i])
      i += 1
    else:
      C.append(B[j])
      j += 1
  return C + A[i:] + B[j:]
```

merge loop invariant

```python
def merge(A, B):
  """Pre: A and B are sorted lists of numbers.
  Post: return a sorted permutation of A+B
  """
  i = j = 0
  C = []
  while i < len(A) and j < len(B):
    if A[i] <= B[j]:
      C.append(A[i])
      i += 1
    else:
      C.append(B[j])
      j += 1
  return C + A[i:] + B[j:]
```

e) every ele in $C_k \leq B[j_k]$

and $\leq A[i_k]$

are these legal indices?

f) $j_k + 1 \leq len(B)$

At the end of each iter k

a) C_k is sorted

b) $len(C_k) = i_k + j_k$

c) $len(C_k) = k$

d) C_k is a permutation of $A[:i_k] + B[:j_k]$

merge loop invariant

```python
def merge(A, B):
  """Pre: A and B are sorted lists of numbers.
  Post: return a sorted permutation of A+B
  """
  i = j = 0
  C = []
  while i < len(A) and j < len(B):
    if A[i] <= B[j]:
      C.append(A[i])
      i += 1
    else:
      C.append(B[j])
      j += 1
  return C + A[i:] + B[j:]
```

merge partial correctness

```
1  def merge(A, B):
2    """Pre: A and B are sorted lists of numbers.
3    Post: return a sorted permutation of A+B
4    """
5    i = j = 0
6    C = []
7    while i < len(A) and j < len(B):
8      if A[i] <= B[j]:
9        C.append(A[i])
10       i += 1
11     else:
12       C.append(B[j])
13       j += 1
14   return C + A[i:] + B[j:]
```

Assume loop exits after iter k

By l7, either $i_k \geq len(A)$ or $j_k \geq len(B)$

$\therefore$ one of $A[i_k:]$ or $B[j_k:]$ is empty.

WLOG, let $A[i_k:] = []$

we return $C_k + [] + B[j_k:]$

PT 1: return val is sorted

- by inv (a), C_k is sorted ✓
- by pre $B[j_k:]$ is sorted
- by inv (e) everything in $C_k \leq B[j_k:]$

merge partial correctness

```python
def merge(A, B):
  """Pre: A and B are sorted lists of numbers.
  Post: return a sorted permutation of A+B
  """
  i = j = 0
  C = []
  while i < len(A) and j < len(B):
    if A[i] <= B[j]:
      C.append(A[i])
      i += 1
    else:
      C.append(B[j])
      j += 1
  return C + A[i:] + B[j:]
```