COMP102P

# Language and Logic

Taught by Robin Hirsch

Notes compiled by Matt Bell and others

# 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.

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

### 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 ::=` $\wedge| \vee | \to | \leftrightarrow$
`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.*

$(r \to \neg g)$

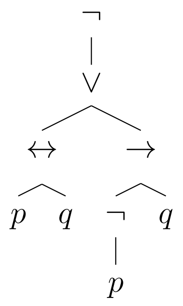
It's pretty easy to be honest, just remember to define your *propositions*.

### 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:

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

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 $\phi$ of all things. Let $\phi = \neg((p \leftrightarrow q) \vee (\neg p \to q))$. This is what its parse tree would look like:



# Chapter 2

# Induction

All about 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 $\Sigma$. 5

**atomic formula** A formula consisting of a single proposition. 1, 5

**Backus-Naur form** A formal way of defining syntax, try googling it sometime. 1, 5

**binary connective** Also known as a binary operator in CS, a way of linking two propositions. 1, 5

**proposition** An assertion or statement in propositional logic. 1, 2, 5

**semantics** The meaning of a formula. 5