Propositional logic

MODERATE

(/edit_new.html#/pages/groups/18)

1. Arguments

Logical arguments are everywhere: from such-and-such assumptions follow these-and-those conclusions. In these notes we will go over *formal* logical arguments. It's important to keep in mind, though, that all arguments (formal or informal) follow similar rules: it's important to

- state assumptions clearly
- describe, for each step, which earlier assumptions or conclusions are inputs, and what conclusion we're now drawing for later use
- organize arguments into digestible (and independently checkable) chunks
- write out everything in enough detail that anyone who comes along later can follow the argument.

Formal logic is the (sometimes painfully) detailed version of the above rules. In practice we typically skip lots of details when we communicate an argument to another human — but it's essential to remember that the details can make or break the correctness of an argument, even if they're invisible.

2. Propositional logic

The reasoning system called propositional logic is at the heart of many other, more complicated reasoning systems. In propositional logic, variables a, b, … (called *propositions* or *predicates*) represent truth values T or F, and connectives $\vee$, $\wedge$, $\neg$, $\rightarrow$ represent OR, AND, NOT, and IMPLIES.

Alternate notations include $\bar{a}$ or $\sim a$ for $\neg a$; $a \supset b$ for $a \rightarrow b$; $a \,\&\, b$, $a \cdot b$, or ab for $a \wedge b$; $a \mid b$ or $a + b$ for $a \vee b$; $\top$ for T; $\bot$ for F. Falsehood F is sometimes also called *absurdity* or *contradiction*.

For readability, we'll often use longer names for variables such as done or $\mathrm{man}(\mathrm{Socrates})$. That last name looks like a function call, and in fact we can give it that semantics in a more expressive system called *first-order logic*, but for now think of it as a really complicated variable name.

1 ¶
2 ¶
3 ¶
4 ¶
5 ¶
6 ¶
7 ¶
8 ¶
9 ¶
10 ¶
11 ¶
12 ¶
13 ¶
14 ¶

3. Inference rules and proofs

To work with statements in propositional logic, we use inference rules. The most famous of these is probably modus ponens: from premises ϕ and $\phi \to \psi$, conclude ψ. The premises ϕ and $\phi \to \psi$ are assumptions or previously proven statements; ϕ and ψ can be single variables or more complex statements containing connectives.

We can put several statements connected by inference rules together into a proof. For example: to say "Socrates is a man; if Socrates is a man then Socrates is mortal; therefore Socrates is mortal", we can write

1. $\text{man}(\text{Socrates})$ [assumption]
2. $\text{man}(\text{Socrates}) \to \text{mortal}(\text{Socrates})$ [assumption]
3. $\text{mortal}(\text{Socrates})$ [modus ponens; 1, 2]

For clarity, we've numbered the statements, and labeled each one with a justification: either that it is an assumption, or how we derived it from previous lines. In the latter case, we listed the inference rule (modus ponens) as well as which statements were the premises (1 and 2).

All statements in a proof must have a justification! The only possible kinds of justifications are the two above: either it's an assumption, or it follows from some earlier statements via an inference rule. Sometimes people omit justifications or skip steps when they are clear from context, but beware: "clear from context" is in the eye of the beholder.

The format above is often called a "two-column proof": we write statements in the left column, and their justifications in the right column.

---

4. Lemmas

The above proof demonstrates the statement

$$\text{man}(\text{Socrates}) \land (\text{man}(\text{Socrates}) \to \text{mortal}(\text{Socrates})) \to \text{mortal}(\text{Socrates})$$

To see why, collect all of the proof's assumptions together with $\land$, take the last line of the proof as the final conclusion, and write "assumptions $\to$ conclusion". (We could actually have taken any line of the proof as the conclusion, but conventionally we put the conclusion last.)

This process — summarizing a proof or part of a proof by collecting together its assumptions and conclusion — is called "implication introduction" or "proving a lemma".

To go with implication introduction, modus ponens is sometimes called "implication elimination."

---

5. Other inference rules

There are lots of useful inference rules besides implication introduction and implication elimination. For example:

- $\wedge$ introduction: if we separately prove ϕ and ψ, then that constitutes a proof of $\phi \wedge \psi$.
- $\wedge$ elimination: from $\phi \wedge \psi$ we can conclude either of ϕ and ψ separately.
- $\vee$ introduction: from ϕ we can conclude $\phi \vee \psi$ for any ψ.
- $\vee$ elimination (also called proof by cases): if we know $\phi \vee \psi$ (the cases) and we have both $\phi \to \chi$ and $\psi \to \chi$ (the case-specific proofs), then we can conclude χ.
- T introduction: we can conclude T from no assumptions.
- F elimination: from F we can conclude an arbitrary formula ϕ. (This rule is sometimes called *ex falso* or *ex falso quodlibet*, from the Latin for "from falsehood, anything.") This rule can be counterintuitive, but one way to think about it is this: we should never be able to prove F, so there's no danger in letting ourselves prove an arbitrary formula given F.
- Associativity: both $\wedge$ and $\vee$ are associative: it doesn't matter how we parenthesize an expression like $a \wedge b \wedge c \wedge d$. (So in fact we often just leave the parentheses out.)
- Distributivity: $\wedge$ and $\vee$ distribute over one another; for example, $a \wedge (b \vee c)$ is equivalent to $(a \wedge b) \vee (a \wedge c)$.
- Commutativity: both $\wedge$ and $\vee$ are commutative (symmetric in the order of their arguments), so we can re-order their arguments however we please. For example, $b \vee c \vee a$ is equivalent to $a \vee b \vee c$.

For conciseness, sometimes we'll allow proofs to use multiple inference rules in each step, and to skip listing out all of them, so long as it is clear which ones we are using. For example, we might rearrange an expression using associativity, distributivity, and symmetry all at once. (But for the duration of the current unit on propositional logic, we'll ask you to use just one inference rule per step.)

---

6. Exercise

Use the above inference rules to prove

$$(a \wedge b) \to (b \wedge a)\,.$$

Write your proof in two-column format: i.e., give an explicit justification for each statement based on previous statements.

Reminder: use *only* the above rules, even if you've learned other useful rules in previous courses.

Exercise, version 2: prove the same statement *without* using the inference rule for commutativity.

---

7. Negation; constructive logic

You may have noticed that none of the inference rules above mention negation. That's because there are actually two interesting ways to handle negation, which we describe next.

In both cases we define $\neg\phi$ as a shorthand for $\phi \to F$. That is, $\neg\phi$ is true precisely when ϕ would lead to a contradiction.

The first way to handle negation is to stop here: we add no new inference rules, and work with negations via the existing rules for implications and falsehood. With this treatment of negation, we get a system called *constructive* or *intuitionistic* propositional logic. Constructive logic is convenient because its semantics nicely mirror those of programming languages — but that's a topic for a different course. (15-317, in case you're interested.)

The one aspect of constructive logic that is sometimes counterintuitive is that it allows formulas to be *neither* true nor false. A formula is true if we can find a proof for it, and false if we can show that it is a contradiction (i.e., find a proof for its negation). But it is perfectly possible that, for some formula ϕ, we can't prove either ϕ or $\neg\phi$.

---

8. Classical logic

The second way to handle negation is *classical* propositional logic, which is the version that most people are familiar with. In classical logic we keep all of the definitions and inference rules above, and add an inference rule that forces statements to be either true or false (even if we don't know which). This rule is called the

- law of the excluded middle: $\phi \vee \neg\phi$ holds for any statement ϕ.

There are in fact many inference rules that we could add instead of (or in addition to) the law of the excluded middle, including

- De Morgan's laws: $\neg(\phi \vee \psi)$ is equivalent to $\neg\phi \wedge \neg\psi$, and $\neg(\phi \wedge \psi)$ is equivalent to $\neg\phi \vee \neg\psi$.
- double negation elimination: $\neg\neg\phi$ is equivalent to ϕ.
- contraposition: $\phi \to \psi$ is equivalent to $\neg\psi \to \neg\phi$.

All of these are equivalent: from any one of these (plus the inference rules above) we can prove 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 rules 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 use to get the same effect using only the law of the excluded middle.)

---

9. Resolution

A useful inference rule — one that is not as well known as some — is *resolution*. Many automated reasoning systems use resolution as their key inference tool. Versions of resolution hold in both classical and constructive logic, but here we focus on the classical version.

Suppose we have two statements of the form

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

Here ϕ, χ, ψ may be any formulas of propositional logic. Note that ϕ appears in both formulas: negated in one, but not the other.

Given the above formulas, resolution lets us conclude

$$\chi \vee \psi\,.$$

That is, we delete ϕ and $\neg\phi$ from our formulas, and join together what's left using $\vee$.

---

10. Resolution on clauses

A common case for resolution is that we have two long *clauses* or *disjunctions of literals*, i.e., formulas where we connect a bunch of literals (variables or negated variables) with $\vee$. (This case is common because it can form the basis of a *complete* reasoning system, i.e., a system that can find a proof for a fact whenever one exists. The details of such a system are beyond this course, but if you're curious, they're covered in 15-780 Graduate AI.)

Given two clauses, we can use resolution exactly when a positive literal in one formula matches a negative literal in the other: we take ϕ to be the positive literal (so that $\neg\phi$ is the negative literal), and take χ and ψ to cover all the remaining literals in our two clauses.

The result is something like this: from

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

we use resolution on the literal d (which appears positively in the first clause and negatively in the second) to conclude

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

The conclusion contains every literal that was in either of the two original clauses, except for d and $\neg d$. Note that we have applied one additional (commonly needed) simplification: the literal b appears in both input clauses, but only shows up once in the output, since $b \vee b \leftrightarrow b$.

---

11. Scoping rules

So far we've looked at simple proofs, where we need at most one lemma. For complicated proofs, we may have several lemmas. These lemmas may even be nested: within a lemma we need a sub-lemma to prove some useful sub-result. For readability, we'll indent the proofs of the lemmas to indicate nesting, as in the following example.

In this example there are three propositions: PB, J, Sandwich. Their intended interpretations are "we have some peanut butter", "we have some jelly", and "we can make a sandwich". The proof connects two different ways of making a sandwich; we'll give a more detailed natural-language interpretation later on.

1. Lemma:
    1. Assume: $PB \wedge J \rightarrow \text{Sandwich}$
    2. Lemma:
        1. Assume: PB
        2. Lemma:
            1. Assume: J
            2. $PB \wedge J$ [from 1.2.1, 1.2.2.1, $\wedge$ -introduction]
            3. Sandwich [from 1.2.2.2, 1, modus ponens]
        3. End lemma: $J \rightarrow \text{Sandwich}$
    3. End lemma: $PB \rightarrow (J \rightarrow \text{Sandwich})$
2. End lemma: $[PB \wedge J \rightarrow \text{Sandwich}] \rightarrow [PB \rightarrow (J \rightarrow \text{Sandwich})]$

We've ended each lemma with a reminder of the result that it proves, using the same rule as above: collect all assumptions made within the lemma, and write "assumptions $\rightarrow$ conclusion."

These nested lemmas obey scoping rules, just like computer programs: each lemma is its own scope, and assumptions are only available inside their scopes (including nested scopes). In more detail:

- Assumptions made outside a lemma, in an enclosing scope, are available to use inside the lemma. But, these assumptions don't get collected in the lemma's result; instead, they will get collected later on when we end the enclosing scope.
- Assumptions made inside 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 a lemma, we skip any assumptions inside sub-lemmas: those assumptions are no longer available in the current scope.

12. Just like computer programs

In fact, we can use Python to define a simple mini-language that lets us construct (and check) proofs: for example, here is a version of the proof above that $(a \wedge b) \rightarrow (b \wedge a)$.

```
startLemma()
assume(And(a, b))
andElim(And(a, b), 0)
andElim(And(a, b), 1)
andIntro(b, a)
endLemma(And(b, a))
```

See the code handout for more details.

---

13. Reading a proof

Let's interpret the proof from above. The overall result is

$$[PB \wedge J \rightarrow \text{Sandwich}] \rightarrow [PB \rightarrow (J \rightarrow \text{Sandwich})]$$

which we can read as: "Suppose that peanut butter and jelly together 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 the proof: the outermost lemma starts at line 1, a second lemma starts at line 1.2, and a final lemma starts at line 1.2.2. Let's look at the innermost scope first (the one that starts at 1.2.2): this lemma proves $J \rightarrow \text{Sandwich}$. In doing so, it gets to use assumptions that were made in enclosing scopes. In particular it uses PB, which was assumed on line 1.2.1, to be able to get $PB \wedge J$; and, it uses $PB \wedge J \rightarrow \text{Sandwich}$, assumed on line 1.1 in the top-level scope, to translate $PB \wedge J$ into Sandwich.

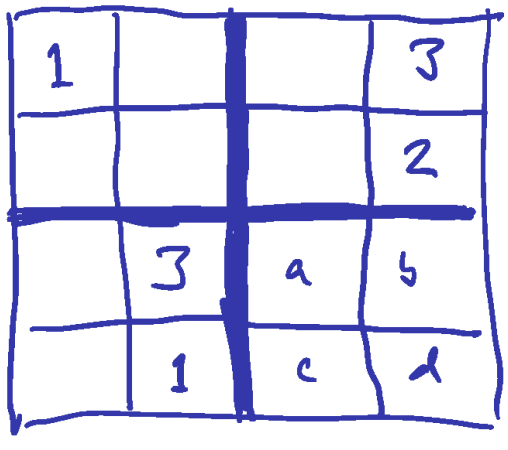
Now let's look at the next scope out (the one that starts at line 1.2): this scope proves $PB \rightarrow (J \rightarrow \text{Sandwich})$. Almost all the work for this happens in the inner lemma that we already looked at: the only thing we need to do in the current scope is assume PB so that this assumption is available for the inner lemma to use.

At the top level scope, the story is the same: our only work in this scope is to make an assumption (line 1.1) that we'll need later on.

---

14. Exercise: mini sudoku

In mini sudoku, the digits 1..4 must appear exactly once in each row, column, and bold-edged 2*2 box of the grid. In the grid below, we've been given five fixed digits (e.g., the 3 in the upper right corner). The squares labeled a, b, c, d are currently blank, and we'd like to figure out how to fill them in:



For example, we know that square d can't contain the digit 2, because there's already a 2 directly above it in the same column.

Fill in the squares a, b, c, d. (Note: no guessing is required.)

Use the rules of propositional logic to write down the constraints that squares a, b, c, d must satisfy. For example, you should write that the digit 1 must appear exactly once in the squares a, b, c, d. (It may take several logical formulas to implement this constraint.) For another example, you should write that the digit 2 can't appear in squares b or d (because of the 2 above them in the same column).

Prove that the solution you gave above is correct, using your formulation of the constraints together with the rules of propositional logic.