CS245: Logic and Computation
Fall 2018

Introduction

These notes were created to help me learn from my mistakes and note down everything that I missed the first time I took this course. I hope they can possibly be useful to whomever I give them to.

Lecture 1

What is Logic?

This course is going to be split up into two main categories

1. First-order (and predicate) logic
2. the applications of said logic.

Logic is the **science of reasoning**.

- Etymology: *Logykos* (Greek) - pertaining to **reasoning**
- Logic = The systematic study of **the form of arguments**

Why study logic? There are many applications of logic in computer science, such as digital circuits, databases, and formal verifications.

Introduction to Propositional Logic

Ok, with those boring bullshit introductions out of the way, lets actually talk about **Propositional Logic**. A proposition is basically a declarative statement that is either *true* or *false*. And they can never be both *true* and *false*. For example, "*Pigs can fly*" is a *false* proposition.

Many sentences in English are actually not propositional. Commands, questions, and non-sensical sentences are not propositions. For example, the statements *is there a test tomorrow?* and *do it now* are not propositions.

Sentences that we don't know if they are *true* or *false* to are still propositions, such as the twin prime conjecture. Here are some more examples of propositions:

- The sum of 3 and 5 is 8
- The sum of 3 and 5 is 35
- $-1 \geq 5$
- Program p terminates, on any input (p is a known program.)
- Every even number greater than 2 is the sum of two prime numbers.

And here are some examples of statements that are not propositions

- Question: Where shall we go to eat?
- Command: Please pass the salt.
- Sentence Fragments: The dogs in the park
- Non-sensical: Green ideas sleep furiously
- Paradox: This sentence is false

Logical Connectives

Connective	Symbol	Meaning
Negation	$(\neg p)$	not p
Conjunction	$(p \land q)$	p and q
Disjunction	$(p \lor q)$	p or q
Conditional	$(p \to q)$	p implies q
Biconditional	$(p \leftrightarrow q)$	p if and only if q

Yo ok so this shit is a basically a table of most of the connectives that are used in Propositional Logic. In this course, we will not be using $\Rightarrow$ or $\Leftrightarrow$, but instead for logic we will use $\to$ and $\leftrightarrow$. These symbols have the same meaning as in Discrete Mathematics, so I will not explain Propositional formulas or the meaning of these connectives further

Usually when translating English sentences to logical formulas, there are some key phrases and expressions to keep in mind.

Connective	Alternative Expressions
$(\neg p)$	p does not hold; p is false; it is not the case that p
$(p \land q)$	p but q; not only p but q; p yet q
$(p \lor q)$	p unless q; p and/or q
$(p \to q)$	p only if q; p implies q; q if p
$(p \leftrightarrow q)$	p is equivalent to q

Let's translate some sentences into propositions. The process is usually to take one statement at a time and assign it to a propositional variable. Then you use the above table to figure out the appropriate connectives.

- $\forall \epsilon > 0, \exists \delta > 0, \text{s.t } ||x - a|| < \delta \implies ||f(x) - f(a)|| < \epsilon$

  Ok so for this one, we can (as we've seen in MATH 135), we can turn this into a hypothesis and conclusion. Let's label our variables h for hypothesis, and c for conclusion. btw this was just a reminder of what we did in 135, its not actually a translation.

$$h : ||x - a|| < \delta$$
$$c : ||f(x) - f(a)|| < \epsilon$$
$$(h \to c)$$

- It is Sunday but I don't want to sleep in.
  We need to be a bit careful when it comes to these kinds of propositions because of the word **not**. When translating the sentence to a propositional variable we must write it without "not". Why? Well there's probably a reason but my main reason is that the TAs and instructors will take off points. Instead it's "better" to avoid using **not** and use negation in your proposition.

$$s := \text{It is Sunday.}$$
$$t := \text{I want to sleep in}$$
$$(s \wedge (\neg\, t))$$

- I will eat ass or food but not both.
  Ah, the classic Exclusive-Or (XOR) question. Basically what we need to do is realize that you can have 1 of them, or the other, but not both, and according to that table "but" (haha I said but) means use $\wedge$, so our proposition ends up taking the form

$$a := \text{I will eat ass}$$
$$f := \text{I will eat food}$$
$$((a \vee f) \wedge ((\neg a) \wedge (\neg f)))$$

- Pigs can fly and the grass is red or the sky is blue
  English is ambiguous sometimes, and this is one of those times. There are many possible interpretations of this. if we let $p :=$ Pigs can fly, $g :=$ grass is red, $s :=$ sky is blue, then here are some ways to translate.

$$((p \wedge g) \vee s)$$
$$(p \wedge (g \vee s))$$

  I mean can you really tell which one is the correct way? cause I can't. So some questions will be ambiguous and you gotta watch out for that my dude.

Lecture 2

Yeet. Ok we're gonna learn some cool stuff this time around. Let's talk about the syntax and semantics of propositional formulas, as well as their properties.

Firstly, we need to realize what actually makes up a formula. Let's use a formal definition and act like this class is hard for a second!

Let $\mathcal{P}$ be a set of propositional variables. We define the set of **well-formed formulas** over $\mathcal{P}$ inductively as follows:

1. A single symbol (or atom) of $\mathcal{P}$ is well-formed
2. If α is well-formed, then $(\neg\alpha)$ is well-formed.
3. If α and β are well-formed, then $(\alpha \star \beta)$ is also well-formed, where $\star$ is one of $\{\wedge, \vee, \rightarrow, \leftrightarrow\}$
4. Nothing else is a well-formed formula

Ok so this is kinda epic. We have a definition for the thing we've been implicitly using up to this point, but this does bring up a good point. You need to absolutely use a parenthesis before a negation, so something like $\neg\neg\alpha$ is incorrect, we need to write it as $(\neg(\neg\alpha))$. Something else to keep in mind is that this definition is inductive but we don't have any sort of way to prove this using our current induction methods, so stay tuned for that.

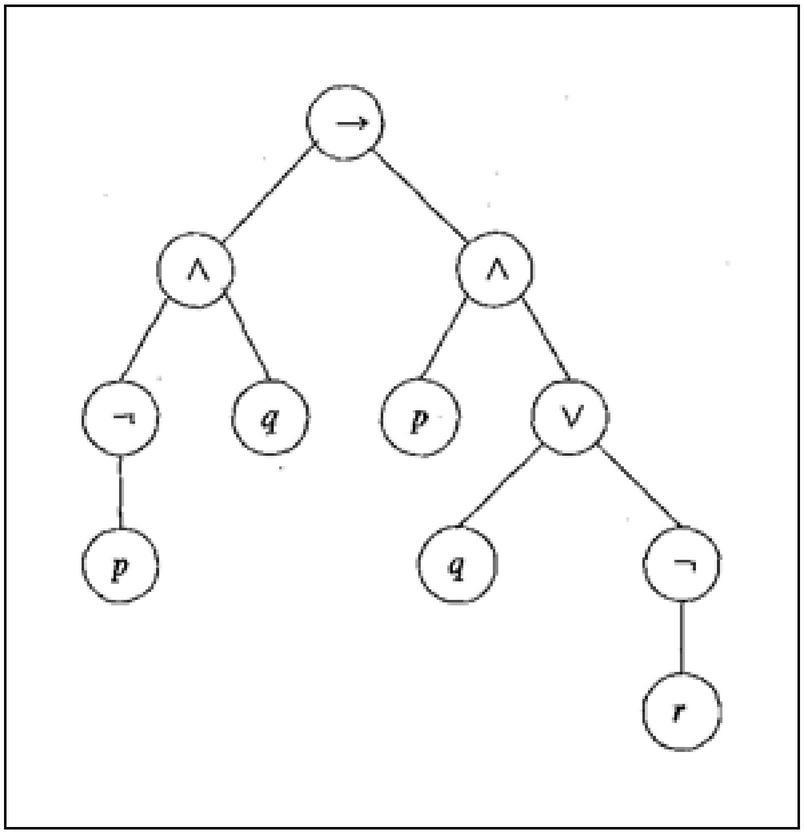
Let's talk about the Parse Tree. So this tree is most easily described as this recursive process:

1. If we have an atom, it will be a leaf and the tree ends there
2. If we have $(\neg\alpha)$ then $\neg$ will become a parent node (and have one child) to the parse tree made by α
3. If we have $(\alpha\star\beta)$ then $\star$ will become the parent node to both the parse trees created by α and β. ($\star$ is a binary connective btw)
4. Nothing else forms a parse tree

There is a clear correspondence between a parse tree and the definition of the well formed formula itself. In fact (and you can convince yourself of this, proving this is not too bad, but writing all that shit in latex for a super observable fact is kinda not worth) there exists a parse tree for every well-formed formula. So let me get super formal again and uhh

Let $f : \mathcal{P} \to$ set of all parse trees. Prove f is a bijective function.... would be a fun exercise to do on your own once you learn structural induction right? Yeah well this proof is left as an exercise to the reader... which is probably only going to be me. Damn.

Anyway, so what does a parse tree look like? Well here is an example.



Hold on though, what actually is this formula? A strategy that helps me is usually starting with a blank formula. Counting the number of branches from the root tells you what type of connective you need. So from the root of this image there are two branches and we have an implication so let just use $(thing \to stuff)$ as a starting point. Figuring out *thing* means we go down the left branch and repeat the same process. so let's do it. If we go left we get another binary connective but its $\wedge$ so $thing = (other \wedge q)$. Well let's find out what *other* is. We can stop on the right for *thing* side since q is an atom and a leaf. *other* seems to be a negation, and its subformula is a propositional variable, which means we are done. so $(thing \to stuff) = ((other \wedge q) \to stuff) = (((\neg p) \wedge q) \to stuff)$.

I'm gonna leave finding out $stuff$ to the reader, but if you manage to get the tree to be equivalent to the formula $(((\neg p) \wedge q) \to (p \wedge (q \vee (\neg r))))$, I think you're probably right. Well enough about parse trees for now, let's talk about a theorem.

Theorem. Every WFF (Well-Formed Formula) has a unique derivation as a well-formed formula. That is each well formed formula can be written in only one way.

Sidenote, I don't really understand why this definition isn't covered after we learn about the thing to prove it but whatever man. It turns out that this theorem is true, and it kind of feels intuitive that it would be true.

Structural induction

Probably the most important way to prove anything in Propositional Logic would be Structural Induction. At a higher level this proof method is used to show inductive structures are well defined, or a certain property of an inductive structure hold. Let's do an example first where we can use both regular mathematical induction and structural induction.

Show that every well-formed formula has an equal number of closing braces and opening braces.

Proof. Let us use mathematical induction on the height of the parse tree. Let $P(n)$ be the statement that every WFF with height n has an equal number of opening and closing braces.

<u>Base Case</u> The minimal parse tree by definition would be the tree of height $n = 1$, with an atom as its root. The WFF associated with this has no parentheses, so there are 0 opening and 0 closing parentheses so $P(1)$ holds.

<u>Inductive Hypothesis</u> Let $P(k)$ hold for some integer $k \in \mathbb{N}$ and also let $P(i)$ hold for all integers $1 \leq i \leq k$.

<u>Inductive Conclusion</u> We need to show that $P(k+1)$ is also true
Exercise left to the writer. fuck this tbh

$\square$

If we were to use structural induction for this proof instead of this weird induction on the parse tree, then