

# **Proofs: Who, When, Where, and Why?**

November 5, 2007

## **Different forms of verification:**

1. Experimental
2. Mathematical; formal proof  
— and/or mathematical demonstration from scientific principles
3. Philosophical analysis

**In my experience, most dissertations, and many MS theses, need all 3 forms of verification (each in its appropriate place).**

## Typical examples of proofs in primarily experimental work:

1. Prove a computational complexity result — at least in the extreme cases.
2. Prove what you're testing is really related to what concerns you.
3. If you can verify/evaluate *part* of your work theoretically, that's more convincing than entirely experimental confirmation/evaluation.

## Significance here:

**Experimental demonstration: Limited to the cases you have checked — which you hope — and should argue — are representative.**

**Theoretical proof: You have demonstrated that something must *always* be true.**

## Typical examples of proofs in design work:

1. Prove an algorithm, protocol, or design is correct.

— and if you're using a formal verification system, it will help you to be able to do hand proofs to guide the formal system.

2. Prove properties of a formal model

- to give you extra information assuming we accept the model
- to give you extra grounds for testing the model, both to help you confirm it and to help you try to refute it
- to give extra intuitive or philosophical motivation for the model

## Comparing your work to others':

1. Provide a language to compare your work to other work in the literature.
2. Write a formal proof comparing your work to someone else's:
  - is the other person really studying the same problem?
  - even if you can't theoretically evaluate your work, can you theoretically show it's better than X's?

Better under certain conditions? (If the latter, of course, you need to verify that those conditions are halfway natural.)

## Proof writing 101: Formal proofs.

1. **You need to clear description of pieces.**
2. **Include enough detail to be convincing**
  - but details omitted should be fairly obvious
  - not only intuitively obvious (the *proof* should be obvious)
  - so think in more detail than you write.
3. **Different techniques from different parts of mathematics**
  - Calculus, differential equations
  - Probability and statistics
  - Logic and combinatorics

It is dangerous to assume that, since you know one style, you'll pick up another with no training.

**Example:** Student who needed tools from linear programming.

## 4. Common proof constructs:

**Definition** : A mathematical statement of what you're talking about.

Must be *very precise*.

Of course, needs to correspond to your problem — no grounds shifting.

**And precision is usually in terms of precise mathematical model, not details of code.**

### Standard Vocabulary:

- **Theorem**
- **Observation** or **Proposition**
- **Lemma**
- **Corollary**

(You'll make yourself look foolish if you call all your observations theorems.)

5. **Prove what you need to prove — not something that looks related.**

(The related thing often is a lemma — but is also often irrelevant.)

**Identifying** what you need to prove also involves taking an abstract view of what you're doing.

6. **When in doubt about whether you've done something right,**

try to “move down one level.”

7. **Don't accept proof by bluff** — from friends, professors, or *yourself!*

**Always try to tear down your own proofs.**

8. **When you're stuck, alternate** between trying to prove it and trying to disprove it.

Your failure in one direction may help guide you in the other.

## 9. Proofs by contradiction:

- Be careful about negating precisely!
- Remember de Morgan's laws!

## Related: Proving the Contrapositive

## 10. Proofs by mathematical induction.

**Example:** Proving quicksort correctly sorts arrays.

```
int partition (DATA *array, int start, int finish)
    // PRECONDITION: start < finish
    // Permutes positions start..finish in array
    // Returns a position pivot, start <= pivot <= finish,
    // where in the permuted array,
    // - if start <= n < pivot,    array[n] <= array[pivot]
    // - if finish >= n > pivot,  array[n] > array[pivot]
    // - array[pivot] = ORIGINAL array[start]
    // and leaves rest of array UNCHANGED
    { ... }

void quicksort
    (DATA *array, int start, int finish)
{ if (start < finish)
    { int pivot = partition(array,start,finish);
      quicksort (array, start, pivot-1);
      quicksort (array, pivot+1, finish);
    }
}
```

## Observations on correctness proof:

- Want to prove just that `quicksort` sorts `array`.
- Since function is recursive, expect proof to be inductive.

⚠️ **But by induction on what?** Induction is over integers  $\geq$  some starting point  $n_0$  (often 0 or 1).

(Set theorists do something more general.)

Here there are several numbers to consider, e.g.

- original size of array
- `start`
- `finish`
- `finish + start`
- `finish * start`

Here, **state** that you're doing induction on `finish - start`.

- Recursive call is to *some* smaller lists  
— so need *strong/complete/course of values* induction.

Assume quicksort works for *all* smaller sizes and show it works for this size.

The analogue of the base case of ordinary induction is whichever cases quicksort happens to solve without recursive calls

— here, lists of size 0 and 1.

- **State your inductive hypothesis *carefully***

...and be sure you use *only* the inductive hypothesis

...and prove all you need to prove!

**Subtle distinction:** Am I proving, by induction on **finish - start**, that quicksort sorts

— this fixed array, ... or

— all arrays.

*Usually* if you subtly misstate things, it doesn't matter — *but* sometimes it does. And “usually” isn't good enough.

- Typical with induction: must prove something *stronger*: here that call to **quicksort** sorts *part of array* from **start to finish**

— *and leaves rest of array unchanged.*

## 11. Some common pitfalls

- Be sure you *fully understand* the mathematics you're using. (This might require to take extra courses!)
- Be sure you *state* and *understand* exactly what you're trying to prove.

And be sure that's exactly what your proof claims to prove.

- Think through all the cases
  - don't carelessly miss some
  - and don't just *assume* they're all the same.

(You may *say* they're all the same

— *if* you've checked them all

— and *if* you've shown the proofs of all the (logically) most complicated ones.)

- What's "obvious"?
  - be skeptical of your own intuitions.

## 12. Quantifiers:

- Remember to prove a “for all” (universal) statement with a general proof
- — whereas an example suffices for an “exists” (existential) statement.
- DeMorgan’s laws: the negation of an existential statement is universal, and the negation of a universal statement is existential.
- Although  $\forall x \forall y$ (some property) is logically equivalent to  $\forall y \forall x$ (some property),  
the **proof** may be different.
- Same for  $\exists x \exists y$ (some property) and  $\exists y \exists x$ (some property).
- And, of course,  $\exists x \forall y$ (some property) and  $\forall y \exists x$ (some property) are not logically equivalent.

**¿Which is stronger?**

- Be particularly careful with “alternating quantifiers”

**Example** from formal language theory:

*If a language  $\mathcal{L}$  is regular, then*

*for some integer  $n > 0$*

*for all words  $x \in \mathcal{L}$  of length  $\geq n$ ,*

*there are strings  $u, v, w$ , where*

*$x = uvw$*

*and the length of  $v > 0$*

*and the length of  $uv \leq n$*

*and for all integers  $k \geq 0$*

*$uv^k w \in \mathcal{L}$*

13. **Final pitfall:** Distinguish between a statement and its converse.