COMP21111 Revision Notes

January 5, 2015

Contents

1 Introduction

Just a short introduction to say that I've written these notes to the best of my understanding of the syllabus, I hope they are right, but if you spot any errors, please tell me (you can email me at Isona7@hotmail.co.uk or message me on Facebook where I am Izzy Whistlecroft)!
These are notes for Logic & Modelling, the exam for 2014 is on the 23rd of January, at 2PM, and is worth 80% of the marks. Good luck!
PS: I've not written any notes for Lecture 1, as it seems to be mostly fluff, the "good stuff" starts at Lecture 2.

2 Lecture 2

2.1 Propositional Logic Syntax

Every variable is a *formula*, also known as an *Atomic Formula* or an *Atom*.
$\top$ and $\bot$ are formulas.
If A is a formula, then $\neg A$ is a formula.
$\top$, $\bot$, $\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$ are *connectives*.

2.2 Subformulas

Formulas $A_1, ..., A_n$ are immediate subformulas of $(A_1 \wedge ... \wedge A_n)$ and $(A_1 \vee ... \vee A_n)$.
Formula A is an immediate subformula of $\neg$A.
Formulas A and B are immediate subformulas of (A $\rightarrow$ B) and (A $\leftrightarrow$ B).
Every formulas is a subformula of itself.
If A is a subformula of B and B is a subformula of C, A is a subformula of C.

2.3 Connective Table

Connective	Name	Precedence
$\top$	verum	
$\bot$	falsum	
$\neg$	negation	5
$\wedge$	conjunction	4
$\vee$	disjunction	3
$\rightarrow$	implication	2
$\leftrightarrow$	equivalence	1

2.4 Parsing

Parsing $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$:

Inside-Out (starting with the highest precedence connectives):

$(\neg A) \wedge B \rightarrow C \vee D \leftrightarrow E$
$((\neg A) \wedge B) \rightarrow C \vee D \leftrightarrow E$
$((\neg A) \wedge B) \rightarrow (C \vee D) \leftrightarrow E$
$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E$

Outside-In (starting with lowest precedence connectives):
$(\neg A \wedge B \rightarrow C \vee D) \leftrightarrow E$
$((\neg A \wedge B) \rightarrow (C \vee D)) \leftrightarrow E$
$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E$

It doesn't really matter what order you insert the brackets!

2.5 Operation Tables

$\wedge$	1	0
1	1	0
0	0	0

$\vee$	1	0
1	1	1
0	1	0

$\neg$	
1	0
0	1

$\rightarrow$	1	0
1	1	0
0	1	1

$\leftrightarrow$	1	0
1	1	0
0	0	1

2.6 Interpretations and Models

We can assign values to boolean variables. An interpretation is a set of such assignments, also known as truth assignments.
If I(A) = 1, we say that that I is a model of A.
If I(A) = 0, then we say that A is false in I.
A is satisfiable if some interpretation satisfies it.
A is valid (a tautology) if it is true in every interpretation.
A is invalid if it isn't satisfied by any interpretation.
Two formulas are equivalent if they have the same models.

2.7 Evaluating a Formula

Evaluating the formula:
$(p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r)$
$\{p \mapsto 1, q \mapsto 0, r \mapsto 1\}$

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

The formula is true in this interpretation.

3 Lecture 3

3.1 Rewrite Rules

$\top \wedge ... \wedge \top \Rightarrow \top$
$\bot \wedge A_1 \wedge ... \wedge A_n \Rightarrow \bot$

$\top \vee A_1 \vee ... \vee A_n \Rightarrow \top$
$\bot \vee ... \vee \bot \Rightarrow \bot$

$\neg\top \Rightarrow \bot$
$\neg\bot \Rightarrow \top$

$A \rightarrow \top \Rightarrow \top$
$\bot \rightarrow A \Rightarrow \top$
$\top \rightarrow \bot \Rightarrow \bot$

$\top \leftrightarrow \top \Rightarrow \top$
$\top \leftrightarrow \bot \Rightarrow \bot$
$\bot \leftrightarrow \top \Rightarrow \bot$
$\bot \leftrightarrow \bot \Rightarrow \top$

3.2 Evaluating a formula

The formula
$(p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r)$
with the evaluation $p \mapsto 1, q \mapsto 0, r \mapsto 1$
is equivalent to (having replaced p with $\top$, q with $\bot$ and r with $\top$):
$(\top \rightarrow \bot) \wedge (\top \wedge \bot \rightarrow \top) \rightarrow (\top \rightarrow \top)$

Applying rewrite rules:
$(\top \rightarrow \bot) \wedge (\top \wedge \bot \rightarrow \top) \rightarrow (\top \rightarrow \top)$
$\bot \wedge (\top \wedge \bot \rightarrow \top) \rightarrow (\top \rightarrow \top)$
$\bot \wedge (\bot \rightarrow \top) \rightarrow (\top \rightarrow \top)$
$\bot \wedge \top \rightarrow (\top \rightarrow \top)$
$\bot \rightarrow (\top \rightarrow \top)$
$\bot \rightarrow \top$
$\top$

The result will be the same, whatever order you rewrite in!

4 Lecture 4

4.1 Compact Truth Table

formula	I1	I2	I3	I4
$(p \to q) \land (p \land q \to r) \to (p \to r)$	1	1	1	1
$p \to r$	1	1	0	0
$(p \to q) \land (p \land q \to r)$	1		0	0
$p \land q \to r$	1	1	1	0
$p \to q$	1		0	1
$p \land q$	0		0	1
p p p	0	1	1	1
q q			0	1
r r		1	0	0

This formula is valid!
The order of variables affects the size but NOT the result!

4.2 Simplification Rules for $\top$

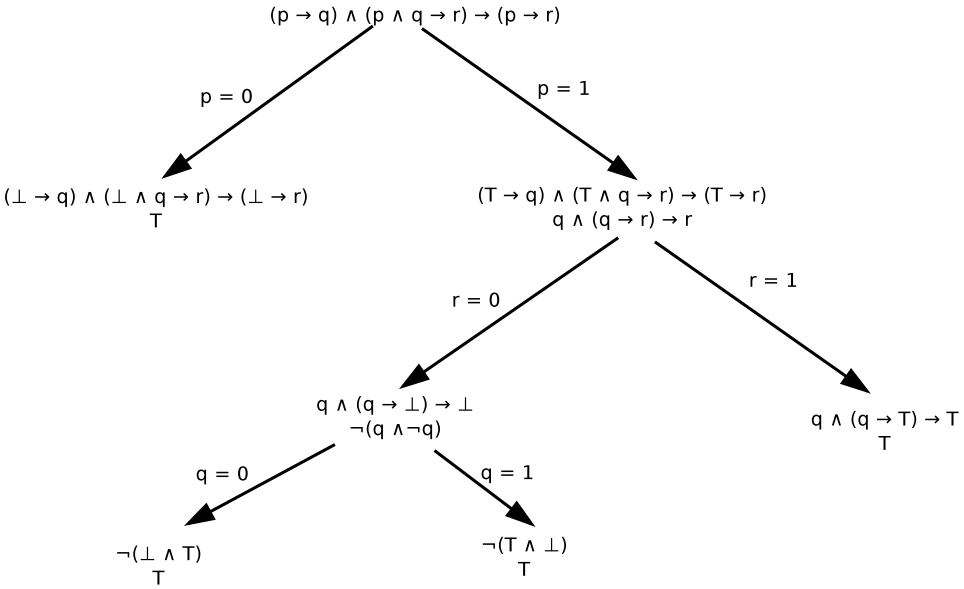
$\neg\top \Rightarrow \bot$
$\top \land A_1 \land ... \land A_n \Rightarrow A_1 \land ... \land A_n$
$\top \lor A_1 \lor ... \lor A_n \Rightarrow \top$
$A \to \top \Rightarrow \top$
$\top \to A \Rightarrow A$
$A \leftrightarrow \top \Rightarrow A$

4.3 Simplification Rules for $\bot$

$\neg\bot \Rightarrow \top$
$\bot \land A_1 \land ... \land A_n \Rightarrow \bot$
$\bot \lor A_1 \lor ... \lor A_n \Rightarrow A_1 \lor ... \lor A_n$
$A \to \bot \Rightarrow \neg A$
$\bot \to A \Rightarrow \top$
$A \leftrightarrow \bot \Rightarrow \neg A$
$\bot \leftrightarrow A \Rightarrow \neg A$

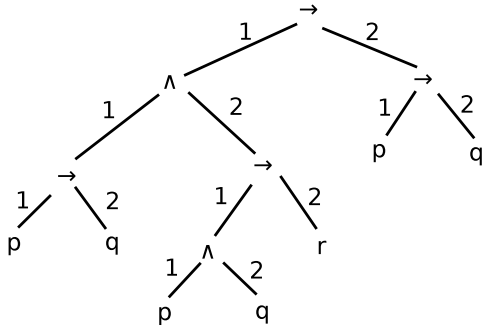
4.4 Splitting Example

The idea of splitting is that at each step, you make a branch on a variable, and replace it with $\top$ or $\bot$ (depending on what the variable is on that branch). Then, apply the simplification rules until it cannot be simplified any more. Continue doing this until all branches end in $\top$ or $\bot$.
An example of splitting on the formula $(p \to q) \land (p \land q \to r) \to (p \to r)$:



4.5 Parse Trees

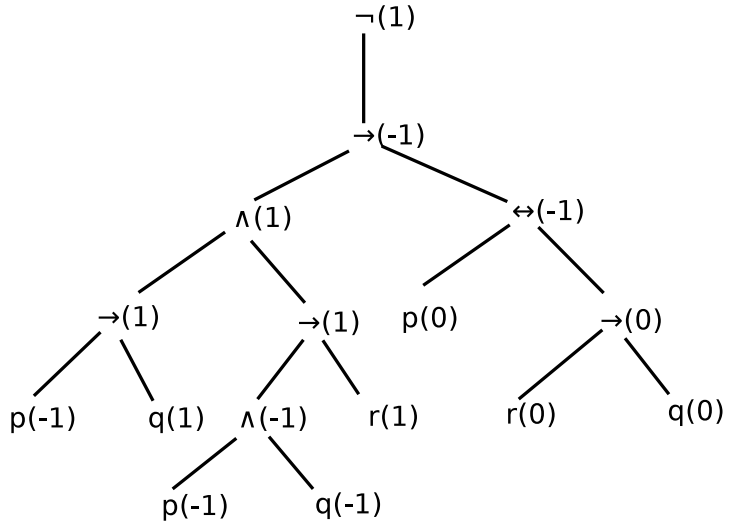
An example of a parse tree for the formula $(p \to q) \wedge (p \wedge q \to r) \to (p \to r)$:



Positions are given by a series of non-negative numbers separated by periods.
A few examples of locations:
The subformula at position 1.2.1 is: $p \wedge q$
The subformula at position 1 is: $(p \to q) \wedge (p \wedge q \to r)$
The subformula at position 1.1.2 is: q

4.6 Polarity

An example showing the polarity of the formula:
$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \leftrightarrow (r \to q)))$



The polarity of the top node in a parse tree is 1.
The polarity of the node below a $\neg$ is reversed (1 becomes -1, -1 becomes 1).
All nodes below a $\leftrightarrow$ have polarity 0.

4.7 Pure Variables

If all occurences of a variable in a formula are positive, then the formula is only satisfiable if that variable is true. Likewise, if all occurences are negative, then the formula is only satisfiable if that variable is false.
This means, when checking for satisfiability, pure variables will be replaced with either $\top$ (when polarity is always positive) or $\bot$ (when polarity is always negative).
In the formula $\neg((p \to q) \wedge (p \wedge q \to r) \to (\neg p \to r))$, p is always negative (trust me on this!).
Due to P being always negative, we can replace p by $\bot$:
$\neg((\bot \to q) \wedge (\bot \wedge q \to r) \to (\top \to r))$
$\neg(\top \wedge (\bot \wedge q \to r) \to (\top \to r))$
$\neg(\top \wedge (\bot \to r) \to (\top \to r))$
$\neg(\top \wedge \top \to (\top \to r))$
$\neg(\top \to r)$
$\neg r$
Now r is pure (always negative) so we can replace it by $\bot$.
$\neg\bot$
$\top$

5 Lecture 5

5.1 Literals and Clauses

A literal is an atom or its negation eg p and $\neg p$.
A clause is a disjunction of literals eg $p \vee q \vee \neg r \vee s$.
The empty clause is denoted by $\Box$ and is always false and occurs when there are no literals.
A horn clause is a clause with at most one positive literal.

5.2 CNF

A formula is in conjunctive normal form when it is "either $\top$ or $\bot$ or a conjunction of disjunctions or literals".
This means that it will be a formula made only of literals and brackets containing only literals separated by $\vee$, connected by $\wedge$.
Examples:
Correct CNF:
$p \wedge \neg s \wedge t \wedge (u \vee \neg w \vee x)$

Not in CNF (there is a $\neg$ outside brackets:
$\neg(p \vee \neg s \vee t) \wedge u \wedge w$

Also not in CNF(Bracket in bracket and $\rightarrow$ is not allowed!):
$(p \vee (\neg s \rightarrow t)) \wedge u \wedge w$

Also not CNF (there are $\vee$ outside of brackets):
$p \vee \neg s \wedge u \vee w$
The last one in CNF (note the added brackets):
$(p \vee \neg s) \wedge (u \vee w)$

5.3 CNF Transformation Rules

$A \leftrightarrow B \Rightarrow (\neg A \vee B) \wedge (\neg B \vee A)$
$A \rightarrow B \Rightarrow \neg A \vee B$
$\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$ (De Morgan's Law)
$\neg(A \vee B) \Rightarrow \neg A \wedge \neg B$ (Also De Morgan's Law)
$\neg\neg A \Rightarrow A$
$(A_1 \wedge ... \wedge A_m) \vee B_1 \vee ... \vee B_n \Rightarrow (A_1 \vee B_1 \vee ... \vee B_n) \wedge ... \wedge (A_m \vee B_1 \vee ... \vee B_n)$

5.4 Problems with CNF

The problem with CNF, is that it may be exponential in size, particularly when $\rightarrow$ and $\leftrightarrow$ are involved.
The solution to this is to introduce definitions/naming.
This involves taking a subformula A, introducing a new name, n for it and then making a statement to say that they are equivalent.

For example:
$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p6))))$
$n \leftrightarrow (p_5 \leftrightarrow p_6)$
Here, a new name n has been introduced for the subformula $p_5 \leftrightarrow p_6$, we can now replace all occurences of this subformula with n (while still including the definition of n):
$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$
$n \leftrightarrow (p_5 \leftrightarrow p_6)$

6 Lecture 6

6.1 Clausal Form

A clausal form for a formula is a set of clauses which have the same models as the original formula. Clausal form is largely produced by using the naming strategy mentioned in the last section.
The benefit of using Clausal Form over CNF is that clausal form can be produced extremely quickly, while CNF is prone to becoming exponential in size.

6.2 Clausal Form Transformation

This algorithm can produce a clausal form of a given formula. The definition of a clausal form is given that as:
If the formula A has the form: $C_1 \wedge ... \wedge C_n$ where each C_i is a clause and there is more than one clause, the clausal form S of A, has the form $S = \{C_1, ..., C_n\}$.
As with most examples, I have taken this from the slides, but will explain it in more detail below.

	subformula	definition	clauses
			n_1
n_1	$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$	$n_1 \leftrightarrow \neg n_2$	$\neg n_1 \vee \neg n_2$ $n_1 \vee n_2$
n_2	$(p \to q) \wedge (p \wedge q \to r) \to (p \to r)$	$n_2 \leftrightarrow (n_3 \to n_7)$	$\neg n_2 \vee \neg n_3 \vee n_7$ $n_3 \vee n_2$ $\neg n_7 \vee n_2$
n_3	$(p \to q) \wedge (p \wedge q \to r)$	$n_3 \leftrightarrow (n_4 \wedge n_5)$	$\neg n_3 \vee n_4$ $\neg n_3 \vee n_5$ $\neg n_4 \vee \neg n_5 \vee n_3$
n_4	$p \to q$	$n_4 \leftrightarrow (p \to q)$	$\neg n_4 \vee \neg p \vee q$ $p \vee n_4$ $\neg q \vee n_4$
n_5	$p \wedge q \to r$	$n_5 \leftrightarrow (n_6 \to r)$	$\neg n_5 \vee \neg n_6 \vee r$ $n_6 \vee n_5$ $\neg r \vee n_5$
n_6	$p \wedge q$	$n_6 \leftrightarrow (p \wedge q)$	$\neg n_6 \vee p$ $\neg n_6 \vee q$ $\neg p \vee \neg q \vee n_6$
n_7	$p \to r$	$n_7 \leftrightarrow (p \to r)$	$\neg n_7 \vee \neg p \vee r$ $p \vee n_7$ $\neg r \vee n_7$

To create the tabular form of this particular formula, we first start by labelling each of the rows as we split the formula (the far left column).
The subformula in each respective row is what we will assign to that row's variable. As you can see in row n_7, where we assign it to $p \to r$. The only thing about this is that any subformulas mentioned in rows below will be replaced with their new name. This means that you can create these tables going either upwards or downwards depending on how you like to think!
To work out the clauses for each row of the table, we will apply the cnf transformation rules to each row of the table.
As an example of working out the clauses in the table we will take row $n_6 \leftrightarrow (p \wedge q)$.
$n_6 \leftrightarrow (p \wedge q)$ (Apply rule for equivalence)
$(\neg n_6 \vee (p \wedge q)) \wedge (n_6 \vee \neg(p \wedge q))$ (De Morgan's Law on right side)
$(\neg n_6 \vee (p \wedge q)) \wedge (n_6 \vee (\neg p \vee \neg q))$
$((\neg n_6 \vee p) \wedge (\neg n_6 \vee q)) \wedge (n_6 \vee \neg p \vee \neg q)$ (Remove brackets)
$(\neg n_6 \vee p) \wedge (\neg n_6 \vee q) \wedge (n_6 \vee \neg p \vee \neg q)$
This leaves us with the clauses, which are separated by $\wedge$.

6.3 Optimised Clausal Form Transformation

There is an optimised version of clausal form transformation that the slides gloss over quite quickly. This produces fewer clauses than the unoptimised

version, but requires the polarity of the subformula to be known (polarity works the same way as earlier in the notes).
The table from before, but with the polarity shown, and superfluous clauses removed:

	subformula	polarity	definition	clauses
				n_1
n_1	$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$	$+1$	$n_1 \to \neg n_2$	$\neg n_1 \vee \neg n_2$
n_2	$(p \to q) \wedge (p \wedge q \to r) \to (p \to r)$	-1	$(n_3 \to n_7) \to n_2$	$n_3 \vee n_2$ $\neg n_7 \vee n_2$
n_3	$(p \to q) \wedge (p \wedge q \to r)$	$+1$	$n_3 \to (n_4 \wedge n_5)$	$\neg n_3 \vee n_4$ $\neg n_3 \vee n_5$
n_4	$p \to q$	$+1$	$n_4 \to (p \to q)$	$\neg n_4 \vee \neg p \vee q$
n_5	$p \wedge q \to r$	$+1$	$n_5 \to (n_6 \to r)$	$\neg n_5 \vee \neg n_6 \vee r$
n_6	$p \wedge q$	$+1$	$(p \wedge q) \to n_6$	$\neg p \vee \neg q \vee n_6$
n_7	$p \to r$	-1	$(p \to r) \to n_7$	$p \vee n_7$ $\neg r \vee n_7$

If you are particularly eagle-eyed and noticed the polarity of n_6 here differs from in the slides, it is because the slides are incorrect, the chapter on this agrees with me.
So, instead of all the time writing $n_i \leftrightarrow A$:
If the polarity of A is 1, write $n_i \to A$
If the polarity of A is -1, write $A \to n_i$
If the polarity of A is 0, then write $A \leftrightarrow n_i$ (same as before...)
Everything else in the table works as before, but converting $\to$ to cnf produces fewer clauses than $\leftrightarrow$.

6.4 Unit Propagation

In my opinion, the slides for this particular topic are great, as they animate as you scroll through them showing the propagation, if you want to look at them, they are the slides for lecture 6 (the good example starting at slide 17).
In unit propagation, we look for any clause only containing a literal (including the negations of literals), as we know that for the set of clauses to be satisfied, we must make that clause true.
We can then delete any clauses including that literal.
We can also delete the negated version of that literal from any clauses containing it (note: not the clauses, only the negated literal).
Shortish Example:
n_1
$\neg n_1 \vee \neg n_2$
$n_1 \vee n_2 \vee n_3$
$n_2 \vee n_3$
$n_2 \vee n_3 \vee \neg n_4$

Since n_1 appears on its own, we can propagate it, removing one clause outright, and deleted $\neg n_1$ from another:
$\neg n_2$
$n_2 \vee n_3$
$n_2 \vee n_3 \vee \neg n_4$
Now we are left with a clause containing only $\neg n_2$, there are no clauses containing $\neg n_2$ itself, but we can delete n_2 from 2 clauses:
n_3
$n_3 \vee \neg n_4$
Now we can propagate n_3, deleting the 2 clauses containing it.
This leaves us with no remaining clauses, and the empty *set* remaining.
If we were left with clauses that looked something like:
n_4
$\neg n_4$
we would propagate on one, delete a literal from the other, and be left with the empty *clause*.
The difference between the empty set and clause is important here! An empty clause is still a clause, and must be satisfied to satisfy the set of clauses, but it cannot be satisfied! The empty set is empty of clauses and hence satisfied!
Takeaway message: Empty *clause* : unsatisfied. Empty *set*: satisfied.

6.5 DPLL