12A: Introduction to Logic

Nir Elber

Spring 2022

Contents

THEME 1: PROPOSITIONAL LOGIC — SYNTAX

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 interestsed 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	$\rightarrow$
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

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

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]
- 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.

[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.

- 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 recommended to do another pass on the reading after lecture.
- There are weekly problem sets, released on Mondays (startig next Monday) and due on Sunday midnights. 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 Propostions 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.

Proposition

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.

Connective

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 theif 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 dice roll is odd, q to be the event that a dice roll is $\{1, 2\}$, and r to be the vent that a dice roll 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 are 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 out notion of truth value.

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.

Unary, binary

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

Truth-functional

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 or not 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.

Truth-functional

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.

Truth-functional

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.

Material conditional ($\to$)

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 convesely).

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."

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 capiral of Belgium.

Note that this argument is valid, even though the second premise is simply false. Giving false premises provides no gaurantees 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.

Sound

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 capiral 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 caree 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 gauranteed 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 standardly 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 actaully 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 parantheses in cases of ambiguity.

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	$p \wedge q$	*conjuation* 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 *antecdent* 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 parantheses 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 bit 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 taht 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 langauge.

Symbols

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 $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.
- A set of parantheses $\{(,)\}$.

1.4.3 Towards Formulae

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

Expression

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 $\langle \rightarrow, \neg, (p\rangle$ 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 "$\rightarrow \neg p($" for the expression $\langle \rightarrow, \neg, (p\rangle$.

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.

Formula, I

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.

Formula, II

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 $(\varphi_1 \rightarrow \varphi_2)$ 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 of 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 $\langle \varphi_1, \dots, \varphi_n\rangle$ satisfying the following conditions.

(a) φ_k may be an atopic formula.

(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 contruction sequence for $(\neg p_1 \to (p_2 \vee p_3))$.

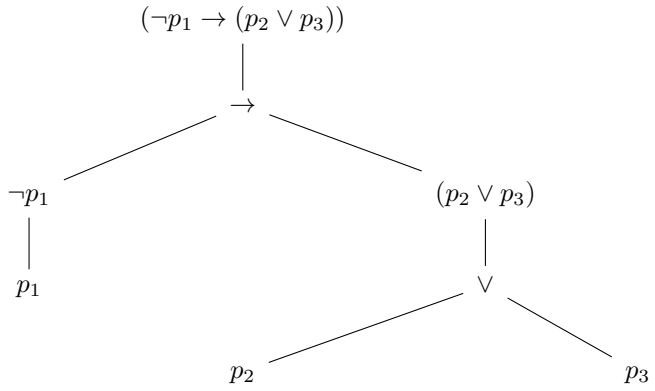
The point is that constructon sequences let us define formulae.

Formula

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.

Formula, again

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 propoerties (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 gauranteed 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 parantheses.

Proof. We proceed by induction. Let S be the set of expressions that have the same number of left and right parantheses, 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 parantheses (and therefore x right parantheses) and ψ has y left parantheses (and therefore y right parantheses). Then we check what happens with our connectives.

- We see that $\neg\varphi$ has the same number left/right parantheses 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 parantheses, 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 paranthesis 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 are 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.

Subformula

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.

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

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.

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

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

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 operatinos 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 paranthesis.
- 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.

Depth

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.

Example 1.86. We have the following computation.

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

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

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

THEME 2: PROPOSITIONAL LOGIC — SEMANTICS

2.1 February 2

Here we go again.

2.1.1 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 2.1. We inted 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 2.2. 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 2.3. For the remainder of the class, we will replace "T" with 1 and "F" with 0.

Example 2.4. 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 2.5. 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. We do not do this to make better analogy with $\vee$, as follows.

Example 2.6. 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 2.7. 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 2.8. 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.

2.1.2 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.

Valuation

Definition 2.9 (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 2.10. 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 2.11. The following describes a valuation for $P = \{p, q\}$.

p	1
q	0

Remark 2.12. The "valuation" $\hat{V}$ is meant to be an abbreviation of "the truth value of."

Inutitively, 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 2.13. 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 choiecs (simultaneously) to determine a unique valuation V.

Now here is our promised idea.

Idea 2.14. Recursion can extend any valuation $V : P \to \{0,1\}$ uniquely to all of $\hat{V} : \mathcal{L}(P) \to \{0,1\}$.

Example 2.15. 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 2.16. 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.

Satisfies

Definition 2.17 (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 2.18. 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

2.1.3 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 2.19. 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.

Argument form

Definition 2.20 (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.

Valid

Definition 2.21 (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 2.22. 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 2.23. 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.