12A: Introduction to Logic

Nir Elber

Spring 2022

Contents

Part I

Propositional Logic

Theme 1
Syntax and Semantics

1.1 January 19

Let's go ahead and get going. Today we are hyping the course.

1.1.1 Symbolic Logic

In this course, we are more interested in symbolic logic. More broadly, we are interested in what kind of reasoning is "logical," and we will do this by abstracting what a good argument is.

In symbolic logic, we have lots of symbols for our reasoning words. Here is a table of such symbols.

word	symbol
not	$\neg$
and	$\wedge$
or	$\vee$
if, then	$\to$
for all	$\forall$
there exists	$\exists$

In this course we will be able to give a rigorous definition for what a valid formula is in the language of these symbols. The truth value of a statement will have no dispute.

Example 1.1. The formula

$$(\forall x \mathsf{Red}(x) \vee \exists y \mathsf{Square}(y)) \to \exists z(\mathsf{Red}(z) \wedge \mathsf{Square}(z))$$

asserts that "if everything is red and something is square, then there is something which is both red and square." This is a good, true assertion: that something which is square must also be red, finishing.

1.1.2 Advertisements

This sort of reasoning has applications in lots of fields.

- Logic is a main branch of philosophy. For example, we will study the syllogistic reasoning of Aristotle.[1]

[1] Here is an example of a syllogism: Suppose that all men are mortal, and that Socrates is a man. Then it follows Socrates is mortal.

- Logic is at the base of mathematics, and careful logical reasoning informs foundational mathematics (e.g., Gödel's incompleteness theorems or the independence of the Continuum hypothesis). For example, we will have to understand (basic) mathematical proofs in this course. We will talk about foundational mathematics a bit at the end of the course.
- Logic and its methods (e.g., λ-calculus) impacts how one does computer programming. For a concrete example, logic is used in SQL to give statements for database queries. As another example, formal hardware and software verification comes down to very careful logical analysis.
- One approach to artificial intelligence is by trying to create a machine which spits out true facts from old ones, for which the formal language of first-order logic is quite important.
- The kind of epistemic logic of trying to reason about what people know and do not know is important in game theory and hence has applications to economics. This can quickly get complicated: for example, we might want to keep track of the fact that (e.g., in poker) Player 1 knows that Player 2 knows that Player 3 has an ace card, for this fact might affect Player 2's behavior.
- Linguistics is interested in what sentences mean, for which one had to keep track of formal semantics to determine truth values.
- In cognitive science, how hard it is to understand/learn something turns out to be directly proportional to the length of the shortest logically equivalent propositional formula. In other words, longer formulae are harder to get in one's head.

1.1.3 Logistics

Let's talk about logistics.

- There is a class Piazza, which hopefully will get some use.
- The course outline in the syllabus is more of a guess than a promise; we may get ahead or behind it, but the syllabus will be updated frequently to match.
- There is a textbook. It is more like a math textbook: one is expected to read things multiple times instead of in an English class where one tries to skim as much as possible. While the material is dense, the course has been designed to try to make the course accessible to everyone.
- All reading (including the textbook) will be freely available online.
- It is better to skim the reading before lecture to not completely be lost during lecture. It is also good to do another pass on the reading after lecture.
- There are weekly problem sets, released on Monday (starting next Monday) and due on Sunday midnight. They will be graded via GradeScope, and there are regrade requests (as usual).
- In theory, the problem sets will not depend on a great deal on the lecture Friday before the deadline.
- The class will be curved upwards depending on its difficulty, at the very end.
- Please come to office hours instead of struggling needlessly on one's own.

Next class we will talk about propositional logic.

1.2 January 21

Today we are talking about propositional logic.

1.2.1 Proportions and Connectives

Roughly, propositional logic is about reasoning with propositions with propositional connectives.

Remark 1.2. Propositional logic might also be called sentential logic or boolean logic.

Here are propositions.

Definition 1.3 (Proposition)**.** In this class, a *proposition* will simply be any declarative sentence.

Example 1.4. The sentence "Paris the capital of France" is a proposition.

Non-Example 1.5. The question "Is Paris the capital of France?" is not a proposition.

Remark 1.6. Logic can handle questions, but we will not discuss it. It is called inquisitive logic.

Here are propositional connectives.

Definition 1.7 (Connective)**.** A *propositional connective* is some word or phrase that we can use to build new propositions from old ones.

There are lots of propositional connectives. Have some examples.

Example 1.8. Suppose p and q are propositions. For example, p can be "the thief entered through the back door," and q can be "the thief left through the side door." Then we have the following.

- "p or q" is a proposition.
- "p and q" is a proposition.
- "If p then q" is a proposition.
- "Not p" is a proposition. (Grammatically, "it is not the case that p.")

These are read by replacing p and q with the propositions they represent.

Example 1.9. Propositional connectives do not have to be absolute. For example, "it is more likely that p than it is that q" is a proposition.

1.2.2 Reasoning

So thus far we have propositions and their connectives. How might we reason with them? Here's an example.

Example 1.10. If we happen to know that "p or q," and we know that "it is not the case that p," then q must be true. This is essentially process of elimination; e.g., we might imagine running this argument with p that "the infection is viral" and q that "the infection is bacterial."

The reasoning in the above argument feels quite true, even without making p or q concrete propositions. The reasoning makes us feel good.

Warning 1.11. Suppose we have the following premises.

- p or q.
- p.

It does not follow that q is false. In short, p or q permits both p and q to be true.

Here is another example of reasoning.

Example 1.12. We have the following premises.

- If p, then q.
- Not q.

Then it follows that not p. Concretely, we can set p to be "the reactor is cooling down" and q to be "the blue light is on." Then the fact that the blue light is not on would imply that the reactor is not cooling down.

We do have to be careful with conditionals, however.

Warning 1.13. The premise "if p then q" does not imply that "if q then p."

And here is some reasoning with less concrete connectives.

Example 1.14. Suppose we are given that p is more likely than q. Then it follows that p is more likely than q and r, for any other event r. In essence, trying to make more events happen is harder. For concreteness, think through this argument with the following concrete premises:

- Set p to be "the US will sign the treaty."
- Set q to be "Russia will sign the treaty."
- Set r to be "China will sign the treaty."

Having more people sign the treaty is harder.

Reasoning can be hard, and sometimes our intuition might be wrong. Here is some bad reasoning.

Non-Example 1.15. Suppose we have the following premises.

- If p, then q.
- Not p.

Then it does not follow that q. Concretely, set p to be "the patient is taking her medicine" and q to be "the patient will get better." Then the fact that patient is not taking her medicine does not imply that the patient will not get better: perhaps the patient will get better for some other reason.

The above reasoning is bad because we went from true premises to false conclusions. This is what we try to avoid.

Here is more bad reasoning.

Non-Example 1.16. Suppose we have the following premises.

- It is more likely that p than q.
- It is more likely that p than r.

Then it does not follow that p is more likely than q or r. For example, take p to be the event that a die roll is odd, q to be the event that a die rolls is $\{1, 2\}$, and r to be the vent that a die rolls is $\{3, 4\}$. The probability that q or r exceeds the probability that p in this case.

As an aside, our bad reasoning might still give good premises at the end. The reason that we like good reasoning better is that every single time we will get good premises from good ones; with bad reasoning, we run the risk of getting bad results at the end.

Example 1.17. The argument that the proposition "grass is green" directly implies "the sky is blue" is not valid reasoning because these two propositions have effectively nothing to do with each other. Nevertheless, "the sky is blue" is a true conclusion.

1.2.3 Truth-Functional Connectives

Earlier we gave examples of lots of different propositional connectives. It turns out that we only care about very few of these: the truth-functional propositional connectives.

We need to have a reasonable notion of truth. We adopt the following conventions.

Convention 1.18. In this course, we take the following.

- All propositions are either true or false.
- No proposition is both true and false.

These probably seem obvious, but we need to be careful.

Example 1.19. The following propositions are bad in that they have unclear truth value.

- The proposition "ice cream is delicious" is a proposition of taste (this depends on the person), so we will ignore it.
- "Bob is bald" is a bit vague because "bald" is not well-defined, so we will ignore it.
- "This proposition is false" is self-referential and more or less breaks down truth (if the proposition is true, then the proposition declares its own falsehood), so we will ignore it. Similar is "this proposition is true." (This is known as the liar paradox.)

One way to escape these problems is to simply declare that they are not propositions. We choose to ignore the altogether.

Nevertheless, we go forwards with our notion of truth value.

Definition 1.20 (Truth value)**.** The *truth value* of a proposition is "true" if the proposition is true and "false" if it is false.

Very quickly, we note that the connectives we've talked about come in two classes.

Definition 1.21 (Unary, binary)**.** A propositional connective is *unary* (respectively, *binary*) if and only if it acts on one (respectively, two) proposition.

Example 1.22. The connective "not" is a unary connective. The connective "We know that" is a unary connective.

Definition 1.23. More generally, the *arity* of a connective is the number of propositions the connective acts on.

Example 1.24. The arity of "not" is 1.

Natural language tends to focus on unary and binary connectives.

We are now ready to define what a truth-functional connective is. Here is the definition for unary connectives.

Definition 1.25 (Truth-functional)**.** A unary connective $\#$ is *truth-functional* means that $\#p$ has truth value which is a function of (i.e., is completely determined by) the truth value of p.

Example 1.26. The connective "not" is a truth-functional connective: the truth value of p tells us what the truth value of "not p" is.

Non-Example 1.27. The connective "the police know that" is not a truth-functional connective: a statement being true or false does not immediately tell us whether the police know it. Hopefully if p is false, then the police do not know p; but if p is true, perhaps the police simply do not it yet. The point is that the truth value of "the police know that p" is simply not a function of p.

Here is truth-functionality for binary connectives.

Definition 1.28 (Truth-functional)**.** A binary connective $\#$ is *truth-functional* means that $p\#q$ has truth value which is a function of (i.e., is completely determined by) the truth values of p and q.

Example 1.29. The connective taking the propositions p, q to "p and q" is truth-functional.

1.3 January 24

Okay, welcome back everybody.

1.3.1 Truth-Functionality

Recall the definition.

Definition 1.30 (Truth-functional)**.** A binary connective $\#$ is *truth-functional* means that $p\#q$ has truth value which is a function of (i.e., is completely determined by) the truth values of p and q.

Example 1.31. The connectives "…and …" and "…or …" are both truth-functional.

Non-Example 1.32. The *counterfactual* connective "if it had been the case that p, then it would have been the case that q" is not truth-functional. To see this, note that p being false permits q to be whatever it wants without assigning a truth value depending on p and q. Here are some examples.

- Consider "if I had overslept, then the speaker gave her lecture." In fact, I did not oversleep, and the speaker would give her lecture anyways, so this is p being false and q being true. Here, the counterfactual is in total true.
- Consider "if I had overslept, then I would have arrived late." In fact, I did not oversleep, and in fact I would have arrived on time in this case, so this is p being false and q being true. Here the counterfactual is in total false.

1.3.2 The Material Conditional

Let's talk about conditionals; they are potentially confusing. The motivation here is that "if ... then ..." is used mathematically quite often, so we are going to formalize this. The following is our truth table.

Definition 1.33 (Material conditional ($\to$))**.** Given two propositions p and q, we define the *material conditional* read as "if p then q" is defined by the following truth table.

p	q	if p, then q
T	T	T
T	F	F
F	T	T
F	F	T

Mnemonically, the only way to make an if-then statement false is for the premise to be true and the conclusion to be false.

Warning 1.34. The statement "if I were 90, then I would be king" is a true statement because the premise is false. However, natural language would dictate this would probably not be accepted as true.

Example 1.35. The only way for Goldbach's conjecture (all even numbers greater than 2 are the sum of two prime numbers) to be false is for there to exist an even number which is not the sum of two prime numbers.

Notably, we do not falsify by showing the existence of odd numbers (such as 11) which are not the sum of two prime numbers. Explicitly, the statement

If 11 is an even number greater than 2, then 11 is the sum of two primes.

is true because the premise is false.

Remark 1.36. We can think of the material conditional $p \to q$ as $(\neg p) \vee q$. Namely, as long as p is false or q is true, then $p \to q$ will be true (and conversely).

In this course, we will focus on truth-functional connectives.

1.3.3 Validity and Soundness

For the previous classes, we have been talking about what "good" arguments feel like. The technical version of this is "valid."

Definition 1.37 (Valid)**.** A form of argument is *valid* if, no matter the truth values of the propositions, whenever the premises are true, the conclusion will also be true.

Example 1.38. The following argument form is valid.

1. p or q.
2. Not p.
3. Therefore, q.

Explicitly, all cases where the first two premises hold require q to be true. (In fact, the only way for this to be true is for q to be true and p to be false.)

We can explicitly plug in to the above "argument form" to generate a real, physical argument.

Example 1.39. The following argument is of the above form.

1. Paris is the capital of France or Berlin is the capital of Belgium.
2. It is not the case that Paris is the capital of France.
3. Therefore, Berlin is the capital of Belgium.

Note that this argument is valid, even though the second premise is simply false. Giving false premises provides no guarantees that our conclusion is true, and indeed, the conclusion is false.

Take note that many arguments take the given form.

Warning 1.40. We apply the word "valid" to forms of arguments, not actual arguments. This prevents confusion about the truth-values of the actual premises.

To deal with the above confusion, we have the following definition.

Definition 1.41 (Sound)**.** If an argument is of valid form, and its premises are true (!), then the argument is *sound*.

Importantly, note that soundness is applies to arguments while validity applies to argument forms.

Example 1.42. The following argument is sound.

1. The number 527 is prime, or the number 527 is composite.
2. It is not the case that the number 527 is prime.
3. Therefore, the number 527 is composite.

Non-Example 1.43. The following argument from earlier is not sound.

1. Paris is the capital of France or Berlin is the capital of Belgium.
2. It is not the case that Paris is the capital of France.
3. Therefore, Berlin is the capital of Belgium.

Notably, the second premise here is false, so even though the argument has valid form, the entire argument is no longer sound.

In this case, we will mostly not care too much about soundness because we don't want to consider the truth-values of premises. Our job is to describe how to reason, which means we care more about valid argument forms than actually sound arguments.

Remark 1.44. We do not say that arguments are "true" or "false" because those words belong to propositions in this class. We will only say "sound" or "unsound."

Let's see some more examples.

Non-Example 1.45. The following is an invalid form of argument.

1. It is not the case that (p and q).
2. Therefore, it is not the case that p.

To see that this is invalid, we note that it's possible for p to be true while q is false. Explicitly, take p to be "5 is prime" and q to be "4 is prime." Here, the premise is true (not both 4 and 5 are prime), but the conclusion is false (5 is actually prime.)

However, we could get lucky. For example, in the above argument form, if we swap the roles of p and q, then the conclusion (4 is not prime) will be true. This is an invalid argument still producing true conclusions; the issue is that we are not guaranteed true conclusions from true premises and invalid arguments.

1.3.4 Truth Tables

We would like to have a more mechanical procedure to check the validity of arguments. For this class, we are restricting our attention to the truth-functional case, which means that we directly compute everything depending on the truth values of the propositions. This computation is typically organized into a truth table.

Here is an example.

Exercise 1.46. The following argument form is valid.

1. p or q.
2. It is not the case that p.
3. Therefore, q.

Proof. The only possibilities for p and q are to be true or false in various combinations. We run through all possible cases in the following table.

p	q	p or q	It is not the case that p	q
T	T	T	F	T
T	F	T	F	F
F	T	T	T	T
F	F	F	T	F

From this table we see that the only possibility where all the premises (i.e., both "p and q" and "It is not the case that p") are true is when p is false and q is the highlighted row, and in this case we do see that q is true. Thus, the argument is valid: all cases where the premises are true also happen to have that the conclusion is true. ■

Remark 1.47. An alternate way to do this computation would be to note that the truth table computation comes down to showing the single formula

$$((p \vee q) \wedge \neg p) \to q$$

is always true, independent of the truth values of p and q.

The previous example is a bit misleading because the simple example had only one row in which all premises are true: in general we must check all rows which have the conclusion true. Here is another example.

Exercise 1.48. The following argument form is valid.

1. p.
2. q or r.
3. Therefore, (p and q) or (q and r).

Proof. Here is our truth table.

p	q	r	p	q or r	p and q	p and r	(p and q) or (p and r)
T	T	T	T	T	T	T	T
T	T	F	T	T	T	F	T
T	F	T	T	T	F	T	T
T	F	F	T	F	F	F	F
F	T	T	F	T	F	F	F
F	T	F	F	T	F	F	F
F	F	T	F	T	F	F	F
F	F	F	F	F	F	F	F

The first three rows are the only ones where all the premises are true, and from there we can see that in all these rows the conclusion is true. (In fact, those are exactly the rows where the conclusion is true.) ■

1.4 January 26

Welcome back everyone.

1.4.1 Validity and Truth Tables

Today we're doing some formal propositional logic. It should be fun. Last time we were discussing validity of an argument form, which means that whenever the premises are true, the conclusion will also be true.

Example 1.49. Validity can be counterintuitive. For example, the following argument form is valid.

1. p.
2. $\neg p$.
3. Therefore, q.

This argument form is in fact valid because every time that the premises are true (which happens to be never) also has the conclusion true.

Let's also give an example of an invalid argument.

Exercise 1.50. The following argument is invalid.

1. It is not the case that (p and q).
2. Therefore, it is not the case that p.

Proof. It suffices to give one row of the truth table with true premises but false conclusion. Here it is.

p	q	p and q	not (p and q)	not p
T	F	F	T	F

This finishes because we have found a way to make the premise "not (p and q) true but "not p" false. ■

Importantly, for invalidity it suffices to give the single inconsistent row, though potentially one might have to compute all rows before finding the inconsistent one.

Warning 1.51. However, for the validity check, one does need to write out the full truth table in order to be sure that one has found all cases where the premises are true.

Note that we don't actually care about the rows of the truth table where at least one of the premises is false, but we must include them in order to be sure that we don't care about them. Anyways, in the future we will have other ways to check validity, but for now this is all that we have.

1.4.2 Symbology

Our goal is to define a language of propositional logic. Here are the ideas.

Idea 1.52. To create a formula in propositional logic, we have three ingredients:

1. We will work abstract propositions as "letters" $\{p, q, \ldots\}$.
2. We will change natural language connectors to symbolic ones, essentially to shorten things.
3. We will allow extra connections between formulae by using parentheses in cases of ambiguity.

Note that we have already done the first step above when we moved from concrete arguments to the more abstract argument forms. For the second step, we introduce the following symbols.

English	Symbology	Name
not p	$\neg p$	*negation* of p
p and q	$p \wedge q$	*conjugation* of p and q
p or q	$p \vee q$	*disjunction* of p or q
if p, then q	$p \to q$	*material conditional* with *antecedent* p and *conditional* q
p if and only if q	$p \leftrightarrow q$	*biconditional*

And our third step is more or less automatic after allowing parentheses into our language. For example, $p \to (q \wedge r)$ is a valid formula.

We can translate from English to our formal language with a little of effort.

Example 1.53. Goldbach's conjecture, that

> If n is an integer and n is even and n is greater than 2, then n is the sum of two prime numbers

can be translated into a formula as follows: let p be the proposition that n is an integer, q that n is even, r that n is greater than 2, and s is the sum of two prime numbers. Then the above becomes

$$(p \wedge q \wedge r) \to s.$$

Remark 1.54. Later in life we will even be able to talk about a proposition via predicate logic, in which case we can let $E(n)$ be true if and only if n is even, and $P(n)$ be true if and only if n is prime. Then Goldbach's conjecture becomes

$$\forall n((E(n) \wedge (n > 2)) \to \exists p_1 \exists p_2 (P(p_1) \wedge P(p_2) \wedge (n = p_1 + p_2))).$$

Anyways, with the above discussion, we can now fix the symbols of our language.

Definition 1.55 (Symbols)**.** For propositional logic, we have the following propositional symbols.

- A set of propositional symbols $\{p_1, p_2, \ldots\}$.
- A set of connectives $\{\neg, \wedge, \vee, \to, \leftrightarrow\}$.
- A set of parentheses $\{(,)\}$.

1.4.3 Towards Formulae

We will be defining our formulae by building them up inductively. This inductive process is formalized by "expressions."

Definition 1.56 (Expression)**.** An *expression* is any finite sequence of symbols $\langle s_1, s_2, \ldots, s_n \rangle$ of our language.

Not all of these expressions are good.

Example 1.57. The sequence $\langle \to, \neg, (p \rangle$ is an expression.

We can also "concatenate" expressions together by just stringing one after another. If we have three expressions e_1 and e_2 and e_3, then it is not too hard to see that

$$e_1(e_2e_3) = (e_1e_2)e_3,$$

where the implicit operation is concatenation.

By convention, we will avoid writing the $\langle$ and $\rangle$ as well as commas from our expressions as much as possible.

Example 1.58. We will write "$\to \neg p($" for the expression $\langle \to, \neg, (p \rangle$.

Now, again not all expressions make grammatical sense, so we would like to define which expressions actually have some meaning, and they will be our formulae.

Definition 1.59 (Formula, I)**.** A *formula* is an expression with some meaning. We will define $\mathcal{L}(P)$ to be the set of these well-formed expressions.

Notably we so far still have not described how we are going to build these formulae. Here is a first attempt.

Definition 1.60 (Formula, II)**.** The set of *formulae* $\mathcal{L}(P)$ is defined as a subset of expressions as follows.

- Any propositional p in P makes an "atomic" formula "p."
- Unary connective: if $\varphi \in \mathcal{L}(P)$ is a formula, then $\neg\varphi$ is a formula.
- Binary connectives: if $\varphi_1, \varphi_2 \in \mathcal{L}(P)$ is a formula, then both $(\varphi_1 \wedge \varphi_2)$ and $(\varphi_2 \vee \varphi_2)$ and $(\varphi_1 \to \varphi_2)$ and $(\varphi_1 \leftrightarrow \varphi_2)$ are formulae.
- There are no other formulae than these.

Example 1.61. Here are some examples.

- All propositions are atomic formulae.
- Given a proposition p, "$\neg p$" is a formula. In fact, $\neg\neg p$ will also be a formula.
- Given propositions p and q, $(p \wedge q)$ is a formula. In fact, from here it follows that $(p \to (p \wedge q))$ is a formula. We can continue this process.

The issue with Definition 1.60 is its rigor in the last point: it is not completely clear how to reduce the set of formulae to exclude other formulae. For example, it is a bit annoying to prove that some expression is not a formula.

Example 1.62. All the rules in our definition preserve that there must be a proposition in a formula, so we can immediately say that the expression "$\neg()$" is in fact not a formula.

Next time we will make precise our definition of a formula. Later on we will give them some meaning ("semantics"), but for now we are just arguing about syntax.

1.5 January 28

Here we go.

1.5.1 Formulae

Our story today continues trying to define what a grammatical formula is in our language. Last time we gave an almost-precise definition of formula; today we will formalize it.

The idea is that complex formulae can be built from simpler ones. Namely, we bring in the idea of "construction sequences."

Example 1.63. We can build the formula $(\neg p_1 \to (p_2 \vee p_3))$ by the construction sequence

$$\langle p_1, \neg p_1, p_2, p_3, (p_2 \vee p_3), (\neg p_1 \to (p_2 \vee p_3))\rangle.$$

The point is that each formula in the list is made via some concatenation rule applied to previous formulae in the list.

Here is our definition.

Definition 1.64. A *construction sequence* from a set of letters P is a finite sequence of expressions $\langle \varphi_1, \ldots, \varphi_n \rangle$ satisfying the following conditions.

(a) φ_k may be an atomic formula in P.

(b) If $k < \ell$, then φ_ℓ may be $\neg\varphi_k$.

(c) If $k, \ell < m$, then φ_m may be $(\varphi_k \wedge \varphi_\ell)$ or $(\varphi_k \vee \varphi_\ell)$ or $(\varphi_k \to \varphi_\ell)$ or $(\varphi_k \leftrightarrow \varphi_\ell)$.

We repeat the same example.

Example 1.65. Let's build our previous construction sequence.

- We start with $\langle p_1 \rangle$.
- From here we can get $\langle p_1, \neg p_1 \rangle$.
- From here we may add many atomic formulae, giving $\langle p_1, \neg p_1, p_2, p_3 \rangle$.
- With p_2 and p_3, we can build to $\langle p_1, \neg p_1, p_2, p_3, (p_2 \vee p_3) \rangle$.
- With $\neg p_1$ and $(p_2 \vee p_3)$, we can build the full $\langle p_1, \neg p_1, p_2, p_3, (p_2 \vee p_3), (\neg p_1 \to (p_2 \vee p_3)) \rangle$.

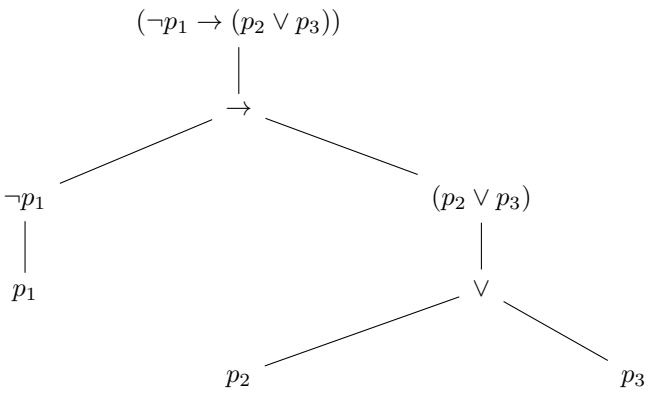
This is a construction sequence for $(\neg p_1 \to (p_2 \vee p_3))$.

The point is that construction sequences let us define formulae.

Definition 1.66 (Formula)**.** An expression φ is a *formula* of $\mathcal{L}(P)$ if and only if it is it an element of some construction sequence from P.

Note that there are infinitely many expressions.

Here is an alternate way to think about this construction sequence: we can build it as a tree.



Observe that this really does convey the same information.

1.5.2 Formulae, Again

Last time we defined what it means to "construct" a formula by manually describing how to construct formulae. We now provide a separate way to do this, which will be helpful for proofs later.

The key here is to view construction steps as "operations" on $\mathcal{L}(P)$. For example, $o_\neg$ is a function which takes the formula φ and returns $\neg\varphi$. In general, for some connector $\#$ such as in $\{\neg, \wedge, \vee, \to, \leftrightarrow\}$, we can write

$$o_\#(\varphi, \varphi') := (\varphi \# \varphi').$$

Now we can provide an "inductive" definition of our formulae.

Definition 1.67 (Formula, again)**.** The set of formulae $\mathcal{L}(P)$ to be a set of expressions satisfying the following.

(a) $\mathcal{L}(P)$ contains all atomic formulae from P.

(b) $\mathcal{L}(P)$ is closed under the operations $o_\neg, o_\wedge, o_\vee, o_\to, o_\leftrightarrow$.

(c) $\mathcal{L}(P)$ is minimal with respect to the properties (a) and (b).

Non-Example 1.68. The set

$$S := \{p, \neg p, \neg\neg p, \ldots\}$$

is closed under $o_\neg$ because appending a $\neg$ to any element in S stays in S. However, this set is not closed under $o_\wedge$ because $p \in S$ but $(p \wedge p) \notin S$.

Remark 1.69. Closure is a nice property. For example, $\mathbb{N}$ is closed under $+$ and $\times$. If we want it to be closed under $-$ (say), we should introduce $\mathbb{Z}$ instead of $\mathbb{N}$.

Note that the conditions (a) and (b) in the definition is too much.

Non-Example 1.70. The set of all possible expressions certainly satisfies (a) and (b), but it has many expressions which we don't want to call formulae, such as $\neg\neg) \, p$.

One possible issue with (c) in Definition 1.67 is that it is not obvious what it means, and once we agree what it means, it's not obvious that $\mathcal{L}(P)$ actually exists as a set. Well, we choose "minimal" to mean that any set S satisfying the properties (a) and (b) immediately implies $\mathcal{L}(P) \subseteq S$.

Thus, to verify that Definition 1.67 does have some $\mathcal{L}(P)$ which exists, we can define

$$\mathcal{L}(P) = \bigcap_{S \text{ satisfies (a), (b)}} S.$$

Then it is not hard to check that $\mathcal{L}(P)$ satisfies (a)—P is a subset of each set we intersect, so $P \subseteq \mathcal{L}(P)$—as well as satisfies (b)—if φ and φ' are two formulae in all sets satisfying (b), then after applying the operation they will still be in all sets satisfying (b).

Anyways, Definition 1.67 is called an "inductive" definition because it will turn out that it gives us an induction: if we want to show that some property holds for all formulae $\mathcal{L}(S)$, we show that the property holds for all propositions and that the property is closed under the operation.

Example 1.71. The natural numbers $\mathbb{N}$ are minimal with respect to $0 \in \mathbb{N}$ and closure under $+1$; its existence can be guaranteed via the same intersection idea. This means that any set containing 0 and closed under $+1$ will contain $\mathbb{N}$.

Remark 1.72. Observe that Definition 1.67 is a bit weird: it's saying an expression φ is a formula if and only if it is a member of each set satisfying (a) and (b). This makes it a little harder to prove that something is a formula because this approach is more "top-down."

Example 1.73. We claim that $(p \wedge q)$ is a formula. Well, let S be some set of expressions satisfying (a) and (b), and we will show $(p \wedge q) \in S$. Then $p, q \in S$ by (a), from which (b) implies $(p \wedge q) \in S$.

Notice how similar the proof in the above example is to actually giving the construction sequence $\langle p, q, (p \wedge q)\rangle$.

1.6 January 31

So class is in-person today. The lecture hall is very large and quite empty.

1.6.1 Formula Induction

Big-picture, we are focusing on symbolic logic. The main point of introducing our formal language right now is that it will help us formally define what rigorous reasoning is and how it works.

Today we're going to discuss induction on formulae, which will be the main technique to show that some property holds for all formulae. Namely, we will be using Definition 1.67 in order to do out proofs.

Remark 1.74. We never actually showed that Definition 1.66 and Definition 1.67 are equivalent, but they are. The annoying thing to do is to check that any set satisfying Definition 1.66 will automatically satisfy Definition 1.67.

Let's do an example proof.

Proposition 1.75. All expressions in $\mathcal{L}(P)$ have the same number of left and right parentheses.

Proof. We proceed by induction. Let S be the set of expressions that have the same number of left and right parentheses, and we show that $\mathcal{L}(P) \subseteq S$. We have the following two checks.

(a) Each atomic expressions $p \in P$ have no parentheses at all, so $p \in S$.

(b) Suppose that $\varphi, \psi \in S$; suppose φ has x left parentheses (and therefore x right parentheses) and ψ has y left parentheses (and therefore y right parentheses). Then we check what happens with our connectives.

- We see that $\neg\varphi$ has the same number left/right parentheses as φ, so these quantities are equal, so $\neg\varphi \in S$.
- We see that $(\varphi \vee \psi)$ and $(\varphi \wedge \psi)$ and $(\varphi \to \psi)$ and $(\varphi \leftrightarrow \psi)$ have $x+y+1$ left and right parentheses, so we are done.

Because $\mathcal{L}(P)$ is the minimal set satisfying (a) and (b), it follows $\mathcal{L}(P) \subseteq S$, so we are done. ∎

An alternate way to see that satisfying (a) and (b) is enough is because we can build any formula in $\mathcal{L}(P)$ by using the steps (a) and (b), by definition of $\mathcal{L}(P)$. The minimality of $\mathcal{L}(P)$ is a way to formalize this intuition.

Remark 1.76. The part where we assume $\varphi, \psi \in S$ before checking $\neg\varphi$ and $(\varphi \vee \psi)$ and its friends is called the "inductive hypothesis." It is very important: if $\varphi = (p \notin S$, then in fact $\neg\varphi = \neg(p \notin S$ as well.

Here is another example: we show that $\neg \to p$ is not a formula, via the following lemma.

Lemma 1.77. Every formula neither starts with $\neg \to$ nor with $\to$.

Proof. Let S be the set of expressions that neither start with $\neg \to$ nor with $\to$. We now induct.

(a) No atomic formula starts with $\neg \to$ nor with $\to$.

(b) If we have a formula φ not starting with $\neg \to$ nor with $\to$, then $\neg\varphi$ will not start with $\neg \to$ because φ did not start with $\to$.

On the other hand, if φ and ψ neither start with $\neg \to$ nor with $\to$, then (for any binary connector $\#$) $(\varphi \# \psi)$ will start with a parenthesis and not with $\neg \to$ nor with $\to$.

Thus, because $\mathcal{L}(P)$ is minimal satisfying (a) and (b), it follows $\mathcal{L}(P) \subseteq S$, so we are done. ■

Here is another result, which we can show without much effort by induction.

Lemma 1.78. If we have letters $P \subseteq Q$, then $\mathcal{L}(P) \subseteq \mathcal{L}(Q)$. We will not prove this here, but it is just another induction.

The point of this is to say that a single formula can belong to multiple languages. For example,

$$(p_1 \wedge p_2) \in \mathcal{L}(\{p_1, p_2\}) \cap \mathcal{L}(\{p_1, p_2, p_3\}).$$

The proof of the lemma is by induction and not hard, so we will omit it.

1.6.2 Subformula

Here is a motivating example.

Example 1.79. The formula $\varphi = (\neg p_1 \to \neg(p_2 \vee p_3))$ has the following subformulae.

$$p_1, \quad \neg p_1, \quad p_2, \quad p_3, \quad (p_2 \vee p_3), \quad \neg(p_2 \vee p_3), \quad (\neg p_1 \to (\neg(p_2 \vee p_3))).$$

The set of proper formulae is the same set except for φ.

Non-Example 1.80. The formula $(p_1 \vee p_2)$ is not a subformula of $(p_1 \wedge p_2)$.

We would like to define a function $\mathrm{sub} : \mathcal{L}(P) \to \mathcal{P}(\mathcal{L}(P))$. For this, we will create a recursive definition for sub.

Definition 1.81 (Subformula)**.** Given a formula $\varphi \in \mathcal{L}(P)$, we define the *subformulae* $\mathrm{sub}(\varphi)$ to be defined as follows.

- $\mathrm{sub}(p) := \{p\}$ for each atomic formula $p \in P$.
- $\mathrm{sub}(\neg\varphi) = \mathrm{sub}(\varphi) \cup \{\neg\varphi\}$.
- $\mathrm{sub}((\varphi \# \psi)) = \mathrm{sub}(\varphi) \cup \mathrm{sub}(\psi) \cup \{(\varphi \# \psi)\}$ for any two formulae $\varphi, \psi \in \mathcal{L}(P)$ and binary connective $\# \in \{\wedge, \vee, \to, \leftrightarrow\}$.

The above definition is called "recursive" because we did not explicitly define it but only how to do this by breaking down connectives. We can show that sub is defined on $\mathcal{L}(P)$ by yet another induction which we will omit.[2]

Let's see some examples.

Example 1.82. We have the following computation.

$$\begin{aligned}
\mathrm{sub}(\neg(p \wedge q)) &= \mathrm{sub}((p \wedge q)) \cup \{\neg(p \wedge q)\} \\
&= \mathrm{sub}(p) \cup \mathrm{sub}(q) \cup \{(p \wedge q)\} \cup \{\neg(p \wedge q)\} \\
&= \{p\} \cup \{q\} \cup \{(p \wedge q)\} \cup \{\neg(p \wedge q)\} \\
&= \{p, q, (p \wedge q), \neg(p \wedge q)\}.
\end{aligned}$$

[2] By definition, sub is defined on atomic formulae, and the domain is closed under connectives, so sub is defined on $\mathcal{L}(P)$.

Example 1.83. We have the following computation.

$$\begin{aligned}\mathrm{sub}((p \wedge p)) &= \mathrm{sub}(p) \cup \mathrm{sub}(p) \cup \{(p \wedge p)\} \\ &= \{p\} \cup \{p\} \cup \{(p \wedge p)\} \\ &= \{p, p \wedge p)\}.\end{aligned}$$

Note that the union of the two sets has killed the duplicate p.

Example 1.84. We have the following computation.

$$\begin{aligned}\mathrm{sub}((\neg p_1 \to p_2)) &= \mathrm{sub}(\neg p_1) \cup \mathrm{sub}(p_2) \cup \{(\neg p_1 \to p_2)\} \\ &= \mathrm{sub}(p_1) \cup \{\neg p_1\} \cup \mathrm{sub}(p_2) \cup \{(\neg p_1 \to p_2)\} \\ &= \{p_1\} \cup \{\neg p_1\} \cup \{p_2\} \cup \{(\neg p_1 \to p_2)\} \\ &= \{p_1, \neg p_1, p_2, (\neg p_1 \to p_2)\}.\end{aligned}$$

It is true that the definition of "subformula" feels intuitively obvious, but we have given the above rigorous definition to introduce the idea of a recursive definition.

Let's discuss the idea of recursion a little more closely because it will come up again.

- Define a set S inductively as the smallest set containing some base objects and closed under some operations. We will also require the following coherence conditions.[3]
  - We will require that the operations never output the same formula. For example, for two formulae φ and ψ, we have
    $$(\varphi \wedge \psi) \neq (\varphi \vee \psi).$$
  - No operation can output a base object. For example, no operation can output an atomic formula because operations will always start with $\neg$ or a parenthesis.
- Then to define a function f inductively, we need to define $f(b)$ for each base object $b \in S$ and provide some function g such that
  $$f(o(a_1, \ldots, a_n)) = g_o(f(a_1), \ldots, f(a_n), a_1, \ldots, a_n)$$
  for each n-ary operation o. For example, we had
  $$\mathrm{sub}(o_\neg(\varphi)) = g_\neg(\mathrm{sub}(\varphi), \varphi),$$
  where $g_\neg(S, \varphi) := S \cup \varphi$.

Let's do another example recursive definition.

Definition 1.85 (Depth)**.** We define the *depth* of a formula $\varphi \in \mathcal{L}(P)$ to be given be a function $\mathrm{depth} : \mathcal{L}(P) \to \mathbb{N}$ defined as followed.

- $\mathrm{depth}(p) = 0$ for each atomic formula $p \in P$.
- $\mathrm{depth}(\neg\varphi) = \mathrm{depth}(\varphi) + 1$ for each $\varphi \in \mathcal{L}(P)$. In other words, adding $\neg$ adds one level of depth.
- $\mathrm{depth}((\varphi \# \psi)) = \max\{\mathrm{depth}(\varphi), \mathrm{depth}(\psi)\} + 1$ for each $\varphi, \psi \in \mathcal{L}(P)$ and binary connective $\#$. In other words, the depth of $(\varphi \# \psi)$ is one more than the larger of the two depths of φ and ψ.

[3] The coherence conditions ensure that the function we create is well-defined.

Example 1.86. We have the following computation.

$$\begin{aligned}\operatorname{depth}((\neg p_1 \to p_2)) &= \max\{\operatorname{depth}(\neg p_1), \operatorname{depth}(p_2)\} + 1 \\ &= \max\{\operatorname{depth}(p_1) + 1, \operatorname{depth}(p_2)\} + 1 \\ &= \max\{0 + 1, 0\} + 1 \\ &= 2.\end{aligned}$$

We can see that this definition is indeed recursive: we computed it as 0 for the basic objects, and then we had

$$f(o_\neg(\varphi)) = g_\neg(f(\varphi), \varphi),$$

where $g_\neg(n, \varphi) = n + 1$, and

$$f(o_\#(\varphi, \psi)) = g_\#(f(\varphi), f(\psi), \varphi, \psi),$$

where $g_\#(n, m, \varphi, \psi) = \max\{n, m\} + 1$.

1.7 February 2

Here we go again.

1.7.1 Semantics for Connectives

Last time we finished our discussion of creating the language of propositional logic. Today we will actually give formulae meaning as a "truth" function of the propositions.

Example 1.87. We hinted for $\neg p$ to mean "not p."

Our goal is to give precise definitions to all of our symbols, in a mathematically precise way.

The way we do this is by "truth-conditional semantics." Namely, we will give meaning to certain formulae by describing when a formula is true or false.

Example 1.88. We can compute the meaning of $\neg p$ by the following truth table.

p	$\neg p$
T	F
F	T

This truth table fully captures what $\neg$ should mean.

Convention 1.89. For the remainder of the class, we will replace "T" with 1 and "F" with 0.

Example 1.90. We have that the truth value of $\neg p$ is $1-$ the truth value of p.

So using numbers will have some utility in this class.

Example 1.91. We have that

p	q	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

In short, we can verify that $p \wedge q$ has truth value equal to the minimum of the truth values of p and q.

We could also say that $p \wedge q$ has truth value equal to the product of the truth values of p and q. However, we do not do this to make better analogy with $\vee$, as follows.

Example 1.92. We have that

p	q	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

In short, we can verify that $p \wedge q$ has truth value equal to the maximum of the truth values of p and q.

Let's wrap up with the last two connectives.

Example 1.93. We have that

p	q	$p \to q$
1	1	1
1	0	0
0	1	1
0	0	1

We can say this as the truth value of $p \to q$ is the indicator for when the truth value of p is less than or equal to the truth value of q.

Example 1.94. We have that

p	q	$p \leftrightarrow q$
1	1	1
1	0	0
0	1	0
0	0	1

Now, we can say this as the truth value of $p \to q$ is the indicator for when the truth value of p is equal to the truth value of q.

1.7.2 Semantics for Formulae

The key idea to extending semantics to all formulae is to use a recursive definition, using the connectives to build arbitrary semantics for formulae. We will need to be somewhat technical for this.

Definition 1.95 (Valuation)**.** A *valuation* V for P is a function $V : P \to \{0, 1\}$ which assigns each proposition $p \in P$ a value of 0 or 1.

Example 1.96. There is a valuation $V : P \to \{0, 1\}$ which gives $V(p) = 1$ for each $p \in P$. There is also a valuation $V : P \to \{0, 1\}$ which gives $V(p) = 0$ for each $p \in P$.

Example 1.97. The following describes a valuation for $P = \{p, q\}$.

p	1
q	0

Remark 1.98. The "valuation" $\hat{V}$ is meant to be an abbreviation of "the truth value of."

Intuitively, a valuation is a description of the world of P: maybe $p_1 \in P$ should be true with p_2 false, maybe something else.

Remark 1.99. If P is finite, then there are only finitely many valuations. Precisely, there are $2^{\#P}$ total valuations: each proposition $p \in P$ has two options (namely, 0 or 1), and we have to make $\#P$ total choices (simultaneously) to determine a unique valuation V.

Now here is our promised idea.

Idea 1.100. Recursion can extend any valuation $V : P \to \{0,1\}$ uniquely to all of $\hat{V} : \mathcal{L}(P) \to \{0,1\}$.

Example 1.101. Once we know $V(p)$ and $V(q)$, we get $V((p \wedge q))$ for free.

Let's give the recursive definition for $\hat{V}$; it works as follows.

- For each atomic formula $p \in P \subseteq \mathcal{L}(P)$, we have $\hat{V}(p) := V(p)$. (This is our base case.)
- We have $\hat{V}(\neg\varphi) := 1 - \hat{V}(\varphi)$.
- We have $\hat{V}((\varphi \wedge \psi)) := \min\{\hat{V}(\varphi), \hat{V}(\psi)\}$.
- We have $\hat{V}((\varphi \vee \psi)) := \max\{\hat{V}(\varphi), \hat{V}(\psi)\}$.
- We have $\hat{V}((\varphi \to \psi)) := 1_{\hat{V}(\varphi) \le \hat{V}(\psi)}$.
- We have $\hat{V}((\varphi \leftrightarrow \psi)) := 1_{\hat{V}(\varphi) = \hat{V}(\psi)}$.

A standard induction can show that the domain of $\hat{V}$ is indeed $\mathcal{L}(P)$. In particular, the domain of $\hat{V}$ contains all the letters $p \in P$ and is closed under our connectives.

Remark 1.102. We could write the above discussion by writing out the full truth tables for each connective, but the above is arguably a cleaner connective.

Now that we've extended V, we can make terminology for when formulae might be true or false.

Definition 1.103 (Satisfies)**.** A valuation $V : P \to \{0,1\}$ *satisfies* $\varphi \in \mathcal{L}(P)$ if and only if $\hat{V}(\varphi) = 1$. We will notate this by $V \models \varphi$.

Intuitively, the world that V describes makes the formula φ true.

Let's do an example.

Exercise 1.104. Fix $P = \{p, q, r\}$ and V a valuation by $V(p) = V(q) = 1$ and $V(r) = 0$. We determine if V satisfies $(p \wedge (r \to q))$.

Proof. We do the computation by hand.

$$\begin{aligned}
\hat{V}(((p \wedge (r \to q)))) &= \min\{\hat{V}(p), \hat{V}((r \to q))\} \\
&= \begin{cases} \min\{\hat{V}(p), 1\} & \hat{V}(r) \le \hat{V}(q), \\ \min\{\hat{V}(p), 0\} & \hat{V}(r) \le \hat{V}(q), \end{cases} \\
&= \min\{\hat{V}(p), 1\} \\
&= \min\{1, 1\} \\
&= \boxed{1}.
\end{aligned}$$

■

The above approach felt more top-down as an evaluation, in contrast to the ore bottom-up a typical truth-table computation, as follows.

p	q	r	$r \to q$	$p \wedge (r \to q)$
1	1	0	1	1

1.7.3 Validity, Again

We would like to use our precise definitions for logic to recreate our definitions for "valid."

We start with a small remark.

Remark 1.105. Suppose $\varphi \in \mathcal{L}(P_1) \cap \mathcal{L}(P_2)$ for some proposition letters P_1 and P_2. (Namely, the letters of φ live in both P_1 and P_2.) Then if $V_1 : P_1 \to \{0,1\}$ and $V_2 : P_2 \to \{0,1\}$ are valuations such that each p present in φ has $V_1(p) = V_2(p)$, then

$$\hat{V}_1(\varphi) = \hat{V}_2(\varphi).$$

To formalize this, we would need a notion of which propositions are present in φ, which we could define as a function recursively. We will not bother to do this here.

Here is our definition of an argument form.

Definition 1.106 (Argument form)**.** An *argument form* in $\mathcal{L}(P)$ is a pair of *premises* $\{\varphi_1, \dots, \varphi_n\} \subseteq \mathcal{L}(P)$ and a *conclusion* $\psi \in \mathcal{L}(P)$.

And here is our definition of validity.

Definition 1.107 (Valid)**.** An argument form $(\{\varphi_1, \dots, \varphi_n\}, \psi)$ is *valid* if and only if each valuation $V : P \to \{0,1\}$ has

$$\hat{V}(\varphi_1) = \cdots = \hat{V}(\varphi_n) = 1 \implies \hat{V}(\psi) = 1.$$

We notate this by $\varphi_1, \dots, \varphi_n \models \psi$ and say "ψ is a *consequence* of $\{\varphi_1, \dots, \varphi_n\}$."

Example 1.108. We have that $(p \vee q), \neg q \models p$.

Intuitively, an argument form is valid if, when the premises are true, the conclusion is also true. This is essentially the definition that we wanted.

Remark 1.109. One can extend this definition to work with infinitely many premises, but we will not do this here.

Observe that checking validity is a finite matter because, to check $\varphi_1, \dots, \varphi_n \models \psi$, we only need to consider the finitely many propositions that take place in any of $\varphi_1, \dots, \varphi_n, \psi$. And then each proposition has only two options, so in total we are safely in a finite universe.

1.8 February 4

Here we go.

1.8.1 Validity for Argument Forms

Last time we left off talking about the validity of an argument form. We had the following definition.

Definition 1.110 (Valid)**.** An argument form $(\{\varphi_1, \dots, \varphi_n\}, \psi)$ is *valid* if and only if each valuation $V : P \to \{0, 1\}$ has

$$\hat{V}(\varphi_1) = \cdots = \hat{V}(\varphi_n) = 1 \implies \hat{V}(\psi) = 1.$$

We notate this by $\varphi_1, \dots, \varphi_n \models \psi$ and say "ψ is a *consequence* of $\{\varphi_1, \dots, \varphi_n\}$."

Remark 1.111. Importantly, a computer could always decide validity in finite time, essentially by making some huge truth table. In other words, determining validity is "decidable."

Let's see an example.

Exercise 1.112. We show that $\{\neg(p \wedge q), p\} \models \neg q$.

Proof. We reason using the equations for the connectives. Namely, suppose $V : \{p, q\} \to \{0, 1\}$ is a valuation such that $\hat{V}(\neg(p \wedge q)) = \hat{V}(p) = 1$. Now, we see

$$1 = \hat{V}(\neg(p \wedge q)) = 1 - \hat{V}((p \wedge q)) = 1 - \min\{\hat{V}(p), \hat{V}(q)\}.$$

Thus, $\min\{\hat{V}(p), \hat{V}(q)\} = 0$, so one of $\hat{V}(p) = 0$ or $\hat{V}(q) = 0$. But $\hat{V}(p) = 1$, so $\hat{V}(q) = 0$ is forced instead. Thus,

$$\hat{V}(\neg q) = 1 - \hat{V}(q) = 1 - 0 = 1,$$

which is what we wanted. ∎

In fact, the above exercise shows the following.

Proposition 1.113. Fix $\alpha, \beta \in \mathcal{L}(P)$. Then $\{\neg(\alpha \wedge \beta), \beta\} \models \beta$.

Proof. Reuse the proof above. ∎

More generally, we have the following notion of substitution.

Proposition 1.114 (Substitution)**.** Suppose that $\{\varphi_1, \dots, \varphi_n\} \models \psi$ in $\mathcal{L}(P)$, then any substitution of the atomic formulae P by formulae in $\mathcal{L}(Q)$ (which induces a map $f : \mathcal{L}(P) \to \mathcal{L}(Q)$), then $\{f\varphi_1, \dots, f\varphi_n\} \models \psi$.

Proof. Induction to make f and then show the statement. ∎

Example 1.115. We know that

$$\{\neg((r \vee s) \wedge (s \to p)), (r \vee s)\} \models \neg(s \to p)$$

is valid by Proposition 1.113.

Here are some more examples.

Exercise 1.116. We show that $\{(p \to q), \neg p\} \not\models \neg q$.

Proof. We set a valuation V by $V(p) = 0$ and $V(q) = 1$. Then

$$\hat{V}((p \to q)) = 1_{\hat{V}(p) \leq \hat{V}(q)} = 1_{0 \leq 1} = 1,$$

and $\hat{V}(\neg p) = 1 - \hat{V}(p) = 1 - 0 = 1$. Thus, our premises are true. But $\hat{V}(\neg q) = 1 - \hat{V}(q) = 1 - 1 = 0$ means the conclusion is true, so we are done. ∎

Remark 1.117. Propositional logic cannot see the validity of all arguments. For example, the following argument cannot be broken down into propositions in the ways that $\mathcal{L}(P)$ provides.

1. Every integer greater than 1 is a product of prime numbers.
2. 999 is an integer greater than 1.
3. Therefore, 999 is a product of prime numbers.

The issue is that we need quantifiers: we need a way to plug in 999 into "every integer," which propositional logic does not provide. We will fix this when we talk about predicate logic later in the course.

Let's run down some valid forms of argument. They can be proven from a similar logic to the above.

- Modus ponens: $\{\varphi \to \psi, \varphi\} \models \psi$.
- Modus tollens: $\{\varphi \to \psi, \neg\psi\} \models \neg\varphi$.
- Contraposition: $\{\varphi \to \psi\} \models \neg\psi \to \neg\varphi$.
- Disjunctive syllogism: $\{\varphi \vee \psi, \neg\varphi\} \models \psi$ and $\{\varphi \vee \psi, \neg\psi\} \models \varphi$.
- Hypothetical syllogism: $\{\varphi \to \psi, \psi \to \chi\} \models \varphi \to \chi$.
- Proof by cases: $\{\varphi \vee \psi, \varphi \to \chi, \psi \to \chi\} \models \chi$. The point is that either antecedent φ or ψ lead to the same conclusion χ.

1.8.2 Validity for Formulae

Here is our definition.

Definition 1.118 (Valid formula, tautology)**.** A formula $\varphi \in \mathcal{L}(P)$ is *valid* or a *tautology* if and only if each valuation $V : P \to \{0, 1\}$ has $\hat{V}(\varphi) = 1$.

Example 1.119. Given any formula $\varphi \in \mathcal{L}(P)$, then any valuation V has $\hat{V}(\varphi) = 1$ or $\hat{V}(\varphi) = 0$, so $\hat{V}(\varphi \vee \neg\varphi) = 1$ in all cases.

Here are some examples. The main point is that valid forms of arguments can turn $\models$ into $\to$ to make a tautology.

- $\varphi \to \varphi$.
- $\varphi \to (\varphi \vee \psi)$ and $\psi \to (\varphi \wedge \psi)$.
- $\varphi \to (\varphi \vee \psi)$ and $\psi \to (\varphi \vee \psi)$.
- "Modus ponens": $((\varphi \to \psi) \wedge \varphi) \to \psi$.
- $((\varphi \to \psi) \wedge \neg\psi) \to \neg\varphi$.
- $((\varphi \to \psi) \wedge (\psi \to \chi)) \to (\varphi \to \chi)$.
- $\varphi \to (\psi \to \varphi)$.
- Peirce's Law: $((\varphi \to \psi) \to \varphi) \to \varphi$.

The last one is not actually a tautology, but it is. The only care we have to worry about is when φ is true, but when φ is true, $(\varphi \to \psi) \to \varphi$ can only be false when $\varphi \to \psi$ can only be false when φ is false.

Remark 1.120. Another way to say that φ is a valid formula is that $\varnothing \models \varphi$. So we will notationally write $\models \varphi$.

Validity of arguments can be turned into validity of formulae, as follows.

Theorem 1.121 (Deduction)**.** An argument form $\{\varphi_1, \ldots, \varphi_n\}$ with conclusion ψ is valid if and only if $(\varphi_1 \wedge \ldots \wedge \varphi_n) \to \psi$ is valid. In other words, $\{\varphi_1, \ldots, \varphi_n\} \models \psi$ if and only if $\psi(\varphi_1 \wedge \cdots \wedge \varphi_n) \to \psi$.

Proof. By definition, $(\varphi_1 \wedge \cdots \wedge \varphi_n) \to \psi$ will be valid if and only if every valuation has V

$$1 = \hat{V}((\varphi_1 \wedge \cdots \wedge \varphi_n) \to \psi),$$

which is equivalent to $\hat{V}(\varphi_1 \wedge \cdots \wedge \varphi_n) \leq \hat{V}(\psi)$ which is equivalent to

$$\min\{\hat{V}(\varphi_1), \ldots, \hat{V}(\varphi_n)\} \leq \hat{V}(\psi).$$

The only case above that we actually care about is that $\hat{V}(\varphi_1) = \cdots = \hat{V}(\varphi_n) = 1$ implies $\hat{V}(\psi) = 1$, which is exactly what $\{\varphi_1, \ldots, \varphi_n\} \models \psi$. ■

1.8.3 Equivalence

We have the following definition.

Definition 1.122 (Equivalence)**.** Two formulae $\varphi, \psi \in \mathcal{L}(P)$ are *equivalent* if and only if

$$\hat{V}(\varphi) = \hat{V}(\psi)$$

for each valuation $V : P \to \{0, 1\}$.

Example 1.123. We have that $\neg\neg p$ is equivalent to p.

Remark 1.124. Saying that φ and ψ are equivalent is the same as asserting $\models \varphi \leftrightarrow \psi$. In fact, we can define φ being equivalent if and only if φ is equivalent to $p \vee \neg p$.

Here are some more examples.

- Idempotence: $\varphi \leftrightarrow (\varphi \wedge \varphi)$.
- Commutativity: $(\varphi \wedge \psi) \leftrightarrow (\psi \wedge \varphi)$ and $(\varphi \vee \psi) \leftrightarrow (\psi \vee \varphi)$.
- Associativity: $(\varphi \wedge \psi) \wedge \chi \leftrightarrow \varphi \wedge (\psi \wedge \chi)$ and $(\varphi \vee \psi) \vee \chi \leftrightarrow \varphi \vee (\psi \vee \chi)$.
- Absorption: $\varphi \leftrightarrow ((\varphi \wedge (\varphi \vee \psi)))$, which we can see by a direct computation.
- Distributivity: $(\varphi \wedge (\psi \vee \chi)) \leftrightarrow ((\varphi \wedge \psi) \vee (\varphi \wedge \chi))$.
- Distributivity: $(\varphi \vee (\psi \wedge \chi)) \leftrightarrow ((\varphi \vee \psi) \wedge (\varphi \vee \chi))$.
- De Morgan Laws: $\neg(\psi \wedge \varphi) \leftrightarrow (\neg\psi \vee \neg\varphi)$.
- De Morgan Laws: $\neg(\psi \vee \varphi) \leftrightarrow (\neg\psi \wedge \neg\varphi)$.

1.9 February 7

We are still talking about semantics. Today we are focusing on satisfiability.

1.9.1 Satisfiability

Satisfiability is dual to validity: if there is any way for the formula to be true, then the formula is satisfiable.

Non-Example 1.125. The formula $p \wedge \neg p$ is unsatisfiable: it is always false.

To help build some intuition, we note that we have some feeling for when multiple propositions may be true or false.

Exercise 1.126. Suppose we have the following statements of agencies responsible for an action.

1. Agency A or Agency B is responsible.
2. It's not the case that both Agencies A and B are responsible.
3. If Agency A is responsible, then the money is coming from fun X.
4. If Agency A is not responsible, then the money is coming from fun Y.
5. If Agency B is responsible, then the money is not coming from fun Y.
6. If Agency B is responsible, then the money is not coming from fun X.

We show not all of these can be true simultaneously; namely, this set of premises is "inconsistent."

Proof. We can translate as follows into our formal language. Here a (resp., b) is "Agency A (resp., B) is responsible" and x (resp., y) is "the money is coming from X (resp., Y)."

1. $a \vee b$.
2. $\neg(a \wedge b)$.
3. $a \to x$.
4. $\neg a \to y$.
5. $b \to \neg y$.
6. $\neg b \to \neg x$.

There is no valuation will make all of these true. If a were true, then x; but also a means $\neg b$, which means $\neg x$. Conversely, if a were not true, then y; but also $\neg a$ means b, which means $\neg y$. So all cases don't make sense. ■

Here are our formal definitions.

Definition 1.127 (Inconsistent)**.** A set of formulae is *inconsistent* if and only if one can prove contradiction from the formulae, using some proof system.

Definition 1.128 (Satisfiable)**.** A set of formulae $\{\varphi_1, \ldots, \varphi_n\} \subseteq \mathcal{L}(P)$ is *satisfiable* if and only if there is a valuation $V : P \to \{0, 1\}$ which satisfies all the formulae. A set of formulae is *unsatisfiable* if and only if it is not satisfiable.

We say that a single formula φ is *(un)satisfiable* if and only if $\{\varphi\}$ is (un)satisfiable. In other words, there is a valuation V such that $V \models \varphi$.

Example 1.129. The formula $p \wedge \neg q$ is satisfiable: $p = 1$ and $q = 0$.

Example 1.130. The formula $p \wedge \neg p$ is unsatisfiable: $p = 0$ and $p = 1$ both give $p \wedge \neg p$ false.

We could also say that $\{\varphi_1, \dots, \varphi_n\}$ is satisfiable if and only if $\varphi_1 \wedge \cdots \wedge \varphi_n$ is satisfiable.

Remark 1.131. In fact, it will be true that any unsatisfiable set of formulae will be able to derive contradiction from our proof system. So unsatisfiability will be equivalent to inconsistent, though this is not immediately obvious.

Satisfiability in fact is more directly dual to validity.

Proposition 1.132. Fix $\varphi \in \mathcal{L}(P)$. Then φ is satisfiable if and only if $\neg\varphi$ is not valid.

Proof. We have φ is satisfiable if and only if there exists some $V : P \to \{0, 1\}$ such that $V \models \varphi$.

Conversely, $\neg\varphi$ is not valid if and only if there exists a valuation $V : P \to \{0, 1\}$ such that $\hat{V}(\neg\varphi) = 0$ if and only if $\hat{V}(\varphi) = 1$. ■

Example 1.133. The expression $p \wedge \neg p$ is unsatisfiable, so $\neg(p \wedge \neg p)$ is valid.

Corollary 1.134. Fix $\varphi \in \mathcal{L}(P)$. Then φ is valid if and only if $\neg\varphi$ is not satisfiable.

Proof. This is the contraposition of Proposition 1.132. ■

To review, we have defined the following semantic notions.

- Valid argument forms.
- Valid formulae.
- Satisfiable sets of formulae.
- Satisfiable formulae.

In fact, we know that studying any one of these can study any other, assuming finiteness.

1.9.2 Infinite Arguments

We quickly remark that we can generalize some of our notions to the infinite case.

Definition 1.135 (Satisfiable)**.** Any set $S \subseteq \mathcal{L}(P)$ is *satisfiable* if and only if there is a valuation $V : P \to \{0, 1\}$ satisfying all of them.

However, this is no longer the same as some formula

$$\bigwedge_{\varphi \in S} \varphi$$

where we and everything together: all of our formulae have finite length. One could generalize formulae to allow infinite formulae, but we won't do so here.

We remark while we are here that we have the following notion of compactness.

Theorem 1.136 (Compactness)**.** A set S of formulae is satisfiable if and only if every finite subset of S is satisfiable.

Proof. The forwards direction is clear: if a valuation satisfies S, then the valuation will satisfy any subset.

The backwards direction is significantly harder. In essence, one shows that a set of formula is unsatisfiable if and only if one is able to derive contradiction. However, deriving contradiction is a finite process which only takes finitely many propositions, so this finite subset would also be unsatisfiable. ■

Example 1.137. Any finite graph can be 4-colored, so by compactness, any graph (possibly infinite) can be 4-colored.

Next class we will start talking about economy of language: exactly what connectives do we need to construct all possible truth functions? It will turn out that we do not need many.

Theme 2
Basic Theory

2.1 February 9

Welcome back everybody.

2.1.1 Defining Translation

As a fun exercise, for the next few lectures we will be reducing the number of connectives we really have to talk about. Recall that two formulae $\varphi, \psi \in \mathcal{L}(P)$ are equivalent if and only if each valuation $V : P \to \{0, 1\}$ has $\hat{V}(\varphi) = \hat{V}(\psi)$.

Our main goal for today is to show that any formula in our language is equivalent to a formula whose only connectives are $\neg$ and $\wedge$.

> **Remark 2.1.** One reason we might care is that this reduces the number of logic gates that we would need. Another reason we might care is that, for proofs on truth values, it reduces our headaches in checking if something is true for any connective if we simply get rid of all connectives.

The main idea in the proof is to define a translation function $T : \mathcal{L}(P) \to \mathcal{L}(P)$, where the output should only use the connectives $\neg, \wedge, \vee$ and be equivalent to the original input formula. As usual, we define T recursively, as follows; fix $\varphi, \psi \in \mathcal{L}(P)$.

- We define $T(p) := p$ for any atomic formula $p \in P$.
- We define $T(\neg\varphi) := \neg T(\varphi)$. Note that this does not mean we output $\neg\varphi$: we still need to translate φ, but the negation in front of φ is safe.
- We define $T((\varphi \wedge \psi)) := (T(\varphi) \wedge T(\psi))$. Again, the point is that we don't care about the connective, but we still need to translate φ and ψ.

The next three clauses are more difficult.

- We define $T((\varphi \vee \psi)) := \neg(\neg T(\varphi) \wedge \neg T(\psi))$. The equivalence is De Morgan's laws: saying that "we are going to the beach or hiking" is the same as "it is not the case that we are neither going to the beach nor going hiking."
- We define $T((\varphi \to \psi)) := \neg(T(\varphi) \wedge \neg T(\psi))$. The equivalence is by noting $(\varphi \to \psi)$ simply means $(\neg\varphi \vee \psi)$, so we get this by applying De Morgan's laws.
- We define $T((\varphi \leftrightarrow \psi)) = \big(\neg(T(\varphi) \wedge \neg T(\psi)) \wedge \neg(T(\psi) \wedge \neg T(\varphi))\big)$. The idea here is that $(\varphi \leftrightarrow \psi)$ means $(\varphi \to \psi) \wedge (\psi \to \varphi)$.

Exercise 2.2. We compute $T(\neg(p \to (q \vee r)))$.

Proof. We mindlessly apply our translation rules. This gives

$$\begin{aligned} T(\neg(p \to (q \vee r))) &= \neg T((p \to (q \vee r))) \\ &= \neg\neg(T(p) \wedge \neg T(q \vee r)) \\ &= \neg\neg(T(p) \wedge \neg\neg(\neg T(q) \wedge \neg T(r))) \\ &= \boxed{\neg\neg(p \wedge \neg\neg(\neg q \wedge \neg r))}, \end{aligned}$$

which finishes. This formula might look worse (it's longer), but it only involves $\neg$ and $\wedge$. We remark that we can get rid of $\neg\neg$ to get an equivalent formula, which gives $p \wedge \neg q \wedge \neg r$. ■

2.1.2 Rigorizing Translation

Now let's actually prove that we translated correctly.

Theorem 2.3. Any formula $\varphi \in \mathcal{L}(P)$ is equivalent to a formula whose only connectives are $\neg$ and $\wedge$.

Proof. The main idea is to use our translation T defined above. We have two different main claims.

Lemma 2.4. Any formula $\varphi \in \mathcal{L}(P)$ only uses $\neg$ and $\wedge$ as its connectives.

Proof. This is an induction which we can see directly. Let $S \subseteq \mathcal{L}(P)$ be the set of formula such that $\varphi \in S$ if and only if $T(\varphi)$ only uses the connectives $\neg$ and $\wedge$. We have the following checks.

- We see that each $p \in P$ has $T(p) = p$ has no connectives at all.
- For the inductive step, suppose $\varphi, \psi \in S$. Then we have the following checks.
  - We see $T(\neg\varphi) = \neg T(\varphi)$ will only use $\neg$ and $\wedge$ because $T(\varphi)$ will only use $\neg$ and φ.
  - We see $T((\varphi \wedge \psi)) := (T(\varphi) \wedge T(\psi))$ will only use $\neg$ and $\wedge$ because $T(\varphi)$ and $T(\psi)$ will only use $\neg$ and φ.
  - We see $T((\varphi \vee \psi)) := \neg(\neg T(\varphi) \wedge \neg T(\psi))$ will only use $\neg$ and $\wedge$ because $T(\varphi)$ and $T(\psi)$ will only use $\neg$ and φ.
  - We see $T((\varphi \to \psi)) := \neg(T(\varphi) \wedge \neg T(\psi))$ will only use $\neg$ and $\wedge$ because $T(\varphi)$ and $T(\psi)$ will only use $\neg$ and φ.
  - We see $T((\varphi \leftrightarrow \psi)) = \big(\neg(T(\varphi) \wedge \neg T(\psi)) \wedge \neg(T(\psi) \wedge \neg T(\varphi))\big)$ will only use $\neg$ and $\wedge$ because $T(\varphi)$ and $T(\psi)$ will only use $\neg$ and φ.

From all these checks, we see that S must contain $\mathcal{L}(P)$, so $S = \mathcal{L}(P)$, so we are done. ■

Lemma 2.5. Any formula $\varphi \in \mathcal{L}(P)$ is equivalent to $T(\varphi)$.

Proof. We proceed by induction. For our base case, we are saying that $p \in P$ is equivalent to $T(p) = p$, which is clear.

For the inductive step, fix φ and ψ which are equivalent to their translation. We have the following checks.

- We show $(\varphi \vee \psi)$ is equivalent to $T((\varphi \vee \psi))$. Well, fix any valuation V, and we see that

$$\begin{aligned}\hat{V}((\varphi \vee \psi)) &= \max\{\hat{V}(\varphi), \hat{V}(\psi)\} \\ &\stackrel{*}{=} 1 - \min\{1 - \hat{V}(\varphi), 1 - \hat{V}(\psi)\}.\end{aligned}$$

  The point is that we can check $\stackrel{*}{=}$ is true by a computation. Here is the table.

x	y	$\max\{x, y\}$	$1-x$	$1-y$	$1 - \min\{1-x, 1-y\}$
1	1	1	0	0	1
1	0	1	0	1	1
0	1	1	1	0	1
0	0	0	1	1	0

  We continue. We see, because φ is equivalent to $\hat{V}(\varphi)$ and similar for ψ, we see

$$\begin{aligned}\hat{V}((\varphi \vee \psi)) &= 1 - \min\{1 - \hat{V}(\varphi), 1 - \hat{V}(\psi)\} \\ &= 1 - \min\{1 - \hat{V}(T(\varphi)), 1 - \hat{V}(T(\psi))\} \\ &= 1 - \min\{\hat{V}(\neg T(\varphi)), \hat{V}(\neg T(\psi))\} \\ &= 1 - \hat{V}((\neg T(\varphi) \wedge \neg T(\psi))) \\ &= \hat{V}(\neg(\neg T(\varphi) \wedge \neg T(\psi))).\end{aligned}$$

  Thus, $(\varphi \vee \psi)$ is equivalent to $\neg(\neg T(\varphi) \wedge \neg T(\psi)) = T((\varphi \vee \psi))$, so we are done.

- The cases of $\to$ and $\leftrightarrow$ are similar, so we will omit them.

- We show $(\varphi \wedge \psi)$ is equivalent to $T((\varphi \wedge \psi)) = (T(\varphi) \wedge T(\psi))$. But we can compute, for any valuation V,

$$\begin{aligned}\hat{V}((\varphi \wedge \psi)) &= \min\{\hat{V}(\varphi), \hat{V}(\psi)\} \\ &= \min\{\hat{V}(T(\varphi)), \hat{V}(T(\psi))\} \\ &= \hat{V}((T\varphi) \wedge T(\psi)),\end{aligned}$$

  which finishes.

- We show $\neg\varphi$ is equivalent to $T(\neg\varphi) = \neg T(\varphi)$. But we can compute, for any valuation V,

$$\begin{aligned}\hat{V}(\neg\varphi) &= 1 - \hat{V}(\varphi) \\ &= 1 - \hat{V}(T(\varphi)) \\ &= \hat{V}(\neg T(\varphi)),\end{aligned}$$

  which finishes.

The above checks complete our induction. ■

The above two claims show that a formula $\varphi \in \mathcal{L}(P)$ is equivalent to some formula $T(\varphi)$, where $T(\varphi)$ only uses the connectives $\neg$ and $\wedge$. ■

> **Remark 2.6.** We might want to optimize the check for, say, $(\varphi \vee \psi)$ being equivalent to $T((\varphi \vee \psi))$ by avoiding the table. However, this cannot really be done because we must deal with what $\neg(\neg T(\varphi) \wedge \neg T(\psi))$ means sometime in the proof, which is where the $\stackrel{*}{=}$ equality comes from.

Remark 2.7. The above checks were pretty annoying. But for any proof about truth in the future will now only have to check $\neg$ and $\wedge$ in an inductive step.

Explicitly, if we are trying to prove some property about formulae which is preserved by equivalence (namely, one that really only cares about formulae as truth functions), then our inductive step only has to deal with formulae which use $\wedge$ and $\neg$.

Non-Example 2.8. The property that a formula only contains one $\neg$ connector is not preserved by equivalence. For example, p is equivalent to $\neg\neg p$.

2.1.3 More Economy

There are other ways we can be economical about our connectives. For example, we could only use $\neg$ and $\vee$ using a translation function $S : \mathcal{L}(P) \to \mathcal{L}(P)$ as follows.

- We define $S(p) := p$ for any $p \in P$.
- We define $S(\neg\varphi) := \neg S(\varphi)$.
- We define $S((\varphi \wedge \psi)) := \neg(\neg S(\varphi) \vee \neg S(\psi))$.
- We define $S((\varphi \wedge \psi)) := \neg(\neg S(\varphi) \vee \neg S(\psi))$.
- We define $S((\varphi \to \psi)) := (\neg S(\varphi) \vee S(\psi))$.
- We define $S((\varphi \leftrightarrow \psi)) := (\neg(\neg S(\varphi) \vee S(\psi)) \vee \neg(\neg S(\psi) \vee S(\varphi)))$. Again, this comes from noting $(\varphi \leftrightarrow \psi)$ is the same as $((\varphi \to \psi) \wedge (\psi \to \varphi))$.

We will omit the checks that this is a valid translation.

2.2 February 11

Here we go.

2.2.1 A Little More Economy

Let's continue with our translation. We will again note that every formula in a language is equivalent to one in which the only connectives are $\{\neg, \to\}$. Here is our translation.

- We define $U(p) := p$ for each atomic formula $p \in P$.
- We define $U(\neg\varphi) := \neg U(\varphi)$.
- We define $U(\varphi \wedge \psi) := \neg(U(\varphi) \to \neg U(\psi))$.
- We define $U(\varphi \vee \psi) :=$
- We define $U(\varphi \to \psi) := (U(\varphi) \to U(\psi))$.
- We define $U(\varphi \leftrightarrow \psi) := (U(\varphi) \to U(\psi)) \wedge (U(\psi) \to U(\varphi))$.

2.2.2 Too Much Economy

We quickly remark that we cannot be too callous. For example, we claim that the connectives $\{\wedge, \vee, \to, \leftrightarrow\}$ are not enough. Here is the main claim.

Lemma 2.9. Any formula $\varphi \in L(\{p_1, \dots, p_n\})$ where φ has only the connectives $\{\wedge, \vee, \to, \leftrightarrow\}$ will have $p_1, \dots, p_n \models \varphi$.

This will be enough because it means any such formula φ cannot be equivalent to $\neg p_1$ because $\{p_1, \dots, p_n\} \not\models \varphi$.

Proof. We induct. For our base case, we note that any of the propositions p_i will have $\{p_1, \dots, p_n\} \models p_i$ for free. Now for our inductive hypothesis, suppose $\{p_1, \dots, p_n\} \models \varphi, \psi$. Then for any $\# \in \{\wedge, \vee, \to, \leftrightarrow\}$ will have

$$\hat{V}(\varphi) = \hat{V}(\psi) = 1 \implies \hat{V}((\varphi \# \psi)) = 1$$

by hand. So $\hat{V}(p_i) = 1$ for each p_i implies $\hat{V}(\varphi) = \hat{V}(\psi) = 1$ (by hypothesis), so $\hat{V}((\varphi \# \psi)) = 1$, finishing. ∎

Remark 2.10. Intuitively, when we set everything to be true, all the given connectives will send true to true.

We close with a question.

Question 2.11. Is any formula equivalent to one involving the symbols $\{\neg, \leftrightarrow\}$?

2.2.3 The Most Economy

We start by defining a new connective.

Definition 2.12 (Sheffer stroke, NAND)**.** Given two formulae φ, ψ, we define $(\varphi \uparrow \psi)$ by the following truth table.

$\hat{V}(\varphi)$	$\hat{V}(\psi)$	$\hat{V}((\varphi \uparrow \psi))$
1	1	0
1	0	1
0	1	1
0	0	1

Intuitively, $\hat{V}((\varphi \uparrow \psi))$ is true if and only if at least one of φ or ψ is false. In symbols, $\hat{V}((\varphi \uparrow \psi)) = 1 - \hat{V}(\varphi)\hat{V}(\psi)$.

We note that $\neg\varphi$ is equivalent to $(\varphi \uparrow \varphi)$ because

$$\hat{V}((\varphi \uparrow \varphi)) = 1 - \hat{V}(\varphi)^2 = 1 - \hat{V}(\varphi) = \hat{V}(\neg\varphi),$$

where $\hat{V}(\varphi)^2 = \hat{V}(\varphi)$.

Further, $(\varphi \vee \psi)$ is equivalent to $\neg(\neg\varphi \wedge \neg\psi)$ is equivalent to $(\neg\varphi \uparrow \neg\psi)$ is equivalent to

$$((\varphi \uparrow \varphi) \uparrow (\psi \uparrow \psi)).$$

Thus, we note that any formula is equivalent to one involving only the connectives $\vee$ or $\neg$, which we can then translate to a formula only involving the connectives.

Of course, there are other translations.

Example 2.13. We note $(\varphi \wedge \psi)$ is equivalent to $\neg\neg(\varphi \wedge \psi)$ is equivalent to $\neg(\varphi \uparrow \psi)$ is equivalent to $(\varphi \uparrow \psi) \uparrow (\varphi \uparrow \psi)$.

Remark 2.14. Any formula is also equivalent to one which only uses the connective "$\downarrow$," which has the following truth table.

$\hat{V}(\varphi)$	$\hat{V}(\psi)$	$\hat{V}((\varphi \downarrow \psi))$
1	1	0
1	0	0
0	1	0
0	0	1

To see the above remark, we note that the top row must be false (or else we preserve truth), and the bottom row must be true (or else we preserve false). But the truth table

$\hat{V}(\varphi)$	$\hat{V}(\psi)$	$\hat{V}((\varphi \# \psi))$
1	1	0
1	0	1
0	1	0
0	0	1

does not work because it does not depend on $\hat{V}(\varphi)$. Similarly,

$\hat{V}(\varphi)$	$\hat{V}(\psi)$	$\hat{V}((\varphi \# \psi))$
1	1	0
1	0	0
0	1	1
0	0	1

does not work because it does not depend on $\hat{V}(\psi)$.

2.2.4 Truth Functions

We should probably go back and show that any function on truth can be achieved by our language.

Definition 2.15. Fix a positive integer n. Then an *n-ary truth function* is a function $f : \{0,1\}^n \to \{0,1\}$.

Example 2.16. The function $f(x) = 1 - x$ corresponds to $\neg$. The function $f(x,y) = \min\{x,y\}$ corresponds to $\wedge$.

Example 2.17. The function

$$f(x,y,z,w) = \min\{x,y,z,1-w,xy,\max\{x,y\}\}$$

is a truth function.

We note that, because there are only finitely many inputs in $\{0,1\}^n$, we can simply tabulate to give the function. For example, here is a truth function.

p	q	r	$f(p,q,r)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

This might look like a mess, but it is perfectly reasonable.

Remark 2.18. Note that, among the 2^n inputs for an n-ary truth function, we have two choices for inputs, so there are 2^{2^n} total possible truth functions. That's a lot.

So far we have been noting that formulae in our language correspond to truth functions.

Example 2.19. The formula $\neg(p \leftrightarrow q)$ corresponds to the truth function described by the following table.

p	q	$f(p,q)$
1	1	0
1	0	1
0	1	1
0	0	0

This is not unique: we could also do $(p \leftrightarrow \neg q)$.

Let's make this rigorous.

Definition 2.20 (Truth functions from formulae)**.** Fix $\varphi \in \mathcal{L}(\{p_1, \ldots, p_n\})$. We then say φ *defines the n-ary truth function* f^n_φ defined as follows: any $(x_1, \ldots, x_n) \in \{0,1\}^n$ defines a valuation $V(p_k) := x_k$, from which we define

$$f^n_\varphi(x_1, \ldots, x_n) := \hat{V}(\varphi).$$

Intuitively, the truth tables for f^n_φ and $\hat{V}(\varphi)$ "match up" in the way that the row for some $(x_1, \ldots, x_n) \in \{0,1\}^n$ corresponds to the row $V(p_1) = x_1, \ldots, V(p_n) = 1$.

Let's see some examples.

Example 2.21. The following truth tables match up.

p	$\neg p$
1	0
0	1

x	$f(x)$
1	0
0	1

So $\neg p$ defines the function f.

Example 2.22. We find a formula to define the following truth function.

p	q	r	$f(p,q,r)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

We claim that $((p \to q) \wedge (\neg p \to r))$ will do the trick. Checking this is a matter of writing out the truth table, but we won't do the work here.

2.2.5 Defining Truth Functions

There is actually an algorithm which will always be able to produce a truth table. Let's work with the following example.

p	q	r	$f(p,q,r)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

(In the following discussion, we will drop parentheses when the meaning is clear or the parentheses do not matter.)

Above we have highlighted all the cases which are true. To create a formula for this, we simply hard-code in all possible cases where are true. To be explicit, the top highlighted row is $(p \wedge q \wedge r)$, the next row is $(p \wedge q \wedge \neg r)$, and so on. Then we just need to ensure that we live in exactly one of these cases, which looks like

$$(p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge r).$$

This is long and horrendous, but it is correct.

Remark 2.23. To feel how bad this algorithm is, consider how annoying it would be to try to create a truth table for "$f(x_1, \ldots, x_n)$ returns true if and only if at least $\lfloor n/2 \rfloor$ of the x_k are true."

Example 2.24. The truth table

p	q	$f(p,q)$
1	1	0
1	0	0
0	1	0
0	0	1

can be defined by the formula $(\neg p \wedge \neg q)$.

We quickly remark that there will be lots of different formulae which can give the same truth function.

Lemma 2.25. Two formula $\varphi, \psi \in \mathcal{L}(\{p_1, \ldots, p_n\})$ have $f_\varphi^n = f_\psi^n$ if and only if φ and ψ are logically equivalent.

Proof. To say that φ and ψ means that any valuation $V : \{0,1\}^n \to \{0,1\}$ gives $\hat{V}(\varphi) = \hat{V}(\psi)$. Translating this into discussion about truth functions, a valuation corresponds to an input $(x_1, \ldots, x_n) \in \{0,1\}^n$, so being logically equivalent is saying that f_φ^n and f_ψ^n are equal on all inputs $(x_1, \ldots, x_n) \in \{0,1\}^n$. ■

2.3 February 14

We're back in action everybody.

2.3.1 Equivalence Classes

Recall that it is possible for two formulae to define the same truth function. For example, consider the truth table as follows.

x_1	x_2	$\max\{x_1, x_2\}$
1	1	1
1	0	1
0	1	1
0	0	0

This function is equal to either $f^2_{p\vee q}$ or $f^2_{\neg(\neg p\wedge\neg q)}$. Indeed, we can compute as follows.

p	q	$p\vee q$	$\neg p$	$\neg q$	$\neg(\neg p\neg q)$
1	1	1	0	0	1
1	0	1	0	1	1
0	1	1	1	0	1
0	0	0	1	1	0

We also recall that we had the following lemma from last class.

Lemma 2.25. Two formula $\varphi, \psi \in \mathcal{L}(\{p_1, \dots, p_n\})$ have $f^n_\varphi = f^n_\psi$ if and only if φ and ψ are logically equivalent.

The above idea gives us a notion of an equivalence class of formulae.

Definition 2.26 (Eqiuvalence class)**.** Fix a formula $\varphi \in \mathcal{L}(P)$. Then we define the *equivalence class* of φ in $\mathcal{L}(P)$ to be the set of formulae which are logically equivalent to φ. We denote this set by $[\varphi]$, for which φ is a representative.

Example 2.27. We have $\neg(\neg p \wedge \neg q) \in [p \vee q]$ by de Morgan's laws. Similarly, $\neg\neg(p \vee q) \in [p \vee q]$.

The point of the equivalence class is that it ignores the underlying syntax of a formula and only cares about its semantics. If the only thing that we care about is what a formula "means" instead of what it looks like, then it makes sense to lump together formulae which are equivalent.

Our notion of equivalence class makes sense, as follows.

Proposition 2.28. Logical equivalence is reflexive, symmetric, and transitive. That is, for any $\alpha, \beta, \gamma \in \mathcal{L}(P)$, we have the following.

- Reflexive: $\alpha \equiv \alpha$.
- Symmetric: $\alpha \equiv \beta$ implies $\beta \equiv \alpha$.
- Transitive: $\alpha \equiv \beta$ and $\beta \equiv \gamma$ implies $\alpha \equiv \gamma$.

Proof. We show the claims as follows. Fix $\alpha, \beta, \gamma \in \mathcal{L}(P)$. Denote equivalence by $\equiv$.

- Reflexive: for any valuation $V : P \to \{0, 1\}$, we have $\hat{V}(\alpha) = \hat{V}(\alpha)$, so $\alpha \equiv \alpha$.
- Symmetric: if $\alpha \equiv \beta$, then any valuation $V : P \to \{0, 1\}$ has $\hat{V}(\alpha) = \hat{V}(\beta)$ so that $\hat{V}(\beta) = \hat{V}(\alpha)$, so $\beta \equiv \alpha$.
- Transitive: if $\alpha \equiv \beta$ and $\beta \equiv \gamma$, then any valuation $V : P \to \{0, 1\}$ has $\hat{V}(\alpha) = \hat{V}(\beta)$ and $\hat{V}(\beta) = \hat{V}(\gamma)$ so that $\hat{V}(\alpha) = \hat{V}(\gamma)$, so $\alpha \equiv \gamma$. ∎

Remark 2.29. The above properties show that logical equivalence is an equivalence relation.

Corollary 2.30. Two formulae $\varphi, \psi \in \mathcal{L}(P)$ have φ equivalent to ψ if and only if $[\varphi] = [\psi]$. So this is also equivalent to $f^n_\varphi = f^n_\psi$.

Proof. We show the directions independently.

- Suppose $\varphi \equiv \psi$. Then, for any $\alpha \in [\varphi]$, we have $\varphi \equiv \alpha$, so $\alpha \equiv \varphi$ and $\varphi \equiv \psi$, so $\alpha \equiv \psi$, so $\alpha \in [\psi]$. By symmetry, any $\alpha \in [\psi]$, we have $\psi \equiv \alpha$ and $\psi \equiv \varphi$, so $\alpha \equiv \varphi$, so $\alpha \in [\varphi]$.
- Suppose $[\varphi] = [\psi]$. Then $\varphi \equiv \varphi$ implies $\varphi \in [\varphi] = [\psi]$, so $\varphi \equiv \psi$. ■

2.3.2 Local Finiteness

We might be interested in counting the total number of equivalence classes, but this might be infinite if the number of propositions is infinite. Namely, each proposition $p \in P$ gives a different equivalence class $[p]$, lower-bounding the total number of formulae.

However, there is a notion of local finiteness.

Proposition 2.31 (Local finiteness)**.** Fix $P = \{p_1, \dots, p_n\}$ a finite set. Then there are only finitely many equivalence classes in $\mathcal{L}(P)$. In other words, there are only finitely many non-equivalent formulae we can write with the propositions $\{p_1, \dots, p_n\}$.

Example 2.32. There are only four different possible truth functions in $\mathcal{L}(\{p\})$, as follows.

p	$p \vee \neg p$
1	1
0	1

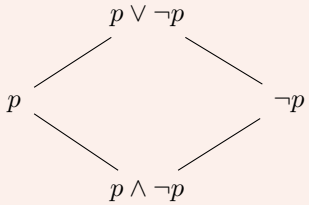
p	p
1	1
0	0

p	$\neg p$
1	0
0	1

p	$p \wedge \neg p$
1	0
0	0

Explicitly, each input $\{0, 1\}$ for $V(p)$ has only two options, giving 2^2 total possible truth tables, which we have manifest above.

Remark 2.33. We can tabulate equivalence classes in $\mathcal{L}(\{p\})$ in a Hasse diagram (moving up means logically implies) as follows.



For example, $p \wedge \neg p \models p \models p \vee \neg p$. As an aside, this is a boolean algebra.

It is a good exercise to make the Hasse diagram for $\mathcal{L}(\{p, q\})$.

Now let's prove our result.

Proof of Proposition 2.31. We can inject equivalence classes $[\varphi]$ in $\mathcal{L}(P)$ to their associated truth function f^n_φ. Namely, the function

$$[\varphi] \mapsto f^n_\varphi$$

is well-defined and one-to-one/injective. We have the following checks.

- Well-defined: if $[\varphi] = [\psi]$, then $\varphi \equiv \psi$, so $f^n_\varphi = f^n_\psi$.
- Injective: $f^n_\varphi = f^n_\psi$ implies $\varphi \equiv \psi$ implies $[\varphi] = [\psi]$.

This embedding implies that the number of equivalence classes is bounded above by the number of truth functions because each equivalence class yields a unique truth function.

Remark 2.34. The last step we just did is the Pigeonhole principle: if we have an injection $A \hookrightarrow B$, then the number of elements of A is at most the number of elements of B. This should feel intuitively obvious to prove, and in fact it makes a pretty good way to define the size of a set to begin with, so there is nothing to prove.

So to finish, we note that there are 2^{2^n} truth functions on n variables: there are only 2^n different ways to set the inputs to set true and false for each $p_\bullet$ because each $p_\bullet$ has two options. But then each of these inputs has 2 options to return true or false, so we have a total of 2^{2^n} different functions. ■

Corollary 2.35. Let $P = \{p_1, \dots, p_n\}$ be a finite set. There are at most 2^{2^n} different equivalence classes of formulae.

Proof. This essentially follows from the proof of Proposition 2.31, where we upper-bounded the number of equivalence classes by the number of truth functions, of which there are at most 2^{2^n}. ■

Remark 2.36. It will turn out that all truth functions are achievable from formulae, so the map $[\varphi] \mapsto f^n_\varphi$ will also be onto/surjective, so the number of logical equivalence classes will be exactly the number of truth functions, which is 2^{2^n}.

2.3.3 Completeness

Let's manifest Remark 2.36. Given a truth function, we need to find its formula.

Example 2.37. The truth table

x_1	x_2	$f(x_1, x_2)$
1	1	0
1	0	1
0	1	1
0	0	0

is $f^2_{\neg p \leftrightarrow q}$.

So here is the main claim.

Theorem 2.38. Any truth function (with finitely many inputs) can be written by a formula in our language.

Proof. Fix $f : \{0,1\}^n \to \{0,1\}$ some truth function with n inputs, and let $P = \{p_1, \dots, p_n\}$ be our propositions. We remark that if f is the zero function, then $f = f^n_{p_1 \wedge \neg p_1}$ will work.

Otherwise, f returns true somewhere. Let S be the set of all inputs which return true, and because f takes finitely many inputs, the input space of f is finite, so S is finite, so set $S = \{s_1, \dots, s_n\}$. Now, for each $s := (x_1, \dots, x_n) \in S$, we create the term

$$\varphi_s := \pm p_1 \wedge \cdots \wedge \pm p_n,$$

where we choose $+p_\bullet = p_\bullet$ when $x_\bullet = 1$ and $-p_\bullet = \neg p_\bullet$ when $x_\bullet = 0$. Then the formula

$$\varphi = \varphi_{s_1} \vee \cdots \varphi_{s_n}$$

will do the trick. Namely, it is true if and only if our input s lives in S because φ_s will be true for the input x if and only if $s = x$. ■

Remark 2.39. Because we showed that any formula is equivalent to one that only uses the connectives $\{\neg, \wedge\}$ or even $\{\uparrow\}$, any truth function can be written by a formula in $\{\neg, \wedge\}$ or even $\{\uparrow\}$.

2.4 February 16

Here we go.

2.4.1 Finishing Completeness

Today we are continuing the proof of Theorem 2.38.

Theorem 2.38. Any truth function (with finitely many inputs) can be written by a formula in our language.

Proof. Here is the idea: suppose that $f : \{0,1\}^n \to \{0,1\}$ is a truth function. Because it has finitely many inputs, the pre-image of $\{1\}$ is finite, so we enumerate them by

$$r_1, \ldots, r_m,$$

where the "row" r_k is the input $(b_{k1}, \ldots, b_{km})$. With this, we define

$$\varphi_{r_k} := \pm p_1 \wedge \cdots \wedge p_m,$$

where we take $+p_\bullet = p_\bullet$ if and only if $b_{k\bullet} = 1$ and $-p_\bullet = \neg p_\bullet$ otherwise.

Definition 2.40 (Literal)**.** A formula of the form $p, \neg p \in \mathcal{L}(P)$ is called a *literal*.

The point is that φ_{r_k} is true if and only if the input is equal to $(b_1, \ldots, b_m)$. In particular, when we take

$$\varphi := \varphi_{r_1} \vee \cdots \vee \varphi_{r_m},$$

we see that φ is true if and only if one of the $\varphi_{r_\bullet}$ is true if and only if one of the inputs is equal to $r_\bullet$. So φ defines the truth function f. ■

2.4.2 Normal Forms

We have the following corollary of Theorem 2.38.

Corollary 2.41 (Disjunctive normal form)**.** Fix P a finite set. Every truth function $P \to \{0,1\}$ can be represented by using $\neg, \vee, \wedge$ as a disjunction of conjunction of literals.

Recall that disjunction means $\vee$ (having one but not the other is legal), and conjunction means $\wedge$ (we must have both).

Proof. This is exactly what Theorem 2.38 does: it produces conjunctions of literals and then disjuncts them together. ■

Definition 2.42 (Disjunctive normal form)**.** A formula φ which is a disjunction of a conjunction of literals is said to be in *disjunctive normal form.*

Example 2.43. Disjunctive normal form is not unique. For example,

$$(p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge r)$$

is equivalent to

$$(\neg p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge r),$$

both of which are in disjunctive normal form.

There is also a dual notion to disjunctive normal form.

Definition 2.44 (Conjunctive normal form)**.** A formula φ which is a conjunction of a disjunction of literals is said to be in *disjunctive normal form.*

To put a formula into conjunctive normal form, we can put $\neg\varphi$ into disjunctive normal form as

$$\neg\varphi \equiv \bigvee_{k=1}^{m} (\ell_{k,1} \wedge \cdots \wedge \ell_{k,n}).$$

In other words, we look for all the 0s in the truth table. Then we negate to take

$$\varphi \equiv \bigwedge_{k=1}^{m} (\ell'_{k,1} \vee \cdots \vee \ell'_{k,n}),$$

which is our conjunctive normal form, where $\ell'_{k,\bullet}$ is the negated literal.

Example 2.45. Consider the truth table as follows.

p	q	r	f
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

Our zeroes are at

$$\neg\varphi \equiv (p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge \neg r).$$

Negating and applying de Morgan's laws, our formula is

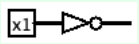
$$\varphi \equiv (\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (p \vee \neg q \vee r) \wedge (p \vee q \vee r).$$

We close by noting our proof of Theorem 2.38 finishes the exactness in the proof of Corollary 2.35. To be more explicit, we showed that there are at most 2^{2^n} different equivalent formulae because this is the number of truth functions, but Theorem 2.38 showed that all truth functions are possible. So there are indeed 2^{2^n} total different equivalence classes of formulae.

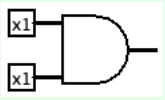
2.4.3 Our Gates

Let's start talking about some digital circuits. We start with some gates.

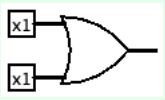
Definition 2.46 (NOT gate)**.** The *NOT gate* (or inverter) is a gate which takes low voltage to high voltage and vice versa.



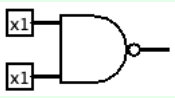
Definition 2.47 (AND gate)**.** The *AND gate* is a gate which takes high voltage if and only if both of its inputs are high voltage.



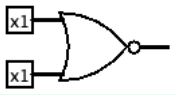
Definition 2.48 (OR gate)**.** The *OR gate* is a gate which takes high voltage if and only if at least one of its inputs are high voltage.



Definition 2.49 (NAND gate)**.** The *NAND gate* is a gate which takes high voltage if and only if at least one of its inputs are low voltage.

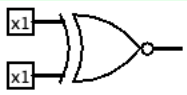


Definition 2.50 (NOR gate)**.** The *NOR gate* is a gate which behaves like negative of the OR gate.



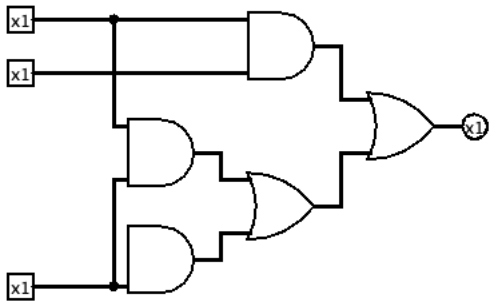
Definition 2.51 (XOR gate)**.** The *XOR gate* is a gate which has high voltage if and only if exactly one of the inputs is high-voltage.

Definition 2.52 (XNOR gate)**.** The *XNOR gate* is a gate which has high voltage if and only if its inputs have the same voltage.



2.4.4 Building Circuits

We can connect these circuits together in fun ways.



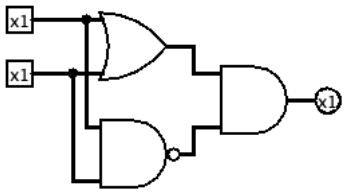
Reading off of this circuit corresponds gives the formula

$$(p \wedge q) \vee ((p \wedge r) \vee (q \wedge r)).$$

Let's see an example problem.

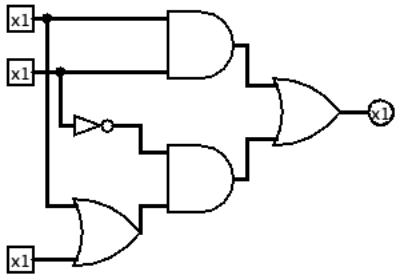
Exercise 2.53. We build a circuit with two switches such that either light switch turns the lightbulb on or off.

Proof. It is not too implausible to think that we can do this with a formula because we are essentially asking for the number of "on" light-switches to be odd to give the light-bulb on. Here is a circuit which works.



It corresponds to the formula $(p \vee q) \wedge \neg(p \wedge q)$. This is essentially an XOR gate. ■

It is possible for circuits to be inefficient. For example, consider the following circuit.



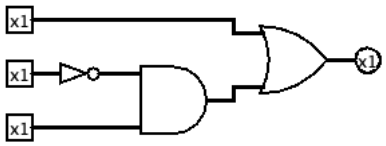
This can be simplified down. Indeed, this corresponds to the formula

$$(p \wedge q) \vee (\neg q \wedge (p \vee r)).$$

We can argue that this is equivalent to

$$p \vee (\neg q \wedge r)$$

by doing some simplification. Thus, we can build the following simpler circuit.



So this is an application of simplifying formulae.

Remark 2.54. We remark that any formula can be written using $\uparrow$, which corresponds to the fact that all circuit functions could be written in only NAND gates.

Remark 2.55. One can build the same theory of equivalence and so on of circuits by porting over the theory from formulae/truth functions to circuits.

2.5 February 18

Here we go.

2.5.1 Algorithms

Today we start talking about algorithms.

Definition 2.56 (Algorithm)**.** An *algorithm* is a procedure or set of rules or instructions that will tell us how to preform successive steps, which will complete a task after a finite number of steps.

This definition can be made more rigorous, but we will not bother for this class.

We have already seen some algorithms so far.

Example 2.57. To determine if a formula is satisfiable, we employ the following algorithm.

1. Fix a formula $\varphi \in \mathcal{L}(\{p_1, \ldots, p_n\})$.
2. Fix some order of the valuations.
3. Choose the first valuation V.
4. Compute $\hat{V}(\varphi)$ by a recursive procedure.
5. If $\hat{V}(\varphi) = 1$, then φ is satisfiable.
6. Otherwise, choose the next valuation V and go back to step 4.
7. If we finish the truth table and never return satisfiable, then φ is unsatisfiable.

Remark 2.58. The above algorithm requires something like 2^n total valuations to check, making the worst-case (if unsatisfiable) requiring 2^n time. This makes this run in "exponential" time.

Remark 2.59. There are other algorithms for satisfiability. The textbook uses the semantic tableaux. We will shortly talk about resolution.

2.5.2 Another CNF Algorithm

For resolution, we will need to start in conjunctive normal form. Recall the definition, as follows.

Definition 2.60 (Conjunctive normal form)**.** A formula φ is in *conjunctive normal form* if and only if it is a conjunction of disjunctions.

We do already have an algorithm for this: find the disjunctive normal form $\neg\varphi$ and then negate and apply De Morgan's laws.

However, the disjunctive normal form requires us to in advance have a truth table, which needs a notion of semantics. Here is an algorithm which is more syntactic.

Proposition 2.61. Fix a formula φ. We put φ into conjunctive normal form.

1. Remove $\leftrightarrow$s: for each $\leftrightarrow$ starting from the left, replace each subformula of the form $(\alpha \leftrightarrow \beta)$ with $((\alpha \to \beta) \wedge (\beta \to \alpha))$.
2. Remove $\to$s: for each $\to$ starting from the left, replace each subformula $(\alpha \to \beta)$ with $(\neg\alpha \vee \beta)$.

At this point, the only legal connectives are $\wedge$, $\vee$, and $\neg$. In particular, if $\neg\psi$ is a subformula, then ψ is either of the form $(\alpha \wedge \beta)$ or $(\alpha \vee \beta)$ or $\neg\alpha$.

3. Drive $\neg$ inside: for each subformula of the form $\neg(\alpha \vee \beta)$ starting from the left, replace it with $(\neg\alpha \wedge \neg\beta)$.
4. Drive $\neg$ inside: for each subformula of the form $\neg(\alpha \wedge \beta)$ starting from the left, replace it with $(\neg\alpha \vee \neg\beta)$.
5. Drive $\neg$ inside: for each subformula of the form $\neg\neg\alpha$ starting from the left, replace it with α.

So now $\neg$ only applies to propositions, so we are only applying $\wedge$ and $\vee$ to literals. We need to normalize to get conjunctives of disjunctions; the only way we violate being in conjunctive of normal form is if we have a subformula which is a $\vee$ of non-literals, and one of these non-literals had better have $\wedge$ lest we just break apart into separate disjunctive terms.

6. Distribute: for each subformula of the form $(\alpha \vee (\beta \wedge \gamma))$ starting from the left, replace it with $((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$.
7. Distribute: for each subformula of the form $((\alpha \wedge \beta) \vee \gamma)$ starting from the left, replace it with $(((\alpha\gamma) \wedge (\beta \vee \gamma))$.
8. Go back to step 6 until done.

We will not prove that this works, but there are remarks throughout the description. More formally, we are defining a recursive function $\mathrm{CNF} : \mathcal{L}(P) \to \mathcal{L}(P)$ with many cases. For completeness, here is the definition.

1. $\mathrm{CNF}(p) = p$ for each $p \in P$.
2. $\mathrm{CNF}(\varphi \leftrightarrow \psi) = (\neg\,\mathrm{CNF}(\alpha) \vee \mathrm{CNF}(\beta)) \wedge (\neg\,\mathrm{CNF}(\beta) \vee \mathrm{CNF}(\alpha))$.
3. $\mathrm{CNF}(\varphi \to \psi) = \neg\,\mathrm{CNF}(\alpha) \vee \mathrm{CNF}(\beta)$.
4. $\mathrm{CNF}(\varphi \wedge \psi) = \mathrm{CNF}(\varphi) \wedge \mathrm{CNF}(\psi)$.
5. $\mathrm{CNF}(\neg(\alpha \wedge \neg\beta)) = \neg\,\mathrm{CNF}(\alpha) \vee \neg\,\mathrm{CNF}(\beta)$.
6. $\mathrm{CNF}(\neg(\alpha \vee \neg\beta)) = \neg\,\mathrm{CNF}(\alpha) \wedge \neg\,\mathrm{CNF}(\beta)$.
7. $\mathrm{CNF}(\neg\neg\varphi) = \mathrm{CNF}(\varphi)$.
8. $\mathrm{CNF}(\alpha \vee (\beta \wedge \gamma)) = \mathrm{CNF}(\alpha \vee \beta) \wedge \mathrm{CNF}(\alpha \vee \gamma)$.

9. $\mathrm{CNF}((\alpha \wedge \beta) \vee \gamma) = \mathrm{CNF}(\alpha \vee \gamma) \wedge \mathrm{CNF}(\beta \vee \gamma)$.

It is not obvious that the above conditions fully determine CNF recursively (in fact, the last two steps make CNF not well-defined because we might have a subformula of the form $(\alpha \wedge \beta) \vee (\gamma \wedge \delta)$), but there is a small justification present in the definition of the algorithm, and it is not too hard to imagine how to fix this. We will not fix this formally out of laziness.

Remark 2.62. The above algorithm need not give the same conjunctive normal form as with the truth table approach. The above algorithm is purely syntactic.

Let's see some examples.

Exercise 2.63. We put $\varphi := \neg(p_1 \wedge \neg(\neg p_2 \wedge (p_3 \to p_4)))$ in conjunctive normal form.

Proof. We have the following computation. To start, we get rid of $\to$.

$$\varphi \equiv \neg(p_1 \wedge \neg(\neg p_2 \wedge (p_3 \to p_4)))$$

Next we push $\neg$s inside and cancel out double negations.

$$\begin{aligned}
\varphi &\equiv \neg(p_1 \wedge \neg(\neg p_2 \wedge (\neg p_3 \vee p_4))) \\
&\equiv \neg p_1 \vee \neg\neg(\neg p_2 \wedge (\neg p_3 \vee p_4)) \\
&\equiv \neg p_1 \vee \neg(\neg\neg p_2 \vee \neg(\neg p_3 \vee p_4)) \\
&\equiv \neg p_1 \vee (\neg\neg\neg p_2 \wedge \neg\neg(\neg p_3 \vee p_4)) \\
&\equiv \neg p_1 \vee (\neg\neg\neg p_2 \wedge \neg(\neg\neg p_3 \wedge \neg p_4)) \\
&\equiv \neg p_1 \vee (\neg\neg\neg p_2 \wedge (\neg\neg\neg p_3 \vee \neg\neg p_4)) \\
&\equiv \neg p_1 \vee (\neg p_2 \wedge (\neg\neg\neg p_3 \vee \neg\neg p_4)) \\
&\equiv \neg p_1 \vee (\neg p_2 \wedge (\neg p_3 \vee \neg\neg p_4)) \\
&\equiv \neg p_1 \vee (\neg p_2 \wedge (\neg p_3 \vee p_4)).
\end{aligned}$$

Now we distribute. We see

$$\begin{aligned}
\varphi &\equiv \neg p_1 \vee (\neg p_2 \wedge (\neg p_3 \vee p_4)) \\
&\equiv (\neg p_1 \vee \neg p_2) \wedge (\neg p_1 \vee (\neg p_3 \vee p_4))) \\
&\equiv \boxed{(\neg p_1 \vee \neg p_2) \wedge (\neg p_1 \vee \neg p_3 \vee p_4)},
\end{aligned}$$

which is what we wanted. ∎

Exercise 2.64. We convert $\varphi := (p_1 \wedge p_2) \vee (\neg p_1 \wedge (\neg p_2 \wedge p_3))$ to conjunctive normal form using the algorithm.

Proof. We only have to distribute right now, so we compute

$$\begin{aligned}
\varphi &= (p_1 \wedge p_2) \vee (\neg p_1 \wedge (\neg p_2 \wedge p_3)) \\
&\equiv ((p_1 \wedge p_2) \vee \neg p_1) \wedge ((p_1 \wedge p_2) \vee (\neg p_2 \wedge p_3)) \\
&\equiv ((p_1 \vee \neg p_1) \wedge (p_2 \vee \neg p_1)) \wedge ((p_1 \wedge p_2) \vee (\neg p_2 \wedge p_3)) \\
&\equiv ((p_1 \wedge p_2) \vee \neg p_1) \wedge ((p_1 \vee (\neg p_2 \wedge p_3) \wedge (p_2 \vee (\neg p_2 \wedge p_3)))) \\
&\equiv ((p_1 \wedge p_2) \vee \neg p_1) \wedge ((((p_1 \vee \neg p_2) \wedge (p_1 \vee p_3)) \wedge ((p_2 \vee \neg p_2) \wedge (p_2 \vee p_3)))),
\end{aligned}$$

which finishes. ∎

2.6 February 23

Today we are talking about resolution.

2.6.1 Resolution

The goal of resolution is to compute if a formula is satisfiable. So far we have a fairly semantic algorithm, which is the truth table algorithm: list out all possibilities and then check each.

The big-picture question is "How can we get a machine to reason?" For example, once we know about satisfiability, we can ask the machine if φ is valid by seeing if $\neg\varphi$ is satisfiable. So to get a machine to reason, we can ask if a formula φ follows from the premises $\varphi_1, \dots, \varphi_n$ if and only if

$$(\varphi_1 \wedge \cdots \wedge \varphi_n) \to \varphi$$

is valid if and only if

$$\neg\big((\varphi_1 \wedge \cdots \wedge \varphi_n) \to \varphi\big)$$

is not satisfiable.

The story so far is that we already have a couple ways to convert a formula into conjunctive normal form: either run the negation of the DNF algorithm, or we had the algorithm from last class which was more syntactic and essentially massaged all the symbols in place (where the key step was to apply distribution of $\vee$ over $\wedge$ to group clauses together). At a high level, CNF looks like

$$(\ell_{1,1} \wedge \cdots \ell_{1,m_1}) \vee \cdots \vee (\ell_{n,1} \wedge \cdots \ell_{n,m_n}),$$

where the $\ell_\bullet$ are literals. To make this formula true, we need one literal in each clause to be true.

For resolution, the key idea is as follows.

Idea 2.65. Suppose φ contains some subformula of the form $(p \vee \psi_1) \wedge (\neg p \vee \psi_2)$. Then $\psi_1 \vee \psi_2$ follows.

Here's an example of this idea in action.

Exercise 2.66. We do resolution on the formula

$$\varphi := (p_1 \vee p_3 \vee p_4) \wedge (\neg p_1 \vee \neg p_2 \vee p_4) \wedge (p_1 \vee \neg p_2 \vee \neg p_3).$$

Proof. Observe that the clause $(p_1 \vee p_3 \vee p_4)$ and $(\neg p_1 \vee \neg p_2 \vee p_4)$ contain both p_1 and $\neg p_1$. So if both of these clauses are to be true, we had better not be checking the p_1 box, so it follows that

$$\psi := p_3 \vee p_4 \vee \neg p_2 \vee p_4.$$

Explicitly, we claim $\varphi \models \psi$. We have two cases.

- If $V(p_1) = 0$, then $\hat{V}(\varphi) = 1$ forces $\hat{V}((p_1 \vee p_3 \vee p_4)) = 1$ forces $\hat{V}((p_3 \vee p_4))$, so $\hat{V}(\psi) = 1$.
- If $V(p_1) = 1$, then $\hat{V}(\varphi) = 1$ forces $\hat{V}((\neg p_1 \vee \neg p_2 \vee p_4)) = 1$ forces $\hat{V}((\neg p_2 \vee p_4))$, so $\hat{V}(\psi) = 1$.

To make our lives easier, we will take ψ and remove the redundancy p_4 to get $\psi = p_3 \vee p_3 \vee \neg p_2$.

This idea gives the following definition.

Definition 2.67 (Resolvent)**.** Fix φ a formula. If we have clauses $C_1 = p \vee \psi_1$ and $C_2 = \neg p \vee \psi_2$ in φ (or some rearrangement of p sitting anywhere in C_1 and C_2), then we call $\psi_1 \vee \psi_2$ a *resolvent* of φ by p.

Remark 2.68. The argument given above generalizes to show that any resolvent ψ of φ is a logical consequence of φ.

Here are the other resolvents of φ.

- In $(\neg p_1 \vee \neg p_2 \vee p_4)$ and $(p_1 \vee \neg p_2 \vee \neg p_3)$, we see we have p_1 and $\neg p_1$, so we get the resolvent $p_1 \vee p_4 \vee \neg p_2$.
- In $(p_1 \vee p_3 \vee p_4)$ and $(p_1 \vee \neg p_2 \vee \neg p_3)$, we see we have p_3 as well as $\neg p_3$, so we get the resolvent $\neg p_2 \vee p_4 \vee \neg p_3$.

Because all of our resolvents are logical consequences, we see that

$$\begin{aligned}\varphi \equiv (p_1 \vee p_3 \vee p_4) \wedge (\neg p_1 \vee \neg p_2 \vee p_4) \wedge (p_1 \vee \neg p_2 \vee \neg p_3) \\ (p_3 \vee p_3 \vee \neg p_2) \wedge (p_1 \vee p_4 \vee \neg p_2) \wedge (\neg p_2 \vee p_4 \vee \neg p_3).\end{aligned}$$

We now do more resolution on this new formula. The clauses $(p_3 \vee p_3 \vee \neg p_2)$ and $(\neg p_2 \vee p_4 \vee \neg p_3)$ resolve to $p_4 \vee \neg p_2$ by p_3. But now, when we write

$$\begin{aligned}\varphi \equiv (p_1 \vee p_3 \vee p_4) \wedge (\neg p_1 \vee \neg p_2 \vee p_4) \wedge (p_1 \vee \neg p_2 \vee \neg p_3) \\ (p_3 \vee p_3 \vee \neg p_2) \wedge (p_1 \vee p_4 \vee \neg p_2) \wedge (\neg p_2 \vee p_4 \vee \neg p_3) \\ (p_4 \vee \neg p_2),\end{aligned}$$

we can check that there are no more resolvents to add. We call this formula $\operatorname{res}(\varphi)$, with the following definition.

Definition 2.69 (Resolution)**.** Fix a formula φ. Then we define the *resolution* of φ by $\operatorname{res}(\varphi)$ to be φ closed under adding in resolvents.

Anyways, this finishes the resolution algorithm by the following relevant theorem; in particular, we see quickly that φ is satisfiable. ■

And here is the relevant theorem.

Theorem 2.70. A formula φ is unsatisfiable if and only if some clause of $\operatorname{res}(\varphi)$ has both p and $\neg p$ as clauses. This conclusion is called an overt contradiction.

Proof. We omit this. The easy direction is that, if $\operatorname{res}(\varphi)$ has both p and $\neg p$ as clauses, then of course there is no way to satisfy $\operatorname{res}(\varphi)$. ■

Remark 2.71. One might complain that this algorithm is quite inefficient. This is essentially to make it simpler; there are many optimizations that one could make by trying to add in fewer or more useful resolvents.

Example 2.72. Exercise 2.66 provides a formula φ with $\operatorname{res}(\varphi)$ which has no overt contradiction and hence is satisfiable.

Le's see another example of resolution.

Exercise 2.73. We apply resolution to

$$\varphi := (p_1 \vee p_3) \wedge (\neg p_1 \vee p_2) \wedge (p_1 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2).$$

Proof. We optimize with the following steps.

- The clauses $(\neg p_1 \vee p_3)$ and $(p_1 \vee \neg p_3)$ gives p_1.
- From $(\neg p_1 \vee p_2)$ and $(\neg p_1 \vee \neg p_2)$, we get $\neg p_1$.

But earlier we had p_1, so we have a contradiction. ■

In truth, there are likely many resolvents we would have to add in before finding the two resolvents given above, but we omitted them for brevity.

2.6.2 Complexity

It is somewhat annoying that determining if a formula φ is satisfiable requires a lot of computation. Consider the following remarks.

- If φ is satisfiable, then technically the truth table method merely requires finding the correct line of the truth table instead of the whole truth table.
- However, if φ is unsatisfiable, then we have to compute everything.
- If φ is unsatisfiable, then technically we only need to compute resolvents until we actually get an overt contradiction.
- However, if φ is satisfiable, then we have to compute everything.

The bad news is that the worst-case performance of both resolution and creating a truth table is exponential. Essentially this is because we have to continue checking for new resolvents, and each step can potentially add lots and lots of resolvents.

Example 2.74. With 10 variables, this would require $2^{10} = 1024$ rows of a truth table.

Here is the question we are interested in: can we make satisfiability faster?

Question 2.75. Is there an algorithm which runs in time bounded by a polynomial in n which can determine if a formula with n connectives is satisfiable?

Question 2.75 is perhaps the most famous unsolved problem in all of theoretical computer science. It is called the P vs. NP problem. Sadly, most people do not think that such an algorithm exists.

2.7 February 28

And we are back on the road.

2.7.1 More on Complexity

Last time we were talking about complexity and in particular Question 2.75, complaining that determining satisfiability seems to have an exponential blow-up in steps as the number of atomic formulae increases. For example, the truth-table method to compute satisfiability requires on the order of 2^n steps; most researchers do not think there is a way to do this much faster.

So let's talk more about this P vs. NP problem.

Definition 2.76 (P)**.** A problem is in *P* if and only if there is a deterministic polynomial-time algorithm to solve the problem. Namely, the algorithm run-time is polynomial in the input size.

Example 2.77. Deciding if a number is even runs in essentially constant time: simply check if the last digit is in $\{0, 2, 4, 6, 8\}$.

Example 2.78. There is an algorithm which can determine if a positive integer is prime in polynomial time.

Definition 2.79 (NP)**.** A problem is in *NP* if and only if there is a non-deterministic polynomial-time algorithm to solve the problem.

Roughly, a non-deterministic algorithm is allowed to guess and be lucky. Here are some examples.

Example 2.80. To check satisfiability of some formula φ, we can just guess a valuation and compute if $\hat{V}(\varphi) = 1$. Doing this check only runs in polynomial (in fact, linear) time with respect to the length of φ. Because life is easy after making this guess, we call the algorithm "non-deterministic" polynomial-time, where the adjective non-deterministic refers to the guess.

Example 2.81. It is difficult to solve a Sudoku puzzle—and there is an exponential blow-up in solving Sudoku as the grid gets bigger—but checking that the Sudoku works is comparatively easy—and checking does not get much harder as the problems gets bigger. Thus, there is a non-deterministic polynomial-time algorithm (guess some filled-in grid and check).

If we wanted to make satisfiability more analogous to Sudoku, we can transform the Sudoku problem as follows: given a partially filled-in $n^2 \times n^2$ Sudoku grid, determine if we can fill in the grid in some valid way. In particular, this is a binary-output and scalable problem like with satisfiability.

Remark 2.82. If we could show that merely satisfiability is in P, this would in fact show that all problems of NP would live in P. In particular, any problem in NP can be "easily" translated (i.e., translated in polynomial-time) into a satisfiability question.

The above remark is why satisfiability alone is enough to answer Question 2.75.

2.7.2 Scheduling via Satisfiability

Suppose we are trying to schedule lectures, so we are given a list of possible rooms and some list of lectures and some list of professor availabilities. Here would be our constraints.

1. Each lecture should be scheduled in some room in some time slot.
2. No two lectures should happen in the same room at the same time.
3. Two lectures given by the same professor cannot be given at the same time.
4. A lecture should not be scheduled at a time when the professor giving the lecture is unavailable.

To turn this into a satisfiability problem, we build some notation. We set $s_{i,j,k}$ to mean that lecture i is to be scheduled in room j during time slot k.

Remark 2.83. Availabilities and who gives lectures is not in the control of the scheduler. The only thing the scheduler can change is when the lecture happens, so the only atomic formulae we will deal with are $s_{i,j,k}$.

We now translate our constraints into propositional logic.

1. "Each lecture should be scheduled in some room in some time slot" becomes

$$\bigwedge_i \left(\bigvee_{j,k} s_{i,j,k} \right).$$

   Namely, for each lecture i, some $s_{i,j,k}$ must be true for some room j and time slot k.

2. "No two lectures should happen in the same room at the same time" becomes

$$\bigwedge_{i \neq i', j, k} \neg(s_{i,j,k} \wedge s_{i',j,k}).$$

   In other words, for any two distinct lectures i and i', they are not both in the same room j and the same time slot k.

3. "Two lectures given by the same professor cannot be given at the same time" becomes

$$\bigwedge_{\substack{i \neq i', \, i, i' \text{ have the same professor} \\ j,k}} \neg(s_{i,j,k} \wedge s_{i',j,k}).$$

4. "A lecture should not be scheduled at a time when the professor giving the lecture is unavailable" becomes

$$\bigwedge_{\substack{i,j,k \\ \text{prof for } i \text{ unavailable at } k}} \neg s_{i,j,k}.$$

   In other words, the professor for lecture i should not be lecturing when available.

So to solve scheduling, we just hand over these constraints to a satisfiability solver, and it'll do it.

Remark 2.84. Scheduling turns out to not be so bad in comparison to worst-case scenarios for satisfiability. In particular, satisfiability is not so bad when there are "few enough" letters and the formula we are testing is not so bad.

Remark 2.85. Professor Holliday would like someone to use a satisfiability solver to run wedding planning (e.g., for dinner placement).

Theme 3
Natural Deduction

3.1 March 2

Welcome back everybody.

3.1.1 Proof

We are moving from "semantic" theory to more "proof-based" theory.

Recall that we already had a way to determine if an argument $\varphi_1, \dots \varphi_n \models \psi$ is by truth table: we simply check that the formula

$$(\varphi_1 \wedge \cdots \wedge \varphi_n) \to \psi$$

is a tautology by truth table. However, this is pretty annoying when there are lots of propositions floating around and in fact impossible when there are infinitely many variables. So we will introduce the idea of formal proof.

Here is an example of a formal proof for some simple fact.

Proposition 3.1. Fix a and b real numbers. If $0 < a < b$, then $a^2 < b^2$.

Proof. We start by assuming $0 < a < b$. Because $0 < a$, we can multiply both sides of $a < b$ by a, which gives

$$a^2 < ab.$$

Similarly, $0 < a < b$ implies $0 < b$, so we can multiply both sides of $a < b$ by b, so

$$ab < b^2.$$

It follows $a^2 < ab$ and $ab < b^2$, so we do get $a^2 < b^2$, which is what we wanted. ■

Remark 3.2. In the above proof, we showed a statement of the form $\varphi \to \psi$ by assuming φ and proving ψ.

To formalize the above, we will diagram our deduction to make a "sub-proof." This is called a Fitch-style formal proof, and they are pretty.

1	⊢	
2	$0 < a < b$	
⋮	⋮	
n	$a^2 < b^2$	
$n+1$	$(0 < a < b) \to a^2 < b^2$	$\to$Intro, 2, n

More generally speaking, a Fitch-style proof with premises $\varphi_1, \ldots, \varphi_n$ and proving ψ looks like the following.

1	φ_1
⋮	⋮
n	φ_n
⋮	⋮
m	ψ

The vertical dot empty spaces are to be filled with sub-proofs or formulae. Our goal is to put everything into this formal structure.

Here is an example of a rule, which is what we used above.

Definition 3.3 ($\to$Introduction)**.** If we have a sub-proof with assumption α and concludes β, then we can pop out of the sub-proof and add $\alpha \to \beta$ with the rule $\to$ with the named lines.

This looks like the following.

	⊢		
i		φ	
⋮		⋮	
j		ψ	
⋮	⋮		
k	$\varphi \to \psi$		$\to$Intro, i, j

3.1.2 Elimination of Conditionals

We can use conditionals to prove other conditionals. For example, suppose we have proven the following lemma.

Lemma 3.4. Fix a and b real numbers. If $0 < a < b$, then $a^2 < b^2$.

Then we can prove the following.

Lemma 3.5. Fix c and d real numbers. If $0 < d < c$, then $(c-d)^2 < c^2$.

Proof. As before, if $0 < c - d < c$, then $(c-d)^2 < c^2$ by the lemma. However, $d < c$ implies $0 < c - d$, and $0 < d$ then gives $0 < c - d < c$. So it does follow $(c-d)^2 < c^2$. ■

This rule is proving the antecedent from the consequent.

Definition 3.6 (→Elimination)**.** If we have shown $\alpha \to \beta$ in some line i and have shown α in some line j, then we can conclude β from $\to$ *elimination*, citing i and j.

This looks like the following.

⋮	⋮	
i	$\alpha \to \beta$	
⋮	⋮	
j	α	
⋮	⋮	
k	β	→Elim, i, j

We can even reverse these as follows.

⋮	⋮	
i	α	
⋮	⋮	
j	$\alpha \to \beta$	
⋮	⋮	
k	β	→Elim, j, i

The point is to get the computer to be able to check this.

Example 3.7. This sort of reasoning is pretty intuitive: if we read "If you received Form 1098T, you may be eligible for a tax credit or deduction" and then we have received Form 1098T, then we can conclude that we may eligible. Here is a formalization.

1		
2	received 1098T → may be eligible for tax credit	
3	received 1098T	
4	may be eligible for tax credit	→Elim, 3, 2

3.1.3 Reiteration

Let's return to the proof of Lemma 3.5. It used both $\to$ introduction and elimination. It looked like the following.

1	⊢	
2	$0 < c - d < c \to (c-d)^2 < c^2$	
3	$\quad 0 < d < c$	
⋮	$\quad 0 < c - d < c$	$(*)$
n	$\quad 0 < c - d < c \to (c-d)^2 < c^2$	Reit, 2
$n+1$	$\quad (c-d)^2 < c^2$	$\to$Elim, 3, 2
$n+2$	$0 < d < c \to (c-d)^2 < c^2$	$\to$Intro, 3, $n+1$

The point of our discussion is our use of the reiteration rule on line 5. Logically speaking, we note that the sub-proof technically lives in a different world than the rest of the outer proof, so we have a rule that says we can pull from the outside world into a sub-proof.

Here is our formal definition.

Definition 3.8 (Open)**.** A subproof is *open* starting from its first assumption until the conclusion of the subproof.

Definition 3.9 (Directly)**.** A formula φ occurs *directly* in a subproof if and only if φ occurs in the main column of S.

Example 3.10. In $(*)$, the lines 2 through 6 are open in the subproof.

Definition 3.11 (Reiteration)**.** The *reiteration* tells us that we may add φ to any subproof currently in the subproof.

Non-Example 3.12. The following is an incorrect use of reiteration.

1	⊢	
2	$\quad$ We are going to the party.	
3	$\quad$ We will have a good time.	
4	We will have a good time.	Reit, 3

Namely, this line does not live in the sub-proof, so we cannot pull it out.