Lecture 1: PROOFS *

10-607 COMPUTATIONAL FOUNDATIONS FOR MACHINE LEARNING

1 Lemmas

One of the most useful tools for organizing a proof is a lemma: we package up a part of our proof so that we can use it repeatedly. To prove a lemma, we start by making some assumptions; call them $\phi_1, \phi_2, \ldots, \phi_k$. We proceed from these assumptions, using them along with any other known facts to reach some conclusion ψ. Finally, we package up our lemma: we conclude

$$\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_k \rightarrow \psi$$

We then get to use this lemma as often as we want later on.

For example, here's a proof of associativity of $\wedge$:

1. Assume $(a \wedge b) \wedge c$
   (a) Conclude c from 1 by $\wedge$ elimination
   (b) Conclude $(a \wedge b)$ from 1 by $\wedge$ elimination
   (c) Conclude a from (b) by $\wedge$ elimination
   (d) Conclude b from (b) by $\wedge$ elimination
   (e) Conclude $b \wedge c$ from (a) and (d) by $\wedge$ introduction
   (f) Conclude $a \wedge (b \wedge c)$ from (c) and (e) by $\wedge$ introduction
2. Conclude $[(a \wedge b) \wedge c] \rightarrow [a \wedge (b \wedge c)]$ from 1 and 1.(f)

This recipe for proving a lemma is our introduction rule for $\rightarrow$: it is the way to prove any expression that contains an implication. So, we could alternately write the last line as

Conclude $[(a \wedge b) \wedge c] \rightarrow [a \wedge (b \wedge c)]$ from 1 and 1.(f) by $\rightarrow$ introduction.

Note how we used indentation to organize our proof: we indented every line between the assumption and the final statement of the lemma.

1.1 Scoping

When we use an assumption to prove a lemma via an application of $\rightarrow$ introduction, it's called *discharging* the assumption. To help organize our proofs, we will make the convention that each assumption needs to be discharged exactly once; we can use it as many times as we want between the line where we assume it and the line where we discharge it. (We can always make the same assumption more than once, if we want to discharge it more than once.)

*Compiled on Thursday 14th September, 2023 at 18:45

The region where we can use an assumption is called its *scope*. We will restrict scopes to be *nested*: that is, we must discharge assumptions in the reverse order of when we make them. This nesting of scopes is what lets us use indentation to mark out our applications of $\rightarrow$ introduction. And, it makes proof organization much clearer, since it helps us keep track of exactly which assumptions a lemma depends on; without scoping rules like this we'd have to manually trace back to see which assumptions we used in deriving each lemma.

Note that we can discharge several assumptions at once, if we are proving a lemma like $a \wedge b \rightarrow c$. The rule is always that we collect all of the assumptions we're discharging, connect them using $\wedge$, and make them into the left-hand side of the new implication. We still have to follow scoping, which means that we must use only the most recent not-yet-discharged assumptions; but we can choose how many of these to use.

The resulting lemma has its own scope: we can continue to use it until the end of the scope where we proved it. That is, suppose we have a proof with the following outline:

1. Assume something
    (a) lorem ipsum...
    (b) lorem ipsum...
    (c) Assume something else
        i. lorem ipsum...
        ii. lorem ipsum...
    (d) Now we have a lemma
    (e) lorem ipsum...
    (f) lorem ipsum...
2. Now the enclosing scope ends
3. lorem ipsum...
4. lorem ipsum...

We can use our lemma in (d) anywhere in the red part of the proof: after the lemma has been proven, up until the enclosing scope ends.

Why? After the enclosing scope ends, its assumptions are no longer available. Since we might have used these assumptions in proving our lemma, the lemma is no longer safe to use. In our example, the assumption in 1 goes out of scope after 2. If we didn't actually use these assumptions, we can always reorganize: we can pull the proof of our lemma out by one scoping level, so that the result remains available longer.

For convenience, we'll also allow "global" assumptions: assumptions we make throughout the proof. These are implicitly or explicitly assumed at the beginning and discharged at the end of our proof. Global assumptions are only a syntactic convenience, and don't change the rules of scoping.

This description of scopes and nesting might remind you of function definitions in a programming language: just as a function definition begins with a header that introduces arguments, a lemma definition begins with a header that introduces assumptions. And just as we can use the function's arguments throughout the function definition block, up until we close the block by using `return`, we can use the lemma's assumptions throughout the lemma definition block, up until we close the block by using $\rightarrow$ introduction.

This similarity is not an accident: lemma definition in proofs plays a role that's completely analogous to function definition in programming languages. In both cases, one of the main purposes is *encapsulation*: we package up a part of a program or a proof so that we can use it repeatedly in the future.

In fact, just as many programming languages organize related function definitions into modules and namespaces, we can optionally organize related lemma definitions the same way. We won't explore the idea of modules or namespaces in proofs any further here, but this sort of organization can become very important when managing larger proofs. .note

1.2 Nested Lemmas

In complicated proofs, we might need several lemmas, possibly even nested inside one another. Here's a slightly larger example that illustrates this idea.

In this example we will use three propositions: PB, J, and Sandwich. Their intended interpretations are "we have peanut butter", "we have jelly," and "we have a sandwich". The proof connects two different ways of making a sandwich.

1. Assume $PB \wedge J \rightarrow$ Sandwich
   (a) Assume PB
      i. Assume J
         A. Conclude $PB \wedge J$ by $\wedge$ introduction from (a) and (a).i
         B. Conclude Sandwich by $\rightarrow$ elimination from 1 and 1.(a).i.A
      ii. Conclude $J \rightarrow$ Sandwich from 1.(a).i and 1.(a).i.B by $\rightarrow$ introduction
   (b) Conclude $PB \rightarrow (J \rightarrow \text{Sandwich})$ from 1.(a) and 1.(a).ii by $\rightarrow$ introduction.
2. Conclude $[PB \wedge J \rightarrow \text{Sandwich}] \rightarrow [PB \rightarrow (J \rightarrow \text{Sandwich})]$ from 1 and 1.(b) by $\rightarrow$ introduction

In English, we can read the last line as: "Suppose that peanut butter and jelly together are enough to make a sandwich. Then if you give me peanut butter, it will mean that jelly is enough to make a sandwich."

There are three nested scopes in this proof: the outermost lemma starts at 1, the next lemma starts at 1.(a), and the innermost lemma starts at 1.(a).i. We close these off in reverse order: the innermost scope concludes at 1.(a).ii, then the middle scope at 1.(b), and the outermost at 2.

If we look at the innermost scope, its job is to prove $J \rightarrow$ Sandwich. In doing so, it gets to use assumptions from enclosing scopes. In particular it uses PB, which was assumed in the middle scope (at 1.(a).i.A), to conclude $PB \wedge J$; and it uses $PB \wedge J \rightarrow$ Sandwich, which was assumed in the outermost scope (at 1), to get Sandwich.

The middle and outer scopes are much simpler than the inner scope: in each of them, we just make an assumption that we'll need later. Note that the innermost scope uses these assumptions but does not discharge them. So these assumptions don't appear in the left-hand side of the innermost scope's lemma. Furthermore, these assumptions remain active in case we wanted to use them for something else in the middle or outer scopes.

The kind of result that we've just proven holds more generally: whenever we have a $\wedge$ on the left-hand side of an implication, like $a \wedge b \wedge c \wedge \ldots \rightarrow z$, we can turn it into a nested sequence of implications, like $a \rightarrow b \rightarrow c \rightarrow \ldots \rightarrow z$. This transformation is called *currying*, after Haskell Curry (a famous logician and programming language theorist who introduced it). For this reason, we'll call the above proof the *curry sandwich*.

Our example illustrates the rules of scoping, which we recap here:

- Assumptions made outside a lemma, in an enclosing scope, are available to use inside the lemma.
- But these assumptions do not get collected in the lemma's result; they'll get collected later on, when we end the enclosing scope.
- Assumptions made at the beginning of a lemma are no longer available once we end the lemma. Instead, only the result of the lemma is available.
- So, when we collect the assumptions for each lemma, we skip any assumptions from sub-lemmas: these assumptions have already been discharged, and are no longer available in the current scope.
- We can keep using the lemma until the enclosing scope ends.

The above proof illustrates a good rule of thumb: always try breaking down a longer proof by focusing on the outermost connective first. We are likely going to need to finish our proof by using the introduction rule for this connective. So, we can figure out what the premises have to be for this rule, and work backward by trying to prove these premises.

In our curry sandwich example, the outermost connective is $\rightarrow$, and in fact the last step of our proof is to use $\rightarrow$ introduction to produce this connective. For this use of $\rightarrow$ introduction, we have to assume the LHS of the implication (which we do at 1), and prove the RHS (which we do at 1.(b). We've now fixed the first line and the last two lines of our proof, and we can proceed to the subgoal of proving $PB \rightarrow (J \rightarrow \text{Sandwich})$. To do so, we can use our heuristic again: the outermost connective is $\rightarrow$ again, so we again want to use $\rightarrow$ introduction. This time we have to assume PB and prove $J \rightarrow$ Sandwich.

Knowledge check
Consider the following proof sketch:

1. Assumption 1
   (a) Statement 1
   (b) Statement 2
   (c) Assumption 2
      i. Statement 3
   (d) Lemma 1
   (e) Statement 4
   (f) Assumption 3
      i. Statement 4
   (g) Lemma 2
2. Lemma 3
3. Statement 5
4. ...

1. **Select all that apply:** In the proof sketch above, which lines can make use of Assumption 2?
   A 1.(a)
   B 1.(c).i
   C 1.(e)
   D 1.(f).i
   E 3
   - **Answer**: Just B. All other options are outside the scope of the assumption.
2. **Select all that apply:** In the proof sketch above, which lines can make use of Lemma 1?
   A 1.(a)
   B 1.(c).i
   C 1.(e)
   D 1.(f).i
   E 3
   - **Answer**: C and D. Because Lemma 1 depends on Assumption 1, once the enclosing scope for that assumption ends, we can no longer safely use Lemma 1. That eliminates option E; options A and B are trivially incorrect as they come before Lemma 1.

2 Negation

The easiest way to handle negation is to define $\neg\phi$ as a shorthand for $\phi \to F$. That is, $\neg\phi$ is true precisely when assuming ϕ would lead to a contradiction. With this definition, to handle $\neg$, we turn $\neg$ into $\to$ and F, which we already have rules for.

If we stop here, we get a system called *intuitionistic* or *constructivist* propositional logic. In this system, all proofs have to be constructive, e.g., there's no notion of proof by contradiction. Counter-intuitively, though, unlike classical logic, a formula of intuitionistic can be neither true nor false. Intuitionistic logic is a fascinating system, since its semantics nicely mirror the semantics of many programming languages; but that's a topic for a different course.

With this definition of negation, we can show for example that $(\neg a \vee b)$ implies $a \to b$:

1. Assume $\neg a \vee b$
   (a) Assume b
   (b) Conclude $b \to b$ by $\to$ introduction from 1.(a)
   (c) Assume a
      i. Assume $\neg a$
         A. Conclude F by modus ponens from 1.(c) and 1.(c).i (recall $\neg a$ is the same as $a \to F$)
         B. Conclude b by ex falso from 1.(c).i.A
      ii. Conclude $\neg a \to b$ by $\to$ introduction from 1.(c).i and 1.(c).i.B
      iii. Conclude b by $\vee$ elimination from 1, 1.(b) and 1.(c).ii
   (d) Conclude $a \to b$ by $\to$ introduction from 1.(c) and 1.(c).iii
2. Conclude $(\neg a \vee b) \to (a \to b)$ by $\to$ introduction from 1 and 1.(d)

Interestingly, we can't show the other direction yet: we can't show that $a \to b$ implies $(\neg a \vee b)$. Instead, to get a complete reasoning system for classical propositional logic, we need one additional rule:

- Double-negation elimination: for any expression ϕ, from $\neg\neg\phi$ we can conclude ϕ.

There are actually several equivalent rules we could add instead: we could pick

- Law of the excluded middle (also called *tertium non datur*, "there is no third choice"): a formula has to be either true or false. For any ϕ, we can conclude $\phi \vee \neg\phi$.
- Contraposition: $\phi \to \psi$ is equivalent to $\neg\psi \to \neg\phi$.
- Pearce's law: for any ϕ and ψ, it holds that $((\phi \to \psi) \to \phi) \to \phi$.
- De Morgan's laws: $\neg(\phi \vee \psi)$ is equivalent to $\neg\phi \wedge \neg\psi$, and $\neg(\phi \wedge \psi)$ is equivalent to $\neg\phi \vee \neg\psi$.

All of these choices are equivalent: from any one of them (plus the other inference rules we introduced above), we can prove all of the others.

To be precise, we can't actually prove an inference rule. We can only prove statements of propositional logic, and inference rules are not statements but tools for working with statements. Instead, we can make a recipe: whenever we were about to use (say) De Morgan's laws, here are the steps we'd take to get the same effect using (say) only double-negation elimination.

Interestingly, once we have double-negation elimination, we no longer need ex falso; that is, anything we can derive with ex falso, we can also derive with double-negation elimination. (The reverse is not true: we cannot use ex falso to eliminate double negatives.) We include ex falso in our set of inference rules even though it is not strictly necessary, since ex falso is important in non-classical logics that don't include double-negation elimination.

3 Resolution

There's one more inference rule that deserves mentioning: the rule of *resolution*. This rule is not as well known as some of the others, but it forms the basis of many automated reasoning systems. Suppose we have two statements of the form

$$\phi \vee \psi \qquad \neg\phi \vee \chi$$

Then resolution lets us conclude

$$\psi \vee \chi$$

That is, we delete ϕ and $\neg\phi$ from our premises, and join what's left with $\vee$. As usual, ϕ, ψ, χ stand for any logical expressions. Note that the formula ϕ appears in both premises: negated in one and not negated in the other.

In resolution, it's common for ϕ to be a *literal* i.e., a single proposition or its negation. We call the two instances of ϕ the *negative* and *positive* literals (meaning with and without the $\neg$ sign.) It's also common for ψ and χ to be *clauses* i.e., disjunctions of several literals.

In this case, the premises of resolution are two clauses. We can apply resolution whenever a positive literal in one clause matches a negative literal in another. This matching literal is called the *resolvent*.

For example, from

$$a \vee b \vee \neg c \vee d \vee \neg e \qquad b \vee \neg d \vee f$$

we can use resolution on d to conclude

$$a \vee b \vee \neg c \vee \neg e \vee f$$

The literals in the conclusion are the union of the literals in the premises, minus d and $\neg d$. Note that we've applied one additional (commonly needed) simplification: we've used idempotence to remove the duplicated literal b, which would otherwise appear twice in the conclusion.

We can treat resolution as a derived inference rule: it follows from the other inference rules that we've already introduced. But resolution is particularly interesting since it can greatly simplify our inference system: we can replace the bulk of any proof with a proof that only uses resolution, so that we don't need general-purpose implementations of any other inference rules. For this reason, resolution forms the basis of many automated reasoning systems.

More precisely, if we start from any set of assumptions in classical propositional logic, we can mechanically transform them into a conjunction of clauses. The clauses in this conjunction form our initial set of known facts.

If we want to conclude a formula ϕ from our assumptions, we look for a proof by contradiction: we assume $\neg\phi$ and try to derive F. So, we translate $\neg\phi$ into a conjunction of clauses, and add these clauses to our set.

Now, if ϕ follows from our assumptions, we are guaranteed to be able to derive F by successive applications of resolution: in each step we pick two clauses from our set that contain matching positive and negative literals, apply resolution, and add the resulting new clause back to our set. If the new clause is empty (a disjunction of zero literals), it is equivalent to F, and our sequence of resolutions constitutes a proof by contradiction that ϕ must hold.

Knowledge check

1. Derive resolution on clauses using the other inference rules we've presented to date.
   - **Hint**: Start by assuming both $\phi \vee \psi$ and $\neg\phi \vee \chi$ and prove $\psi \vee \chi$
   - **Answer**:
     1. Assume $(\phi \vee \psi) \wedge (\neg\phi \vee \chi)$
     2. Conclude $\phi \vee \psi$ by $\wedge$ elimination from 1
     3. Conclude $\neg\phi \vee \chi$ by $\wedge$ elimination from 1
     4. Conclude $\phi \vee \neg\phi$ by law of the excluded middle
        (a). Assume χ
            i. Conclude $\psi \vee \chi$ by $\vee$ introduction from 4.(a)
        (b). Conclude $\chi \rightarrow (\psi \vee \chi)$ by $\rightarrow$ introduction from 4.(a) and 4.(a).i
        (c). Assume ϕ
            i. Assume $\neg\phi$
                A. Conclude F by modus ponens from 4.(c) and 4.(c).i
                B. Conclude χ by ex falso from 4.(c).i.A
                C. Conclude $\psi \vee \chi$ by $\vee$ introduction from 4.(c).i.B
            ii. Conclude $\neg\phi \rightarrow (\psi \vee \chi)$ by $\rightarrow$ introduction from 4.(c).i and 4.(c).i.C
            iii. Conclude $\psi \vee \chi$ by $\vee$ elimination from 3, 4.(b) and 4.(c).ii
        (d). Conclude $\phi \rightarrow (\psi \vee \chi)$ by $\rightarrow$ introduction from 4.(c) and 4.(c).iii
        (e). Assume ψ
            i. Conclude $\psi \vee \chi$ by $\vee$ introduction from 4.(e)
        (f). Conclude $\psi \rightarrow (\psi \vee \chi)$ by $\rightarrow$ introduction from 4.(e) and 4.(e).i
        (g). Assume $\neg\phi$
            i. Assume ϕ
                A. Conclude F by modus ponens from 4.(g) and 4.(g).i
                B. Conclude ψ by ex falso from 4.(g).i.A
                C. Conclude $\psi \vee \chi$ by $\vee$ introduction from 4.(g).i.B
            ii. Conclude $\phi \rightarrow (\psi \vee \chi)$ by $\rightarrow$ introduction from 4.(g).i and 4.(g).i.C
            iii. Conclude $\psi \vee \chi$ by $\vee$ elimination from 2, 4.(f) and 4.(g).ii
        (h). Conclude $\neg\phi \rightarrow (\psi \vee \chi)$ by $\rightarrow$ introduction from 4.(g) and 4.(g).iii
        (i). Conclude $\psi \vee \chi$ by $\vee$ elimination from 4, 4.(d) and 4.(h)
     5. Conclude $[(\phi \vee \psi) \wedge (\neg\phi \vee \chi)] \rightarrow (\psi \vee \chi)$ by $\rightarrow$ introduction from 1 and 4.(i)