PROOF? A SUPPLEMENTARY NOTE FOR CS161

HPH

Abstract. This notes aims towards those who are not familiar with the way to write correctness proofs. In our context, correctness proof serves to convince ourselves why a proposed algorithm works (correct and terminate) for all possible inputs. The moral of the story: correctness proof doesn't follow a format, contrary to some belief, it is not tedious either; rather, it's a piece that is durable against any counterexamples. This is by no means exclusive, and please contact us at Piazza for clarification. The proof presentations here are often more pedantic than needed; for example, one usually does not need to prove program termination unless it is not obvious.

When we want to prove something, it's really claiming that

"{What we know and trust} ("P") $\Rightarrow$ (Statement to be Proof) ("Q")".

1. Some Commonly Used Techniques

Generally:

(1) Direct Proof
(2) Proof by counterexample. A single counterexample itself IS sufficient for a disproof.

Sometimes, we just want to prove $P \Rightarrow Q$ for some formulated statements P, Q. We could use the following:

(1) Prove by contraposition ($\tilde{Q} \Rightarrow \tilde{P}$).
(2) Proof by contradiction. ($\tilde{Q} \Rightarrow$ something that contradicts with P)[1].
(3) If and only if (aka iff): Combining things above. Example:
   - Direct Proof + Contrapositive:
     Q: Suppose n is an integer. Then n is odd iff n^2 is odd.
     A: ($\Rightarrow$): Suppose n is odd, i.e. $n = 2k+1$ for some $k \in \mathbb{Z}$, in which case $n^2 = (2k+1)^2 = 4k^2+4k+1 = 2(2k^2+2k)+1$ where $2k^2 + 2k \in \mathbb{Z}$, thus n^2 is odd.
     ($\Leftarrow$): Suppose n is even, i.e. $n = 2k$ for some $k \in \mathbb{Z}$ then $n^2 = (2k)^2 = 4k^2 = 2(2k^2)$ where $2k^2 \in \mathbb{Z}$, thus n^2 is even. *(here, we are really claiming if n is NOT odd, then n^2 is NOT odd; but since it is clear that being even is equivalent to NOT odd, we omit this in the proof)*

Sometimes, there are more than 2 statements. In fact, there might be INFINITY statements in total. How do we prove them collectively? In general, we first prove (or are given) that one of them is correct. Then, we prove equivalence.

(1) Equivalence:
   (a) Proof by mutual subset relationship: This is usually used for set equality. Example: Some special cases would be magnitude equality of 2 values, or even "if and only if" proof.
   (b) Circular.
      - Given 3 statements S_1, S_2, and S_3. To prove that all of them are equivalent, it suffices to prove that $S_1 \Rightarrow S_2$, $S_2 \Rightarrow S_3$, and $S_3 \Rightarrow S_1$. (so, suppose)

     The special case where there are only 2 statements is precisely the "if and only if" (aka pairwise equivalence) proof.
   (c) Combination: As long there is Hamiltonian path subgraph in the resulting "arrow graph" we're all good! (refer to the last page)
(2) Proof by Enumerations (aka Cases): When all other (clever) ways failed, there is always one idiot-proof method to fall back on. That is, by considering all cases one-by-one.
   (a) Proof by (Simple) Induction: Indeed, this the gist of induction, except that the language of induction generalizes the 'inductive' process - it behaves like a domino where each fall is identical. Formally, by induction, to prove the proposition $P(n)$ for all $n \in \mathbb{Z}_{\geq 0}$, we prove the following (in order):
      (i) The base case: $P(0)$ or $P(1)$ depending on context
      (ii) The inductive step: $P(k-1) \Rightarrow P(k)$ for any $k > 0$(suppose base case is $P(0)$)

     Induction shows up in many forms. For example:
      (i) Loop Invariance:
        Q: Given an array of reals A, provide an algorithm that sorts it in $O(n^2)$ time.
        A: We propose the bubble sort algorithm:

Date: June 24, 2016.

[1]This achieves the same as contraposition, except that it is more general

Algorithm

```
Algorithm bubblesort
Input:
Integer[]: A
Output:
Integer[]: sorted by increasing order

for i ← 1 to A.size - 1 do
  for j ← i + 1 to A.size do
    if A[i] > A[j] then
      tmp ← A[i]
      A[i] = A[j]
      A[j] = tmp
    end if
  end for
end for

return A
```

Correctness Proof:

We prove the sub array $A[1 \dots i]$ is sorted after the i^{th} outer for loop, using induction w.r.t. i, and that it terminates, thus proving that the algorithm works by considering the case of $i = n$.

(A) Initialization (aka Base Case): For $i = 0$, the invariant is respected: the empty sub-array $A[1 \dots 0]$ is trivially sorted.

(B) Maintanence (aka Inductive Step): Given the sub-array $A[1 \dots n-1]$ sorted. Iteration n inserts at position n the smallest of the remaining unsorted elements of $A[n \dots A.size]$, as computed by the inner for loop. $A[1 \dots n-1]$ contains only elements smaller than $A[n \dots A.size]$, and $A[n]$ is smaller than any element in $A[n+1 \dots A.size]$, then $A[1 \dots n]$ is sorted and the invariant is preserved.

(C) Termination (because this is a computer program we want it to stop): The algorithm terminates after the double for loop, and there is no obstruction in any of the loops to prevent it from exiting. (this is really clear, so we do not expect it to be included in 161 proof).

Runtime Proof:

The outer loop and the inner loop both consists of $O(n)$ iterations, resulting in $O(n^2)$ total loops, where the operation within each loop (1 check and at most 1 swap) takes $O(1)$ time. Thus the runtime is $O(n^2) \cdot O(1) = O(n^2)$ as desired.

Occasions in CS161 that involve strong induction include correctness proof for recursion and DP[2]. Usually, the

(b) Proof by Strong Induction: Sometimes, going back one step is not enough - in fact, sometimes it is unclear how we can formulate it in terms of simple induction. In this case we employ a more general form, i.e. strong induction. Formally, by strong induction, to prove the proposition $P(n)$ for all $n \in \mathbb{Z}_{\geq 0}$, we prove the following (in order):

(i) The base case: $P(0)$ or $P(1)$ depending on context

(ii) The inductive step: $\{P(1), \dots, P(k-1)\} \Rightarrow P(k)$ for any $k > 0$ (suppose base case is $P(0)$)

Examples:

- 2-player Nim (taken from Luca Trevisan's CS103): Nim is a family of games played by two players. The game is set up with several of piles of stones. Players take turns removing stones from the piles, such that each move involves removing one or more stones from a single pile. The winner of the game is the player who removes the last stone from play. In other words, if there are no stones available at the start of a player's turn, they have lost the game. Consider games of Nim which begin with two piles with an equal number of stones. Proof that the player who plays second can always win such a game.

  Proof: Let $P(n)$ be "In a game of Nim, if both piles of stones have n stones each and it's the first player's turn, the second player can always win if she plays correctly." We show that $P(n)$ holds for all $n \in \mathbb{N}$, using strong induction on n.

  - Base case : If both piles have 0 stones in them, the first player loses according to the rules of the game. Thus, the second player always plays 'correctly' by doing nothing, and wins, so $P(0)$ holds.
  - Induction hypothesis : Assume that for some nonzero $n \in \mathbb{N}$, $P(i)$ is true for $0 \leq i < n$. This means that in any game of Nim, if both piles of stones have i stones each and it's the first player's turn, the second player can win if she plays correctly, for $0 \leq i < n$.
  - Induction step: We show that $P(n)$ holds. Consider a game of Nim in which there are two piles of stones, A and B, with n stones in each. Without loss of generality, let A be the pile that the first player chooses to remove stones from. The first player must remove k stones from pile A such that $1 \leq k \leq n$, leaving $n-k$ stones in pile A and n stones in pile B. If the second player removes k stones from pile B, this leaves two piles with $n-k$ stones in each. It is now the first player's turn. By the induction hypothesis, the second player can now win this game because there are two piles with $n-k$ stones in each, $0 \leq n-k < n$, and it is the first player's turn. Thus, the second player has a strategy by which to win a game of Nim with n stones, and $P(n)$ holds, completing the induction

[2]Recursive algorithm and DP shares the same correctness proof, the only difference between them is the latter takes advantage of "caching" which saves the trouble (runtime) of recalculation of preceeding cases

- Size of Binomial Heaps: Consider the sequence $\{F_n\}_{n=0}^{\infty}$ with $F_0 = 1$, $F_n = F_{n-1} + \ldots + F_0 + 1$ for all nonnegative integer n, then $F_n = 2^n$ for all $n \in \mathbb{Z}$ [3].
  Proof: The base case (when $n = 0$) holds since $F_0 = 1 = 2^0$. The inductive step holds, since suppose $F_l = 2^l$ for all $l < k$, we have $F_k = (2^{k-1} + \ldots + 1) + 1 = 2^k - 1 + 1 = 2^k$ (since $2^{k-1} + \ldots + 1 = 2 \cdot (2^{k-1} + \ldots + 1) - (2^{k-1} + \ldots + 1) = (2^k + \ldots + 2) - (2^{k-1} + \ldots + 1) = 2^k - 1$), as desired.

In general, the technique of induction does not just apply to the set of (nonnegative) integers, it applies to any well-ordered set (a totally-ordered set with a least element)

(c) Or simply... Combination: After all, there is no reason why we follow any of the following orders in proving correctness:
- $P(1) \to P(2) \to P(3) \to \ldots$ (simple induction)
- $P(1) \to \{P(1), P(2)\}, \to \{P(1), P(2), P(3)\} \to \ldots$ (strong induction)

The above just happens to be a convenient way when applicable.

2. Proof in the Context of CS161

The proof techniques introduced above are just tools in proof-writing (which whole point is to justify why a claim (be it correctness or proof of an algorithm) is irrefutable) in CS161. Generally for a word problem, we do the following:

(1) Formulate the given word problems into entities involved in the correctness proof
(2) Explain why the formulae/expressions claimed corresponds to the given problem statement
(3) Perform the correctness proof and the runtime proof (sometimes these are merged); if relevant, also prove that the algorithm terminates

Quote lecture notes whenever possible. If they are correct as claimed (and checked by you), they're impeccable and save your time. This is why software libraries exist, isn't it? *Anyway, taking CS161 is a mean to expand P, thus make it possible to expand Q, and make Q part of P"*

3. For further information

- http://web.cse.ohio-state.edu/~pouchet/lectures/doc/888.11.algo.6.pdf
- http://theory.stanford.edu/~trevisan/cs103-14/hw2sol.pdf
- Book of Proof by Richard Hammack
- How to Prove It: A Structured Approach by Daniel J. Velleman

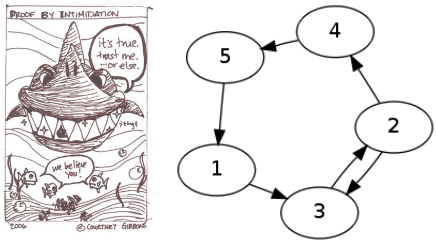

Figure 1. Left: And no... no proof by intimidation...zzz; Right: This is a (finite) graph of statements to be proven, each node represents a statement, it is a valid proof process since it has a hamiltonian cycle as a subgraph: Please identify the Hamiltonian cycle

[3] F_n is the size of the n^{th} Binomial heap (an implementation choice of a meldable priority queue)