Lecture 5: More on Proofs

In this lecture, we look at proofs in more detail.

Proof Strategies

Last time, we saw that proofs are written for humans, in a semi-formal language (one can define fully formal languages for computer-based proofs, but we do not do that here). A proof is a valid argument, in the sense that every step follows logically from the previous steps. Techniques for proofs include direct, indirect and by-contradiction proofs, and case distinctions; a good overall proof strategy is to unpack any definitions (e.g. 'is even'), calculate, and then pack up again.

Compare the following claims over the integers:

- If a is even and b is odd, then $a + b$ is odd.
- $n(n + 1)$ is even.

A note on notation here: to be really precise, one can only prove *statements*, not *terms*. Thus the two example claims here, written symbolically, are

- a is even $\wedge$ b is odd $\vDash a + b$ is odd
- $\vDash n(n + 1)$ is even

You could read the top one as: for all rows in an infinite truth table over all possible (integer) values of a, b, in every row where a is even and b is odd, there $a+b$ is odd. Of course, we can say this without referencing a fictional table as 'For all values of a, b where a is even and b is odd, their sum is odd.'

A statement like the second one that starts with a $\vDash$ means that the right-hand side is a tautology, that is the claim holds for every possible (integer) value of n.

The kinds of statements that appear as claims in proofs are thus *semantic* objects: they make a claim about all possible values of some variables. It is conventional to write such a claim in mathematical English as 'if a is even and b is odd, then the sum $a + b$ is odd' using the words 'if…then', and this is understood to mean a statement rather than a logical formula.

However, proofs are usually *syntactic* objects. In arithmetic, unlike in propositional logic, we cannot make a truth table of all possible values so we are forced to use syntax rules instead. The rules of arithmetic are sound, so anything you can prove with them is also a valid semantically, but they are not complete so there are semantically true statements in arithmetic that one cannot prove with any of the usual techniques.

Going back to our two statements, the first is a claim of an implication: $X \vDash Y$ is the statement that $X \to Y$ is a tautology. The standard proof strategy is to assume the antecedent X (namely that a is even, and b is odd), and then do something, where we are allowed to use our assumptions any time we like such as setting $a = 2k$ (as long as k is not already in use). If we reach the conclusion we want this way, we are done.

The second statement is a claim for all integers, with no further assumptions. In both cases, and for many other proofs, we can start by introducing variables we already know. The phrases 'let', 'assume' and 'pick' (and many others) all mean the same thing, namely 'define a new variable'. So, we could say 'Let a be an even integer' in the first proof, and 'Let n be an integer' in the second

proof. As long as we do not make any restrictions (such as calling something even rather than just an integer) that are not justified by the claim we are trying to prove, then anything we conclude will hold for all values in the domain of our variable.

Thus, 'Let n be an integer. Calculate ...Therefore $n(n+1)$ is even.', if every step is valid, proves a claim about all integers. (Induction, like proof by contradiction, should only be used if a direct proof will not work.) If we started with 'Let a be an even integer.' then we would be allowed to use the fact that a is even later in our proof, but any conclusions we reach would only hold for even integers.

A final note on notation. Many books use $\Rightarrow$ for 'we do a step in a proof' and $\Leftrightarrow$ for 'we do a reversible step in a proof'. In this case, $\Rightarrow$ is our $\vdash$ and $\Leftrightarrow$ is our $\widehat{=}$. However, many other books use $\Rightarrow$ and $\Leftrightarrow$ for our $\rightarrow$ and $\leftrightarrow$. To avoid any ambiguity, and to avoid the common beginner mistake of applying term-operations to statements or vice versa, we do not use double-barred arrows in this unit.

Laws of Logical Reasoning, Part 1

In most proofs, we work on two layers: on the lower layer we have some theory in which we are doing the proof (such as arithmetic, or geometry, or statistics) and on the upper layer, we have logic (to be precise, 'a logic' since there are many different logics).

Of course, if we are doing proofs in logic itself, then we have both a lower and an upper layer that is made of logic, and we should not confuse the two. (For example, this is why we need separate symbols for $a \leftrightarrow b$ which is a term, and $a \equiv b$ which is not. The former lives on the lower layer of logic, and the latter on the upper layer.)

On the upper layer of a proof, we can use laws of logical reasoning, that are independent of whether we are doing a proof in arithmetic, or geometry, or logic itself. These laws come in the form 'if you know some things, then you can conclude other things' and are written in the form $A \vdash B$ to mean 'if you know A you can conclude B' or, if you need more than one assumption, $A, B, C \vdash X$ means 'if you know A, B and C then you can conclude[1] X'.

Two laws of logic are important enough that they get their own names, traditionally written in Latin:

Definition 1 (modus ponens)**.** *Modus ponens* is the law of logic that says: if you know A, and you know $A \rightarrow B$, then you can conclude B. In symbols: $A, A \rightarrow B \vdash B$.

Definition 2 (modus tollens)**.** *Modus tollens* is the law of logic that says: if you know $\neg B$ (that is, B is false) and you know $A \rightarrow B$, then you can conclude $\neg A$. In symbols: $\neg B, A \rightarrow B \vdash \neg A$.

One of our original motivating examples had a microwave with the safety property 'if the door is open, then the magnetron is off'. If a microwave has this property, then you can conclude:

- If you observe the door and it is open, then by modus ponens, you can conclude that the magnetron is off.
- If you observe that the magnetron is on (maybe you are measuring the electric current), then by modus tollens, you can conclude that the door is closed.

Proofs with these laws can often take the form of substituting a specific thing (your A or $\neg B$) into a more general rule $A \rightarrow B$. For example:

[1] $A, B, C \vdash X$ is the upper-layer counterpart to the statement $A \wedge B \wedge C \rightarrow X$ on the lower layer. Like most other uppper-layer constructions, anything with $\vdash$ is not a term.

Claim. For any integer n, $(n(n+1))^2$ is even.
Proof. We have already proven that $n(n+1)$ is always even, and that if x is even, then x^2 is even. By modus ponens, it follows that $(n(n+1))^2$ is even.

Claim. For any integer n, $(n(n+1))^2$ is even.
Proof. We have already proven that $n(n+1)$ is always even, and that if x^2 is odd then x is odd. By modus tollens, since $n(n+1)$ is not odd, therefore neither is $(n(n+1))^2$.

In both these proofs we were using a law of logic to, in effect, substitute $x = n(n+1)$.

Laws of Logical Reasoning, Part 2

A law of reasoning where two premises lead to one conclusion is sometimes called a *syllogism*. Modus ponens and modus tollens are examples of this, but there are more.

Definition 3 (syllogisms)**.** Laws of logical reasoning include:

- **Disjunctive Syllogism**: if we know $A \vee B$, and we know $\neg A$, then we can conclude B. In words: if it's one or the other, and it's not one, then it's the other. This is a special case of the law of the excluded middle, but a common enough form of argument that it gets its own name. In symbols: $A \vee B, \neg A \vdash B$. Of course, the variant $A \vee B, \neg B \vdash A$ also works.
- **Hypothetical Syllogism**: if A implies B, and B implies C, then A implies C. In symbols, $A \to B, B \to C \vdash A \to C$.

For example, if we know that all integers are either odd or even (whether this 'or' is inclusive or exclusive does not matter), and we at some point find that a term a is not odd, then we can conclude that a is even. One way to find that a is not odd would be to assume a is odd, and then derive a contradiction.

A hypothetical syllogism is just modus ponens applied twice, as follows: assume A. By modus ponens and $A \to B$, conclude B. By modus ponens and $B \to C$, conclude C within the scope of A.

Scope in Proofs

If we are trying to prove a claim $A \to B$, we can start the proof with 'assume A'. But, we can also make new assumptions at any point in a proof, and every time we do this we open up a new scope or 'block' in the proof. Anything we conclude in this scope is only valid inside the scope. This lets us write proofs, like programs, in a modular way.

More importantly, it lets us explore the implications of 'What if A were true?', without having to justify why A is true. This is how proof by contradiction works after all: assume A is true, calculate, and if you get a contradiction then you have shown that A is in fact false.

Variables in programs have scopes. How exactly this works depends on the programming language, but in most cases, assuming `print()` prints an integer to standard output, in code like this:

```
int main() {
    int x = 0;
    int y = 0;
    print(x);
    y = addone(x);
    print(x);
    return 0;
}
```

```
int addone(int x) {
    x = x + 1;
    print(x);
    return x;
}
```

will print 0, 1, 0. Inside the scope of `addone()`, the value of x increases to 1, but when this function returns, the last print command refers to the value of x back in the main function, where it is still 0.

Another example:

```
int main() {
    for(int i = 0; i < 4; i++) {
        print(i);
    }
    print(i); // DANGER!
    return 0;
}
```

In C, the variable i comes into scope when the loop starts, but goes out of scope again when the loop ends. So this code should not compile, because there is no i in scope in the last print statement. Other, dynamic lanugages might start running code like this and print 0, 1, 2, 3, and then either throw some kind of exception, or print something like 'null' or 'undefined' when the last statement is reached because it does not refer to a variable in scope.

Proofs work a similar way: at any point in a proof, the things that we 'know' are, more precisely, the things currently *in scope*. In programs, there are different ways of opening a new scope: in C, any time we open a curly brace, we open a new scope, that includes functions, control structures such as 'if' or 'for', or 'naked' blocks that you can use wherever a statement is allowed in the language[2].

Definition 4 (scope)**.** In a mathematical proof, you start a new scope whenever you introduce a variable with the word *assume*[a]. You close a scope when this is clear from the context[b].

When you close a scope, all the facts that you concluded inside the scope go out of scope again and you cannot refer back to them later in the proof, but you do gain the new fact $A \vdash B$ (or $A \to B$ if you prefer) where A was the assumption that you opened the scope with, and B is the last statement before you closed the scope.

[a] Or one of its synonyms like *let, pick, define, take*.

[b] Some proof systems, like natural deduction that we will look at soon, need you to make explicit when you are closing a scope. As the writer of a proof, it is always your duty to the readers to be clear, including pointing out when you are closing a scope and this is not obvious.

[2] This might not, however, be allowed in your C programming unit. Ask the unit director before trying this.

For example, for the proof that $n(n+1)$ is always even, with explicit scopes shown by indentation:

1 | We prove that $n(n+1)$ is even by case distinction.
2 | Assume that n is even.
3 | Then $n = 2k$ for some k.
4 | Then $n(n+1) = 2k(2k+1) = 2(2k^2+k)$, which is even.
5 | So $n(n+1)$ is even.
6 | Therefore, when n is even, it follows that $n(n+1)$ is even.
7 | Assume that n is odd.
8 | Then $n = 2k+1$ for some k.
9 | Then $n(n+1) = (2k+1)(2k+2) = 2(k+1)(2k+1)$, which is even.
10 | So $n(n+1)$ is even.
11 | Therefore, when n is odd, it follows that $n(n+1)$ is even.
12 | By case distinction, it follows that $n(n+1)$ is always even.

Each time we make an assumption, we open a new scope, and any conclusions we draw hold only in that scope. The assumption 'n is even' on line 2 lets us conclude inside that '$n(n+1)$ is even' on line 5, but only inside the scope of the assumption on line 2. When we close the scope on line 6, we get 'n even $\vdash n(n+1)$ even'.

Lines 5 and 10 are identical, but they live in different scopes: one exists to produce line 6, and the other to produce line 11 'n odd $\vdash n(n+1)$ even' which is different from line 6.

In line 12, we can do the case distinction (the assumptions from lines 2 and 7 cover all integers).

It is completely fine here to use the variable k twice in lines 3 and 8, as the scope for the first k closes before the second one opens.

Nested Scopes

We can open scopes within other scopes:

Claim. If a is even and b is odd, then ab is even.
Proof.

1 | Assume that a is even.
2 | Then there is a k such that $a = 2k$.
3 | Assume that b is odd.
4 | Then there is a m such that $b = 2m+1$.
5 | Then $ab = (2k)(2m+1) = 2(2km+k)$.
6 | Therefore, ab is even.
7 | Therefore, if b is odd then ab is even.
8 | Therefore, if a is even and b is odd, then ab is even.

Here we open the scope 'b is odd' inside the scope 'a is even', since we need both assumptions to conclude that ab is even using this approach. We cannot re-use k for b in line 4 as there is still a k from line 2 in scope[3], so we use m.

In line 7, the claim 'if b is odd then ab is even' does not hold in general (it is false if a is odd too). It only holds within the scope of 'a is even', but that is enough for our proof.

[3] Some programming languages allow this, and call it 'shadowing'. It causes more problems than it solves, and is generally considered bad design.

When we open a scope with 'assume A' and close with B:

- If B is a term, we can conclude $A \vdash B$, or $A \to B$ if we prefer.
- If B is not a term but a statement of the form $X \vdash Y$, then we can conclude $A \wedge X \vdash Y$, or $A \wedge X \to Y$ if we prefer.

In the above proof, line 7 closes the inner scope yielding 'b odd $\vdash$ ab even', and line 8 closes the outer scope which started with 'a even' and ended with line 7, so what we have proven is that 'a even $\wedge$ b odd $\vdash$ ab even'.

As we explained earlier, unless we are being especially pedantic, we can write this in English as 'if a is even and b is odd, then ab is even', although we are really talking about a statement not a term.

Sometimes, we need nested scopes; at other times, they are a matter of style. It is fine to open a scope with an assumption of the form 'assume A and B', or even to open a single scope with more than one assumption (which means the same thing as the logical $\wedge$ of all the assumptions). But, if we want to be really formal, then we have to take the 'A and B' assumption apart inside the scope if we want just A at some point.

Laws of Logical Reasoning, Part 3

The following laws apply to using the logical operations $\wedge, \vee$ and $\neg$, explaining how to use them in a precise manner in proofs.

Definition 5 (logical introduction and elimination)**.** The following are laws of logical reasoning.

- If we know A and we know B then we can conclude $A \wedge B$. This is called $\wedge$ introduction. In symbols, $A, B \vdash A \wedge B$.
- If we know $A \wedge B$ then we can conclude A. We can also conclude B. This is called $\wedge$ elimination. In symbols, $A \wedge B \vdash A$ and $A \wedge B \vdash B$.
- If we know A, then for any B we like, we can conclude $A \vee B$, or if we prefer, $B \vee A$. This is called $\vee$ introduction[a]. In symbols, $A \vdash A \vee B$ and $A \vdash B \vee A$.
- If we know $A \vee B$, and we know $A \vdash C$ and $B \vdash C$, then we can conlcude C. This is called $\vee$ elimination[b], in symbols $(A \vee B, A \vdash C, B \vdash C) \vdash C$.
- If we know that A is false, we can conclude $\neg A$. This is called $\neg$ introduction, in symbols, $(A \vdash F) \vdash \neg A$.
- If we know (in some scope) that both A and $\neg A$ are true, then we have a contradiction. This is called $\neg$ elimination. Technically, we can now conclude anything we like, in symbols, $A \neg A \vdash B$ where B is a free term.

[a] A term like the B in $\vee$ introduction that appears only on the right-hand side of a $\vdash$ statement is called a free term. When we use the rule, we can replace it with anything we like.

[b] This is an example of a law of reasoning where some of the assumptions are laws themselves, rather than terms.

If we conclude both A and $\neg A$, surely that is a sign that we have made a mistake? No. We may just have reashed this conclusion within a particular scope. This is what we want to do in a proof by contradiction: we are trying to show for example that A is false, so we assume it is true, calculate to get a contradiction, and then we have shown that A is false.

Example of $\wedge$ introduction

With these rules, we can write the last proof with only a single scope:

Claim. If a is even and b is odd, then ab is even.
Proof.

1 Assume that a is even and b is odd.
2 Using $\wedge$ elimination, we know that a is even.
3 By the definition of even, there is a k such that $a = 2k$.
4 Using $\wedge$ elimination, we know that b is odd.
5 By the definition of odd, there is a m such that $b = 2m + 1$.
6 Then $ab = (2k)(2m + 1) = 2(2km + k)$.
7 As this matches the definition of even, we conclude that ab is even.
8 Therefore, if a is even and b is odd, then ab is even.

Inside the scope 'a even $\wedge$ b odd', if we are being really pedantic, we do not directly have access to the statement 'a even', but we can derive it from the assumption if we need to with $\wedge$ elimination. Most of the time, we do not have to worry about this level of detail, unless we are trying to write a proof that can be checked by a computer.

Example of $\vee$ elimination

The $\vee$ elimination rule is a formal statement of case distinction: if we know $A \vee B$ is true (for example, 'n is odd or n is even'), and we can show that A implies some C (for example, $n(n + 1)$ is even) and B also implies the same C, then C must be true generally — or at least, within the scope we are currently operating. A slightly more formal way of writing said proof:

1 Let n be any integer.
2 Assume that n is even.
3 Then $n = 2k$ for some k, from the defintion of even.
4 Then $n(n + 1) = 2k(2k + 1) = 2(2k^2 + 2k)$.
5 Then $n(n + 1)$ is even, from the definition of even.
6 Therefore, n even $\vdash n(n + 1)$ even.
7 Assume that n is odd.
8 Then $n = 2k + 1$ for some k, from the definition of odd.
9 Then $n(n + 1) = (2k + 1)(2k + 2) = 2((k + 1)(2k + 1))$.
10 Then $n(n + 1)$ is even, from the definition of even.
11 Therefore, n odd $\vdash n(n + 1)$ even.
12 All integers are even or odd, so we know (n even $\vee$ n odd).
13 By $\vee$ elimination on line 12 using lines 6 and 11, we conclude $n(n + 1)$ even.
14 This proves that $n(n + 1)$ is even.

The completely formal way to do a case distinction is first to show, as in line 12, that the logical $\vee$ of the cases we consider covers all possibilities, then to eliminate said $\vee$ as in line 13 by citing a proof of each case (the law as stated only works for exactly two cases, but one can show it works for any finite number of cases by induction). Line 14 is just 'housekeeping', to close the scope that we opened in line 1.

Natural Deduction

Natural deduction is a method to do proofs in logic itself (rather than arithmetic), using purely syntactic techniques. In natural deduction, a statement of the form $A \vdash B$ is called a *sequent*, pronounced 'derives'. Closing a scope in natural deduction is called a 'discharge' and it produces a sequent where A is the assumption that opened the last scope, and B is the last statement inside that scope. You cannot close a scope in natural deduction if the last line inside produced a sequent rather than a term (that is, you cannot close two scopes in a row).

The only things you can do on a line of a proof in natural deduction are: open a scope by assuming a single term (that can, of course, include an $\wedge$); close a scope to produce a sequent, or apply a law of natural deduction. These are essentially the laws from the last defintion, plus laws to introduce and eliminate $\rightarrow$ operators.

Claim. $A \wedge B \vdash B \wedge A$ (since this still holds when you swap A, B, this proves $A \wedge B \mathrel{\widehat{=}} B \wedge A$, that is the commutative law for $\wedge$).

Proof.

1	Assume $A \wedge B$	
2	$\quad A$	using $\wedge$ elimination on 1
3	$\quad B$	using $\wedge$ elimination on 1
4	$\quad B \wedge A$	using $\wedge$ introduction on 3, 2 (note the order)
5	$A \wedge B \vdash B \wedge A$	discharge

Another example that shows nested scopes:

Claim. $A \vdash \neg\neg A$. This is one half of the proof that $A \mathrel{\widehat{=}} \neg\neg A$.

Proof.

1	Assume A	
2	$\quad$ Assume $\neg A$	
3	$\quad\quad F$	using $\neg$ elimination on lines 1, 2
4	$\quad \neg A \vdash F$	discharge
5	$\quad \neg\neg A$	$\neg$ introduction on line 4
6	$A \vdash \neg\neg A$	discharge

This proof should win some kind of award for the least intuitive proof strategy! We start by assuming A because that is the antecedent of what we are trying to prove, so far so good. We then open a nested scope assuming $\neg A$, which directly contradicts the outer scope. That is the point: we are doing a proof by contradiction.

We are trying to get the term $\neg\neg A$. The only way you can get a $\neg$ in natural deduction, without opening a scope, is by $\neg$ introduction, and to get $\neg\neg A$ with this rule we need the sequent $\neg A \vdash F$. So we open a scope to produce this.

We can only conclude F on line 3 because both A (line 1) and $\neg A$ are in scope. In words: assume A. Within the scope of this assumption, if we assume $\neg A$ then we get a contradiction (well, duh!), therefore still within the scopeof A, $\neg A$ is false (still duh!) but that gives us the sequent $\neg A \vdash F$ (still inside the scope of A) that we need for our $\neg$ introduction.