CSC236 winter 2020, week 6: Recursive correctness

Recommended supplementary reading: Chapter 2 Vassos course notes

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

February 10, 2020

Master Theorem proof sketch

$a \in \mathbb{N}^+$ $\quad b \in \mathbb{N}^+ - \{1\}$

A postscript to last week

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^d) \xRightarrow{\text{Master Theorem}} T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_b n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

if $a = b^d$

$*$ becomes

$$\sum_{i=0}^{\log_b n} n^d$$

$$\begin{aligned} T(n) &= n^d + aT(n/b) \\ &= n^d + a((n/b)^d + aT(n/b^2)) \\ &= n^d + a((n/b)^d + a((n/b^2)^d + aT(n/b^3))) \\ &= n^d + a(n/b)^d + a^2(n/b^2)^d + a^3T(n/b^3) \\ &\dots \end{aligned}$$

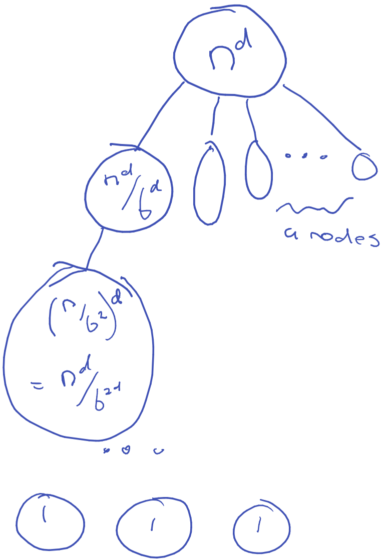
$\dots T(n/n)$

$$= \sum_{i=0}^{\log_b n} a^i (n/b^i)^d = a^i \, n^d / b^{di} \quad *$$

Master Theorem proof sketch

$$T(n) = n^d + a\,T(n/b)$$

Input size	#nodes	Work at level
n	1	$1 \times n^d$
n/b	a	$a \times n^d/b^d$
n/b^2	a^2	
$\dots$	$\dots$	
$1 = n/b^{\log_b n}$	$a^{\log_b n}$	$a^{\log_b n}$



Proving correctness

- Formally proving that my code will ‘do the right thing’ on *any* appropriate input
- Contrast with unit testing: verifies that my code does the right thing on certain specific inputs
  - “Beware of bugs in the above code; I have only proved it correct, not tried it.” - Donald Knuth
- Do people do this in the real world? Yes! (Sometimes)
  - Think of code that runs on medical devices or airplanes

Example: factorial

```python
1 def fact(n):
2     """Return n!
3     """
4     if n == 0:
5         return 1
6     return n * fact(n-1)
```

$P(n)$: $fact(n) = n!$

Basis When $n=0$, $fact(n) = 1$ by lines 4-5, so $fact(n) = 0!$

$P(0)$.

IS Let $n \in \mathbb{N}$, assume $P(n)$

Since $n+1 > 0$, $fact(n+1)$ reaches l6

So $fact(n+1) = (n+1) \times fact(n)$

$= (n+1) \times n!$ # by IH

$= (n+1)!$

So $P(n+1)$

A more complex example: max

```
def max(A):
  if len(A) == 1:
    return A[0]
  max_tail = max(A[1:])
  if A[0] > max_tail:
    return A[0]
  else:
    return max_tail
```

def badmax(A):
  if len(A) <= 1:
    return 15

Our first predicate idea. Didn't work out.

$P(n)$: given a list, A, of length n if we append some ele x to A, to make A', then the $\max(A') =$ greater of $\max(A)$ and x

Q: $\forall n \in \mathbb{N}^+\ P(n) \Rightarrow$ "max is correct" ?

A more complex example: `max`

max([]) ⇒ IndexError?

(Actually, turns out to be infinite recursion.)

```python
def max(A):
  if len(A) == 1:
    return A[0]
  max_tail = max(A[1:])
  if A[0] > max_tail:
    return A[0]
  else:
    return max_tail
```

$a = [\,]$

$a[1:]$

$P(n)$: For any list `A` of length `n`, `max(A)` returns the maximum element of that list.

$\text{max}(A) = x$, where $\forall i \in \mathbb{N}, (i < \text{len}(A) \Rightarrow (A[i] \leq x)) \wedge \exists j \in \mathbb{N}, A[j] = x$

- Is $P(0)$ true? No.
- Is $P(2)$ true?

suppose A = [0, "hi"]

Correctness can only be defined relative to some *specifications*

- **Precondition**: For what inputs is the behaviour of my algorithm defined?
  - Note: my algorithm can do *anything* on other inputs (including crash)
- **Postcondition**: What *should* be true after my algorithm terminates?
  - Usually describes the return value. Or, for an 'in-place' algorithm, describes how the inputs have changed.
- 'My algorithm is correct' $\equiv$ Whenever the precondition holds, my algorithm terminates, and the postcondition is true.

(Often these will be given to you along with the algorithm. Sometimes you'll have to come up with reasonable ones to make your proof work.)

Specifications for `max`

```
def max(A):
  if len(A) == 1:
    return A[0]
  max_tail = max(A[1:])
  if A[0] > max_tail:
    return A[0]
  else:
    return max_tail
```

Pre(A): len(A) > 0 and all elements of A are comparable

Post(A): $max(A) = x$ such that $\exists j \; A[j] = x \wedge \forall i \; i < len(A) \Rightarrow A[i] \leq x$

$P(n)$: for any list `A`, $\texttt{len(A)} = n \wedge \text{Pre}(A) \implies \text{Post}(A)$[1]

[1] You don't need to define separate predicates for pre- and post-conditions, but you may find it notationally convenient.

Proving `max` correct, relative to specifications

```
1  def max(A):
2    if len(A) == 1:
3      return A[0]
4    max_tail = max(A[1:])
5    if A[0] > max_tail:
6      return A[0]
7    else:
8      return max_tail
```

IS Let $n \in \mathbb{N}^+$, assume $P(n)$

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

assume $Pre(A)$

Since $n+1 \geq 2$, so we reach $\ell 4$

Since $len(A[1:]) = n$, and $P(n)$

and $Pre(A[1:])$

max_tail = maximum of $A[1:]$

$Post(A[1:])$, i.e. $\exists j \in \mathbb{N}\ A[1:][j] = \text{max\_tail} \wedge \forall i \in \mathbb{N},\ i < n \Rightarrow A[1:][j] \geq A[i]$

Case 1: max_tail $\geq A[0]$

then we return max_tail

and $Post(A)$ holds "by inspection"

Case 2: max_tail $< A[0]$

then we return $A[0]$

- $A[0]$ in A? Yes
- $A[0] \geq$ all eles of A?
  - by IH + transitivity + case, $A[0] \geq$ any ele in $A[1:]$ $\wedge$ $A[0] \geq A[0]$, so $Post(A)$ ...

Proving `max` correct, relative to specifications

```
1 def max(A):
2   if len(A) == 1:
3     return A[0]
4   max_tail = max(A[1:])
5   if A[0] > max_tail:
6     return A[0]
7   else:
8     return max_tail
```

...

thus Post(A) holds in all cases.

So for all A of length n+1, Pre(A) ⇒ Post(A)

thus P(n+1)

Basis

P(0) vacuously true (Pre(A) never holds)

– but who cares?

n=1, let A be a single element list.

by lines 2-3, we return that element

∧ Post(A) holds

So P(1).

Recipe: proving correctness of recursive programs

1. Determine the program's **preconditions** (i.e. valid inputs) and **postconditions** (the 'correct' behaviour)
2. Define predicate $P(n) \approx$ 'for all inputs of size n, precondition $\implies$ postcondition'
3. Use induction to prove $\forall n \in \mathbb{N}, P(n)$
   3.1 Basis: Corresponds to the part of the program where the recursion 'bottoms out' (usually at the start of the function)
   3.2 Inductive step: Prove that, if the program does the right thing on smaller inputs, it does the right thing on the next input size.

   - May use simple induction (if the algorithm reduces problems of size n to $n-1$), or complete induction (e.g. if problems of size n are reduced to problems of size $n/2$)

Divide-and-conquer maximum

Pre:
Post:
P(n):
[as before]

```python
def maximum(A):
  if len(A) == 1:
    return A[0]
  mid = len(A) // 2
  L_max = maximum(A[:mid])
  R_max = maximum(A[mid:])
  if L_max > R_max:
    return L_max
  else:
    return R_max
```

pre and post are the 'API'

they don't include implementation details

Divide-and-conquer maximum

```
1 def maximum(A):
2   if len(A) == 1:
3     return A[0]
4   mid = len(A) // 2
5   L_max = maximum(A[:mid])
6   R_max = maximum(A[mid:])
7   if L_max > R_max:
8     return L_max
9   else:
10    return R_max
```

Proof sketch (complete induction)
Let $n \in \mathbb{N}^+$. Assume $\forall k \in \mathbb{N}, 0 < k < n \implies P(k)$. (IH)
Let A be a list of length n, and assume Pre(A). WTS: Post(A)
Case 1: $n = 1$. Post(A) holds (same reasoning as our previous basis)
Case 2: $n > 1$.
Show that our IH applies to recursive calls on lines 5-6
Use IH and outcome of check on line 7 to show that postcondition holds in either case, whether we reach line 8 or line 10.

Detail: showing that our IH applies to recursive calls

```python
def maximum(A):
  if len(A) == 1:
    return A[0]
  mid = len(A) // 2
  L_max = maximum(A[:mid])
  R_max = maximum(A[mid:])
  if L_max > R_max:
    return L_max
  else:
    return R_max
```

For convenience, let

- L = A[:mid]
- R = A[mid:]

WTS:

- $0 < len(L) < n$
- $0 < len(R) < n$

$mid = n//2 = \lfloor n/2 \rfloor$

$len(L) = mid = \lfloor n/2 \rfloor$

$n \geq 2 \Rightarrow 0 < \lfloor n/2 \rfloor \wedge \lfloor n/2 \rfloor < n$

$Len(R) = n - len(L)$

$= n - \lfloor n/2 \rfloor$

$= \lceil n/2 \rceil$ # by lemma

$0 < \lceil n/2 \rceil < n$

Lemma

$\forall n \in \mathbb{N}, \lfloor n/2 \rfloor + \lceil n/2 \rceil = n$

Proof:

you can use this in your proofs

Case 1: n is even ✓

[Proof was explained orally. See Piazza for a write-up.]

Detail: showing postcondition holds

```python
1  def maximum(A):
2    if len(A) == 1:
3      return A[0]
4    mid = len(A) // 2
5    L_max = maximum(A[:mid])
6    R_max = maximum(A[mid:])
7    if L_max > R_max:
8      return L_max
9    else:
10     return R_max
```

– return value ≥ all eles in A

use IH + result of l7 + transitivity

– return value is in A, follows from IH

$Post(L) \wedge Post(R)$

$Post(A)$

A very silly way to sum a list

```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)
```

$P(n)$: For any list A of length n,
$\text{Pre}(A) \implies \textit{Post}(A)$

Prove $\forall n \in \mathbb{N}, P(n)$ by induction?

$P(n)$: for any list A, of length n

$\text{Pre}(A) \Rightarrow \text{Post}(A)$

$P(n) \overset{?}{\Rightarrow} P(n+1)$

$P(0)$ ✓

Need 2 proofs

1. prove that when A[0] = n,
sum(A) returns n + sum(A[1:])
2. Use 1 to prove overall correctness of fn.

Silly sum

```
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)
```

Proving the correctness of median

```
def median(A):
  """Pre: A is a non-empty list of unique ints.
  Post: returns the median of A."""
  return select(A, len(A)//2)

def select(A, k):
  """Pre: A is a non-empty list of unique ints.
  Post: returns the element that would be at
  index k if A were sorted."""
  pivot = A[0]
  tail = A[1:]
  S = [x for x in tail if x < pivot]
  L = [x for x in tail if x > pivot]
  if len(S) == k:
    return pivot
  elif len(S) > k:
    return select(S, k)
  else:
    return select(L, k - len(S) - 1)
```

Proving the correctness of median

```python
def median(A):
  """Pre: A is a non-empty list of unique ints.
  Post: returns the median of A."""
  return select(A, len(A)//2)

def select(A, k):
  """Pre: A is a non-empty list of unique ints.
  Post: returns the element that would be at
  index k if A were sorted."""
  pivot = A[0]
  tail = A[1:]
  S = [x for x in tail if x < pivot]
  L = [x for x in tail if x > pivot]
  if len(S) == k:
    return pivot
  elif len(S) > k:
    return select(S, k)
  else:
    return select(L, k - len(S) - 1)
```