CS 30: Discrete Math in CS (Winter 2019): Lecture 16

Date: 1st February, 2019 (Friday)

Topic: Proving Programs Correct

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please discuss in Piazza/email errors to deeparnab@dartmouth.edu

---

1. **Extended Euclid Algorithm: Correctness.**

   We return to the Extended Euclid's Algorithm and prove that the algorithm you coded the other day is indeed correct. This is another chance to understand this beautiful fact.

```
1: procedure ExtGCD(a, b) ▷ Assume a ≥ b ≥ 1.
2:     ▷ Returns the GCD of a and b. Also returns x, y such that xa + yb = gcd(a, b)
3:     Divide a by b to get a = bq + r.
4:     if r = 0 then:
5:         return (b, 0, 1)
6:     else:
7:         Let (g, x', y') = ExtGCD(b, r).
8:         return (g, y', x' − y'q)
```

Theorem 1. For any two natural numbers $a \geq b$, the algorithm ExtGCD(a, b) returns (g, x, y) such that (a) $g = \gcd(a, b)$, and (b) $g = xa + yb$.

Proof. We will prove the theorem by (strong) induction. The trickiest part, perhaps, is to figure out what we will be inducting on. That is, what predicate should we fix. So far, the way we have been talking about induction, the predicates have involved only one number n. But now the theorem is about two numbers a, b. How should we proceed?

One way to is to figure out one "measure" which takes two numbers and gives one number; a measure in which we can compare pairs of numbers with. We will choose the sum here. So, we define the following predicate:

For any natural number n, $P(n)$ is true if and only if for *all* pairs of natural numbers a, b such that $a + b = n$, we have that algorithm ExtGCD(a, b) returns (g, x, y) such that (a) $g = \gcd(a, b)$, and (b) $g = xa + yb$.

Remark: *Understand this predicate before proceeding.*

Now note that if we prove $\forall n \in \mathbb{N} : P(n)$ is true, then we would have proved the theorem. Indeed, for any a and b, consider the predicate $P(a + b)$. This would be true (assuming we are successful) which would imply ExtGCD(a, b) works as it should. Now we are in the familiar realm of induction; let us proceed.

Base Case: We first establish $P(1)$ is true. That is, for any two natural numbers a and b such that $a + b = 1$, we have ... Wait, there are no two such numbers, and thus $P(1)$ is *vacuously* true. This case is not interesting, and we need to establish $P(2)$. That is, for any two natural numbers a and b such that $a + b = 2$, we need to show EXTGCD(a, b) behaves like it should. Indeed, the only way two natural numbers can add to 2 is if $a = 1$ and $b = 1$. If you run EXTGCD$(1, 1)$, then you see it returns $(1, 0, 1)$ which is the correct behavior.

Inductive Case: Now fix a natural number $k \geq 2$. Assume $P(2), P(3), \ldots, P(k)$ is true. We intend to show $P(k + 1)$ is true. Let us first again write down what we need to show; we need to show that for any natural numbers a and b such that $a + b = (k + 1)$, the algorithm EXTGCD(a, b) does what it should (as asked for in the theorem). To that end, *fix* any two such numbers a and b with $a + b = (k + 1)$.

Case 1: b divides a. Note $\gcd(a, b) = b$ in this case. In that case, $a = bq + 0$ for some q. The algorithm EXTGCD(a, b) then runs Line 5 and returns $(b, 0, 1)$. Indeed, $b = \gcd(a, b)$, and $b = a \cdot 0 + b \cdot 1$. Thus the behavior of EXTGCD(a, b) is correct.

Case 2: $a = bq + r$ and $0 < r < b$. From the *key GCD fact*, we know $\gcd(a, b) = \gcd(b, r)$. The algorithm EXTGCD(a, b) now proceeds to Line 7. In particular, it obtains (g, x', y') by calling EXTGCD(b, r). Crucially, not $b + r < b + b \leq a + b = (k + 1)$. Thus, $m := b + r$ is strictly less than $(k + 1)$. By the (strong) induction hypothesis, $P(m)$ is true. Therefore, we get

- $g = \gcd(b, r)$ (I1)
- $g = bx' + ry'$ (I2)

The algorithm then *returns* (in Line 8) $(g, y', x' - y'q)$. We need to show this is correct behavior. That is, we need to show

- $g = \gcd(a, b)$. Indeed, this follows from the key GCD fact that $\gcd(a, b) = \gcd(b, r)$ and (I1).
- $g = y'a + (x' - y'q)b$. Indeed, this is true since the RHS is $x'b + y'(a - bq) = x'b + y'r = g$ from (I2).

Therefore, we have proved $P(k + 1)$. And therefore, by strong induction, the correctness of EXTGCD is established.

□

2. **Binary Search**

```
1:  procedure BinSearch(A[1 : n], x) ▷ Assume A[1 : n] is sorted strictly increasing.
2:      ▷ Returns true if x ∈ A, otherwise returns false.
3:      if n = 1 then:
4:          if x = A[1] then:
5:              return true.
6:          else:
7:              return false.
8:      else:
9:          m = ⌊n/2⌋.
10:         if x = A[m] then:
11:             return true.
12:         else if x < A[m] then:
13:             return BinSearch(A[1 : m], x).
14:         else: ▷ That is, x > A[m]
15:             return BinSearch(A[m + 1 : n], x).
```

We say that BinSearch works properly on the input $(A[1 : n], x)$ if it return true when $x \in A$, and returns false otherwise. The correctness of BinSearch amounts to proving the following theorem.

Theorem 2. For any sorted array of numbers $A[1 : n]$ and any number x, BinSearch works properly on $(A[1 : n], x)$.

Proof. Let $P(n)$ be the predicate which is true if for *any* sorted array $A[1 : n]$ of *length* n and any x, BinSearch works properly on the input $(A[1 : n], x)$. We wish to prove $\forall n \in \mathbb{N} : P(n)$. We proceed by induction.

Base Case. Is $P(1)$ true? That is, given any sorted array $A[1 : 1]$ and any x, does BinSearch work properly on $(A[1 : 1], x)$? To answer this, let us fix a sorted array $A[1 : 1]$ and a number x. If x was indeed in the array $A[1 : 1]$, then it must be that $x = A[1]$. Line 5 then tells us that in this case BinSearch does return true. Similarly, if x was not in the array $A[1 : 1]$, then $x \neq A[1]$. Line 7 then tells us that in this case BinSearch does return false. Thus, in both cases the algorithm behaves properly. $P(1)$ is thus established to be true.

Inductive Case. Fix a natural $k \in \mathbb{N}$. The (strong) induction hypothesis is that $P(1), P(2), \ldots, P(k)$ are all true. We now need to prove $A[k + 1]$. For brevity's sake, let us call $N := k + 1$; we wish to prove $P(N)$, and $P(a)$ is true for all $a < N$. And we have $N > 1$.

That is, we need to show given given any sorted array $A[1 : N]$ and any x, BinSearch works properly on $(A[1 : N], x)$. To this end, let us fix a sorted array $A[1 : N]$ and an x.

Case 1: $x \notin A$. In this case, the algorithm should return false. Since $x \notin A$, $x \neq A[m]$. Thus, Line 10 does not run. Furthermore, $x \notin A$, implies $x \notin A[1 : m]$ *and* $x \notin A[m + 1 : N]$. Since both the arrays $A[1 : m]$ and $A[m+1 : N]$ have lengths $\lfloor N/2 \rfloor$ and $\lceil N/2 \rceil$ which are $< N$ for all $N > 1$, by the (strong) induction hypothesis we have that both BinSearch$(A[1 : m], x)$ and BinSearch$(A[m + 1 : N], x)$ return false. Thus, no matter which of Line 13 or Line 15 runs,

the algorithm BINSEARCH$(A[1 : N], x)$ will return false. Thus, in this case, the algorithm works properly.

Case 2: $x \in A$. In this case, there is some $1 \leq j \leq N$ such that $x = A[j]$. If $j = m$, then Line 10 will return true. If $j < m$, then *since the array is sorted* $x = A[j] < A[m]$. Thus, Line 13 will run. Since $x \in A[1 : m]$ and $m = \lfloor N/2 \rfloor < N$, by the (strong) inductive hypothesis, we know that BINSEARCH$(A[1 : m], x)$ will return true. If $j > m$, then *since the array is sorted* $x = A[j] > A[m]$. Thus, Line 15 will run. Since $x \in A[m+1 : N]$ whose length is $\lceil N/2 \rceil < N$, by the (strong) inductive hypothesis, we know that BINSEARCH$(A[m+1 : N], x)$ will return true.

□