COMP102P

Language and Logic

Taught by Robin Hirsch

Notes compiled by students

Chapter 1

Propositional Logic

Propositional logic is reasonably straight forwards. You make a bunch of assertions and test if they are true or not. Let's check out the syntax whilst trying to keep it nice and simple.

1.1 Syntax

An element or assertion in logic (e.g., "it is sunny today", "I slept through my lecture") is called a *proposition*. They're normally given single letters like p or q (p for proposition, right?), but as with all things maths, definitions change when the author feels like it.

If you have the proposition on its own, it's known as an *atomic formula*. However, you can connect more than one proposition together using a *binary connective*, which are a way of creating a new proposition from two propositions (there's also a unary connective, which is the not connective, or the negator ($\neg$)). Here, let's try this out, shamelessly ~~stolen~~ borrowed from Hirsch's notes.

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

1.1.1 Backus-Naur Form

Backus-Naur form (BNF) is used to define syntax, and in this case we'll use it to properly define what is a proper propositional formula and what isn't. Okay, here goes:

`proposition ::=` $p_0|p_1|p_2|\ldots$
`bin_connective ::=` $\wedge|\vee|\rightarrow|\leftrightarrow$
`formula ::= proposition |` $\neg$ `formula | (formula bin_connective formula)`

So, er, the `::=` symbol is just assignment like in programming. And then the | symbol just means it can be either of the definitions. Simple!

A couple of things to note. See how `formula` can be defined in terms of itself? That allows you to have rather complex formulas that are still valid. The other thing to note

is that if two formulas are used with a binary connective, they **must** be surrounded by brackets to be valid. No exceptions.

1.1.2 Convert English to Logic

We may need to know how to convert a normal sentence into a weird logical formula, so here is a couple of examples.

Example. *If it rains today, I won't go to the lecture.*

r–It rains today.
g–I go to the lecture.

$(r \rightarrow \neg g)$

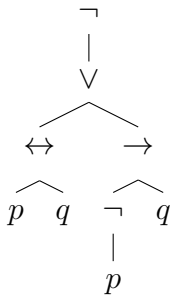
It's pretty easy to be honest, just remember to define your *propositions*.

1.1.3 Parsing

So, in a roundabout way, the notes tell you that propositional formulas can only contain a certain selection of characters, called an *alphabet*. This is limited to:

$$\Sigma = \{\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, (,), p_0, p_1, \dots\}$$

So with that, it's kinda easy to write your own parser, right? Probably, you've already done it by now for coursework! But just as a refresher, you need to use a parse tree. Let's make a formula and call it ϕ of all things. Let $\phi = \neg((p \leftrightarrow q) \vee (\neg p \rightarrow q))$. This is what its parse tree would look like:



That's a little complicated, but you can see where it's coming from right? It shows very easily how the logic is combined to form the full formula.

1.2 Semantics

Semantics is all about the meaning apparently. This involves assigning truthy or falsy values to the propositions in order to make a formula true or false. Yeah okay, it's a bit of a long winded way of doing things, but welcome to theory! You use something called a *valuation*, a function normally represented by the Greek letter iota (ι). So, an example of what this coule be like would be:

$$\begin{aligned} \iota : p &\longmapsto \top \\ q &\longmapsto \bot \\ r &\longmapsto \bot \end{aligned}$$

So, you got that. So what? In the world of theory where everything is defined properly, you have to extend this to a *truth function*, which will pretty much always be υ. Dem computer scientists love Greek letters. The truth function can be applied to any propositional formula that uses the letters you provided, which makes it kinda neat. You can then say that, if using the valuation above, that $\upsilon(p) = \top$ for instance. There's a bunch of 'base cases' for using this with bigger functions, which are pretty much common sense, so why am I writing them? (oh yeah, borrowing them from the notes, again) Okay, so let's say you made some propositional formulas, and named them ϕ and ψ. You then work out whatever $\upsilon(\phi)$ and $\upsilon(\psi)$ are, or maybe you don't, and then these following things apply ($\iff$ means iff):

$$\begin{aligned} \upsilon(\neg\phi) = \top &\iff \upsilon(\phi) = \bot \\ \upsilon(\phi \vee \psi) = \bot &\iff \upsilon(\phi) = \upsilon(\psi) = \bot \\ \upsilon(\phi \wedge \psi) = \top &\iff \upsilon(\phi) = \upsilon(\psi) = \top \end{aligned}$$

Chapter 2

Induction

Induction is all about proving some property for a given set. There are three types of induction you need to know about for this part of the course: Weak arithmetic induction, strong arithmetic induction and structured induction on formulas. Induction is used to prove those super-handy arithmetic sum formulae, amongst other stuff!

2.1 Weak Induction

Weak induction allows you to prove something, a property, about the set of natural numbers $\mathbb{N}$ (0,1,2,..) by going through a sequence of steps:

1. **Prove that the property holds true for some *base case*** — usually $n = 0$ or $n = 1$.

   This bit's pretty friendly, as you just need to substitute in the value on the left-hand side and show it's the same as if you put it in the right-hand side!

2. **Proving that the property holds true for the *inductive step*** — here we form an *inductive hypothesis* where you assume that the property holds true for $n = k$, which we then use to try and show that if the property holds for $n = k$ then it must also hold for $n = k + 1$ as a result.

   This is the trickier bit! Here you generally take the expression for $k+1$ and separate it from that of k, as it's assumed that the property is true for k. Once separated, you just need to do to do a bit of rearrangement magic to show you can the property for $k + 1$ using the property for k and the expression for $k + 1$.

3. **Conclude that the property holds for all $n \in \mathbb{N}$ greater than the base case as it is proven to hold for the *base case* and through the *inductive step*** — this is just a bit of formal maths talk saying, look we've proved it works if the previous one works and because we proved the first one works they should all work!

A more formal explanation: In general, suppose that we wish to prove some property $P(n)$ for all $n \in \mathbb{N}$ where $n \geq b$ for some *base case* b. To do this we must prove first the *base case* and then the *inductive step*. Once we have proven *both* of these we are able to conclude that $P(n)$ holds for all $n \in \mathbb{N}$, $n \geq b$.

2.1.1 Proving the Base Case

To prove the *base case* is true for a value b we just substitute in the value we've chosen into both the left-hand side and right-hand side and show that they equate to the same thing. By doing this we prove $P(b)$ and as a result the *base case*.

2.1.2 Proving the Inductive Step

Here we assume that $P(j)$ is true for some arbitrary value $j \geq b$, and it doing so we form our *inductive hypothesis*. Then, using the hypothesis we just formed, we try to **prove** that $P(j+1)$ is true. Often we split $P(j+1)$ into $P(j)$ and the value of the expression for $n = j+1$, which we then rearrange to achieve an expression equivalent to $P(j+1)$ — this works as proof because we've assumed that $P(j)$ is true.

2.1.3 Concluding the Property Holds

If we have successfully proved that the property $P(n)$ holds for both the base case b and for the inductive step, we can conclude that as the property $P(j+1)$ holds if it also holds for $P(j)$ and because we have shown $P(b)$ to be true then the property is true for all $n \geq b$ by induction.

Chapter 3

Predicate Logic

Predicate logic.

Chapter 4

Boolean Algebra

Boolean algebra.

Glossary

alphabet A fixed character set that can be used. Sometimes referred to as Σ. 2, 7

atomic formula A formula consisting of a single proposition. 1, 7

Backus-Naur form A formal way of defining syntax, try googling it sometime. 1, 7

base case The case for a property where the value of n is the lowest possible value for which the property holds. 4, 5, 7

binary connective Also known as a binary operator in CS, a way of linking two propositions. 1, 7

inductive hypothesis The inductive hypothesis is part of the inductive step in a proof by induction where we make the assumption that a property is true for an arbitrary value. 4, 5, 7

inductive step A step of proof by induction where a property is assumed true for some arbitrary value and the assumption is then used to prove that it is also true for the successor of that value. 4, 7

proposition An assertion or statement in propositional logic. 1, 2, 7

semantics The meaning of a formula. 7

truth function A function accepts a formula and returns if it is true or not, depending on the valuation. 3, 7

valuation A function which maps propositions to true or false. 2, 7