CO519 - Theory of Computing - Logic

Dominic Orchard
School of Computing, University of Kent

Last updated on September 14, 2020 (print version)

These are the accompanying notes for the logic part of CO519. They provide a counterpart to the lectures, but they do not replace them; **the lectures will provide content, examples, detail, and discussion not given in these notes**. However, the notes also provide additional exercises and examples, and some additional detail beyond what is assessed on this course—this will be pointed out when it happens.

These notes also act as a counterpart to the **course textbook**, *Logic in Computer Science* (by Huth and Ryan) which I recommend. We will only cover some of the material from the first two chapters due to the short length of this part of the course. I recommend reading these chapters (they have further exercises and examples), but the assessable material for this course is covered in lectures and these notes (excluding the appendices); the textbook contains extra detail which is not assessed, but worth learning.

If you spot any errors or have suggested edits, the notes are written in LaTeX and are available on GitHub at `http://github.com/dorchard/co519-logic`. Please fork and submit a pull request with any suggested changes.

The **last page of these notes** provides a section for you to write down any topics you do not yet understand, or feel that you need to work on more. By writing these down during the course, you offload the task of remembering what you need to work on. You can then look back at this list when you are studying, and eventually cross off items as you get to grips with the material.

Part A
Propositional logic & its natural deduction proofs

Truth tables are a handy way to give meaning to logical operators and formulas, in terms of truth or falsehood. But they are difficult to use for reasoning about anything but small formulas since the number of rows is 2^n where n is the number of contingent formulas. For example, the truth table for $P \vee (Q \wedge R) \rightarrow (P \vee Q) \wedge (P \vee R)$ has $2^3 = 8$ rows as there are three contingent formulas: P, Q, and R whose truth or falsehood determines the truth or falsehood of the overall formula. Enumerating all possibilities takes time even for this modest truth table, and is error prone (humans are bad at repetitive detailed tasks).

Instead, logicians have constructed formal languages (calculi) for building complex chains of reasoning (proofs) in a more compact form. In this course, we use the *natural deduction* calculus due to 20^{th} century logicians such as Gerhard Gentzen (who formulated natural deduction in 1934) and Dag Pravitz (who promoted this style in the 1960s and converted various other results of Gentzen into the natural deduction style).

1 Natural deduction for propositional logic

Natural deduction provides a system of *inference rules* which explain how to construct and deconstruct formulas to build a logical argument. These rules represent the derivation of a formula (the *conclusion*) from several other formulas that are assumed or known to be true (the *premises*) via the notation:

$$\frac{premise_1 \ldots \quad \ldots premise_n}{conclusion}\ (label)$$

Each logical operator (like conjunction $\wedge$ ("and") and disjunction $\vee$ ("or")), will have one or more rules for *introducing* that operator (deriving a conclusion formula which uses that operator) and one or more rules for *eliminating* that operator (deriving a conclusion from a premise formula that uses that operator). These rules can be stacked together to form a logical argument: *a proof*, which looks like the following:

$$\dfrac{\dfrac{P_1 \quad P_2}{P_3} \quad P_4}{P_5}$$

This isn't an actual concrete proof yet, it's just an example of how a natural deduction proof looks when you stack together its inference rules. Informally, we might have something like the following:

$$\dfrac{\dfrac{\textit{I forgot my coat} \quad \textit{It's raining}}{\textit{I get wet}} \quad \textit{My hairdryer broke}}{\textit{My hair remains wet}}$$

But this isn't always the most helpful format to derive the proofs. Instead, we'll use a special "box"-like notation called Fitch-style which you can find in the recommend textbook *Logic in Computer Science* by Huth and Ryan. We will still apply the rules of natural deduction, but the Fitch-style gives us a nice way to layout the proof as we are deriving it.

We will step through the natural deduction rules for the core logical operators: $\wedge$ (conjunction/and), $\vee$ (disjunction/or), $\rightarrow$ (implication/if-then), $\neg$ (negation), as well as truth $\top$ and falsehood/falsity which is often written in propositional logic as $\bot$ (pronounced "bottom").[1] Note that, in the literature, logical operators are sometimes alternatively called *logical connectives*.

1.1 Properties of formulas

We will consider three properties that a logical formula may have:

- *valid*: a formula which is always true (also called a *tautology*). We will mostly prove the validity of formulas in *Part A*. For example, $P \wedge Q \rightarrow Q \wedge P$ is true no matter the truth/falsehood of P and Q.
- *satisfiable*: a formula which is true for *some* assignments of truth/falsehood to the atoms/variables it contains, e.g., $a \wedge b$ is satisfiable, and the *satisfying assignment* is that $a \mapsto \top$ and $b \mapsto \top$ (both must be true). A valid formula is trivially satisfiable (true for any and all assignments)

  Satisfiability, and an algorithm for finding a satisfying assignment, will be covered in *Part C*.
- *unsatisfiable*: a formula which is always false (all rows in the truth table are false) regardless of assignments to the variables/atom, e.g. $P \wedge \neg(P \vee Q)$. We can prove a formula is unsatisfiable by proving its negation is valid.

1.1.1 Entailment and sequents

Suppose we have a set of formulas $P_1, \ldots, P_n$ from which we want to prove Q by applying the rules of a particular logic (propositional logic here). The formulas $P_1, \ldots, P_n$ are the premises and Q is our goal conclusion. This is often written using the following notation called a *sequent*:

$$P_1, \ldots, P_n \vdash Q$$

[1] *Bottom* $\bot$ is often used in maths and computer science to represent undefined values or behaviour. In logic, if have arrived at falsity $\bot$ during a proof then we are in a situation where anything could be true as we've arrived at a logically inconsistent situation. This is sometimes quite useful for doing proofs-by-contradiction, as we will see in Section 1.5.

The turnstile symbol $\vdash$ is read as *entails* and the premises to the left are sometimes call the *context* of assumed formulas. This is a compact representation of a formula Q along with any assumptions used to deduce it.

We say a sequent is *valid* if there is a proof that derives the conclusion from the premises. For example, $P \wedge Q \vdash Q \wedge P$ is valid, meaning from $P \wedge Q$ there is a proof of $Q \wedge P$.

When there are no premises, we often drop the $\vdash$, i.e., writing something like "$(P \wedge Q) \to P$ *is valid*" instead of "$\vdash (P \wedge Q) \to P$ *is valid*".

1.2 Conjunction ("and")

Recall the truth table for conjunction:

P	Q	$P \wedge Q$
F	F	F
F	T	F
T	F	F
T	T	T

From this, we see that in order to conclude the truth of $P \wedge Q$ we need the truth of P and the truth of Q. This justifies the following natural deduction rule for *introducing* conjunction:

$$\frac{P \qquad Q}{P \wedge Q} \wedge_i$$

The label is subscripted with 'i' for introduction. That is, given two premises that P is true and Q is true, then $P \wedge Q$ is true. There is just one introduction rule, corresponding to the fact that there is only one "true" row for $P \wedge Q$ in the truth table (highlighted above).

The letters P and Q are place-holders here for *any propositional formula* so we could instantiate the rule to something like the following if we needed it:

$$\frac{(P \vee R \to Q) \quad S}{(P \vee R \to Q) \wedge S} \wedge_i$$

What about elimination? Reading the highlighted line in the truth table from right-to-left shows how to *eliminate* a conjunction, i.e., what smaller formulas can we conclude are true if we know that $P \wedge Q$ is true? We get two rules:

$$\frac{P \wedge Q}{P} \wedge_{e1} \qquad\qquad \frac{P \wedge Q}{Q} \wedge_{e2}$$

The labels have a subscript 'e' for elimination; this convention will continue. (*Aside*: If the syntax of the inference rules let us have multiple conclusions then we could collapse the two eliminations into one rule, but natural deduction instead has single conclusions. There are different proof systems which allow multiple conclusions (like the *sequent calculus*) but we won't cover that here).

Let's write a proof with just these rules by stacking them together into a *proof tree*.

Example 1. For any formula P, Q, R then $P \wedge (Q \wedge R) \vdash (P \wedge Q) \wedge R$ is valid, i.e. given $P \wedge (Q \wedge R)$ we can prove $(P \wedge Q) \wedge R$.

$$\dfrac{\dfrac{\dfrac{P \wedge (Q \wedge R)}{P}\wedge_{e1} \quad \dfrac{\dfrac{P \wedge (Q \wedge R)}{Q \wedge R}\wedge_{e2}}{Q}\wedge_{e1}}{P \wedge Q}\wedge_i \quad \dfrac{\dfrac{P \wedge (Q \wedge R)}{Q \wedge R}\wedge_{e2}}{R}\wedge_{e2}}{(P \wedge Q) \wedge R}\wedge_i$$

The root of the tree is our goal $(P \wedge Q) \wedge R$, which is built from the premises on the line above. These chains of reasoning go up to the "leaves" of the tree, which is the assumed formula $P \wedge (Q \wedge R)$. At each step (each line) we've applied either conjunction introduction or one of the conjunction elimination rules (as can be seen from the labels on the right).

The following exercise is to prove the converse of the above property.

Exercise 1.1. Prove that $(P \wedge Q) \wedge R \vdash P \wedge (Q \wedge R)$ is valid by instantiating and stacking together inference rules.

This proof, and the one above, together imply that conjunction $\wedge$ is *associative*, i.e., $P \wedge (Q \wedge R) = (P \wedge Q) \wedge R$. We come back to such equations on formulas in Section 2.

Fitch-style proof So far we have constructed proofs by stacking natural deduction inference rules on top of each other. This leads us towards a *bottom-up* proof strategy starting with the goal and working up towards the premises. In this course, we use a different approach called "Fitch-style". This style begins with assumptions, numbers each line of a proof, and uses indentation and boxes to represent sub-proofs and the scope of their assumptions.

The previous proof tree is rewritten in the following way in Fitch notation:

1.	$P \wedge (Q \wedge R)$	premise
2.	P	$\wedge_{e1}$ 1
3.	$Q \wedge R$	$\wedge_{e2}$ 1
4.	Q	$\wedge_{e1}$ 3
5.	R	$\wedge_{e2}$ 3
6.	$P \wedge Q$	$\wedge_i$ 2, 4
7.	$(P \wedge Q) \wedge R$	$\wedge_i$ 6, 5 $\quad \square$

The proof follows in a number of linear steps. On the left, we number each line of the proof. On the right, we explain which rule was applied to which formulas as premises, e.g., on the second line we have applied conjunction elimination $\wedge_{e1}$ taking line 1 as the premise, concluding with the formula P. Or for example, on line 6, conjunction introduction is applied to lines 2 and 4 (i.e., they are taken as the premises of the rule) to get $P \wedge Q$. We finish on line 7 with our goal, which is marked with $\square$ which is a way of saying the proof is finished and we've reached our goal (the symbol means $Q.E.D$ which is an abbreviation of *quod erat demonstrandum*, Latin for "what was to be demonstrated").

We haven't used any sub-proofs yet (which have a box drawn around them); these appear in the next subsection on implication.

Order of numbers in labels Note that the order of the line numbers in labels tells us the order of the premises to a natural deduction rule and so the order is important. For example, line 6 above applies ($\wedge_i$ 2, 4) to introduce $P \wedge Q$, but if it was actually ($\wedge_i$ 4, 2) we would be introducing $Q \wedge P$ which is not our intended goal.

Exercise 1.2. Rewrite your proof to Exercise 1.1 using Fitch style.

Remark. (**important**) Depending on what is being proved, a top-down approach (starting from the premises) or bottom-up approach (starting from the goal/conclusion) can be easier. In practice, if you are stuck it can help to start *at both ends* and work towards the middle. You can do this by putting the goal near a bottom of a piece of paper, giving enough space to meet in the middle. You can sort out the numbering afterwards.

It doesn't matter if things get messy—the primary goal is to reach a proof. You can rewrite it afterwards to be more clear; you should do so in your class work and assessments.

1.3 Implication

Recall the truth table for implication:

P	Q	$P \to Q$
F	F	T
F	T	T
T	F	F
T	T	T

Implication $P \to Q$ is interesting because if $\neg P$ (if P is false) then Q can be true or false, i.e., Q can be anything if P is false (the top two lines).

Exercise 1.3. Recall that $P \to Q = \neg P \vee Q$. Show this is true by comparing the truth tables for each side of this equation.

As with conjunction, we'll consider the two styles of rule: elimination and introduction. The elimination rule for implication in natural deduction is:

$$\frac{P \to Q \qquad P}{Q} \to_e$$

This rule is also known as *modus ponens*.[2] It says that if we know $P \to Q$ and we know P then we know Q. You can verify the soundness of this rule by looking at the truth table: indeed Q is true when both $P \to Q$ and P are true.

There are various other natural deduction rules one might construct by looking at the truth table– but this one can be used to derive the others. The particular set of natural deduction rules we look at was carefully honed by logicians to provide a kind of "minimal" calculus for proofs.

Introduction of an implication $P \to Q$ follows from a *subproof* (which is drawn in a box) which starts with an assumption of P and ends with Q as a conclusion after any number of steps. The rule is written as follows:

$$\frac{\boxed{\begin{array}{c} P \\ \vdots \\ Q \end{array}}}{P \to Q} \to_i$$

Thus subproofs in both tree- and Fitch-style proofs are of the form:

$$\boxed{\begin{array}{c} \textit{assumption} \\ \vdots \\ \textit{conclusion} \end{array}}$$

Remark. (**important**) When we start a subproof box, the first formula is always an assumption, which we are free to choose. When the box is closed, the assumption does not go away but becomes the premise of the implication when applying the $\to_i$ rule.

This is an important point: when proving a theorem we have to be careful not to introduce additional assumptions which are not part of the theorem. For example, let's say we are proving a theorem expressed by a formula Q but in doing so we assume P but P is not one of Q's assumptions. Then instead we will have proved $P \to Q$ rather than Q. This is something to keep in mind when writing complex proofs by hand in an informal way. The proof system of natural deduction allows us to keep track of our assumptions and their eventual inclusion in the final result.

Aside: mechanised proof assistants (software systems in which we can write machine-checked proofs, such as *Coq*, *Isabelle*, *Agda*) have a similar basis to natural deduction and give us confidence and precision in writing proofs.

There is an alternate presentation of natural deduction called *sequent-style natural deduction* (described in the appendix for this part) where the inference rules are expressed in terms of sequents $P_1, \ldots, P_n \vdash Q$. This won't be assessed on the course, but is worth looking at if you want to read more widely on logic. Another proof calculus (also due to Gentzen) is the *sequent calculus* which you can read about elsewhere.

[2] *modus ponens* is short for the Latin phrase *modus ponendo ponens* which means "the way that affirms by affirming".

Example 2. The following simple formula about conjunction and implication is valid: $\vdash (P \wedge Q) \to (Q \wedge P)$. Here is its proof in Fitch-style:

1.	$P \wedge Q$	assumption
2.	P	$\wedge_{e1}$ 1
3.	Q	$\wedge_{e2}$ 1
4.	$Q \wedge P$	$\wedge_i$ 3, 2
5.	$P \wedge Q \to Q \wedge P$	$\to_i$ 1-4 $\square$

In the last line, we apply implication introduction and we label it with the range of the lines of the subproof used (in this case 1-4).

Remark. In Example 1 we proved that given $P \wedge (Q \wedge R)$ then $(P \wedge Q) \wedge R$. We can turn this into an implication $P \wedge (Q \wedge R) \to (P \wedge Q) \wedge R$ simply by using implication introduction on the original proof. Indeed, we have a general meta-theorem of propositional logic, that if $P \vdash Q$ then $\vdash P \to Q$, and vice versa.

Example 3. The following is valid:

$$(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$$

Here is its proof:

1.	$P \to (Q \to R)$	ass.
2.	$P \to Q$	ass.
3.	P	ass.
4.	$Q \to R$	$\to_e$ 1, 3
5.	Q	$\to_e$ 2, 3
6.	R	$\to_e$ 4, 5
7.	$P \to R$	$\to_i$ 3-6
8.	$(P \to Q) \to (P \to R)$	$\to_i$ 2-7
9.	$(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$	$\to_i$ 1-8 $\square$

Here we have an example of multiple nesting of subproofs. (Tip: I proved this by working top-down and bottom-up at the same time).

Occasionally it is useful to "copy" a formula from earlier in a proof. For example, the following proof of $\vdash P \to P$ copies a formula from one line of the proof to the other in order to introduce a trivial implication:

1.	P	assumption
2.	P	copy 1
3.	$P \to P$	$\to_i$ 1-2 $\square$

Exercise 1.4. Prove $P \to (Q \to P)$ is valid.

1.3.1 Bi-implication ("if and only if")

Propositional logic often includes the bi-implication operator $\leftrightarrow$ also read as "*if and only if*" and sometimes written as *iff* (double f). A bi-implication $P \leftrightarrow Q$ is equivalent to the conjunction of two implications, pointing in opposite directions:

$$P \leftrightarrow Q \stackrel{\text{def}}{=} (P \to Q) \wedge (Q \to P)$$

This means that P and Q have exactly the same truth table, or we say they are *equivalent* (see Section 2 later).

Therefore to construct or deconstruct a logical bi-implication one can consider it as "implemented" by conjunction and implication, reducing the number of introduction/elimination rules that need to be remembered. Nonetheless, thinking about what these introduction/elimination rules would be is a nice exercise.

Exercise 1.5. (optional) Try to derive your own elimination and introduction rules for bi-implication. There is usually one introduction and two eliminations.

1.4 Disjunction ("or")

Recall the truth table for disjunction, which has three rows where $P \vee Q$ is true:

P	Q	$P \vee Q$
F	F	F
F	T	T
T	F	T
T	T	T

The fact that we can conclude $P \vee Q$ from either P or from Q separately justifies the following two introduction rules for disjunction in natural deduction:

$$\frac{P}{P \vee Q}\,\vee_{i1} \qquad \frac{Q}{P \vee Q}\,\vee_{i2}$$

Example 4. Prove $(P \wedge Q) \to (P \vee Q)$ is valid.

1.	$P \wedge Q$	assumption
2.	P	$\wedge_{e1}$ 1
3.	$P \vee Q$	$\vee_{i1}$ 2
4.	$P \wedge Q \to P \vee Q$	$\to_i$ 1-3 $\square$

This could have been written equivalently as a natural deduction proof tree:

$$\dfrac{\boxed{\dfrac{\dfrac{P \wedge Q}{P}\wedge_{e1}}{P \vee Q}\vee_{i1}}}{(P \wedge Q) \to (P \vee Q)}\to_i$$

This will be the last tree-based proof we see; from now on we'll keep using Fitch style.

What about disjunction elimination? Given the knowledge that $P \vee Q$ is true, what can we conclude? Either P is true, or Q is true, or both are true. Therefore, we don't know exactly what true formulas we can derive from $P \vee Q$, we just know some possibilities.

The natural deduction way of eliminating disjunction is to have two subproofs as premises which are contingent on the assumption of either P or Q:

$$\dfrac{P \vee Q \qquad \boxed{\begin{array}{c}P\\ \vdots \\ R\end{array}} \qquad \boxed{\begin{array}{c}Q\\ \vdots \\ R\end{array}}}{R}\vee_e$$

Example 5. For any propositions P, Q, R then $(P \wedge Q) \vee (P \wedge R) \to P$ is valid.

1.	$(P \wedge Q) \vee (P \wedge R)$	assumption
2.	$P \wedge Q$	assumption
3.	P	$\wedge_{e1}$ 2
4.	$P \wedge R$	assumption
5.	P	$\wedge_{e1}$ 4
6.	P	$\vee_e$ 1, 2-3, 4-5
7.	$(P \wedge Q) \vee (P \wedge R) \to P$	$\to_i$, 1-6 $\square$

You can see that the application of disjunction elimination $\vee_e$ involves three things: a disjunctive formula (line 1) and two subproofs (lines 2-3 and lines 4-5) which respectively assume the two subformulas of disjunction and conclude with the same formula (P), which forms the conclusion of the subproof on line 6.

An important point to remember is that **you cannot just eliminate a disjunction $P \vee Q$ into one side, e.g. to P.** This would be *unsound* as we can see from the truth table and it is not what disjunction elimination provides us. If $P \vee Q$ is true it might be because Q is true, and not because of P, so we cannot just conclude P. Consider the true statement (*This module has module code CO519*) $\vee$ (*It sunny every day in England*)— we cannot from this conclude that it is true that *It is sunny every day in England*.

Exercise 1.6. Prove $P \vee Q \to Q \vee P$ is valid.

Remark. From looking at the truth table for disjunction, one might wonder why disjunction elimination does not look like:

$$\dfrac{P \vee Q \qquad \boxed{\begin{array}{c} P \\ \vdots \\ R \end{array}} \qquad \boxed{\begin{array}{c} Q \\ \vdots \\ R \end{array}} \qquad \boxed{\begin{array}{c} P \wedge Q \\ \vdots \\ R \end{array}}}{R} \vee_e$$

This would match more closely the idea of reading the truth-table "backwards" from right-to-left on true values of $P \vee Q$. The reason we don't have this is that natural deduction strives for minimality and the third subproof with assumption $P \wedge Q$ is redundant since if we have $P \wedge Q$ true we can apply either the subproof $\boxed{P \ldots R}$ or the subproof $\boxed{Q \ldots R}$ by first applying $\wedge_{e1}$ or $\wedge_{e2}$ to the assumption $P \wedge Q$ to get P or Q respectively.

1.5 Negation

Negation introduction and elimination are given by:

$$\dfrac{\boxed{\begin{array}{c} P \\ \vdots \\ \bot \end{array}}}{\neg P} \neg_i \qquad \dfrac{P \qquad \neg P}{\bot} \neg_e$$

Introduction says that given a subproof that assumes P but ends in falsehood $\bot$ then we know $\neg P$. This is similar to the notion of *proof by contradiction*, which is derived from negation introduction (see below).

Elimination states that given a proof of P and a simultaneous proof of $\neg P$ then we conclude falsehood $\bot$, i.e., we have a logical inconsistency on our hands and so end up proving false: P and $\neg P$ cannot both be true at the same time.

Example 6. For all P, Q then $P \to Q \vdash \neg Q \to \neg P$.

1.	$P \to Q$	premise	
2.	$\neg Q$	assumption	
3.	P	assumption	
4.	Q	$\to_e$ 1, 3	
5.	$\bot$	$\neg_e$ 2, 4	
6.	$\neg P$	$\neg_i$ 3-5	
7.	$\neg Q \to \neg P$	$\to_i$ 2-6	$\square$

Remark. This example is often given as a derived inference rule called *modus tollens*[3] that is similar to modus ponens (implication elimination):

$$\frac{P \to Q \quad \neg Q}{\neg P}\ mt$$

If an inference rule can be derived from others we say it is *admissible*. The system of rules we take as the basis for natural deduction reasoning contains no admissible rules.

Remark. If we want to prove a formula P is unsatisfiable then we can instead prove that $\neg P$ is valid (always true), hence proving that P is unsatisfiable (always false).

Exercise 1.7. Prove that $P \wedge \neg(P \vee Q)$ is unsatisfiable.

Remark. Some formulae are not valid, e.g., $P \to \neg P$, which can be seen from drawing its truth table. However, this formula is *satisfiable*, if P is false then $P \to \neg P$ is true. Natural deduction does not help us to prove satisfiability. Part C will look at algorithmic approaches to deciding satisfiability.

1.5.1 Double negation

A special rule called *double-negation elimination* removes double negations on a proposition:

$$\frac{\neg\neg P}{P}\ \neg\neg_e$$

Example 7. The principle of *proof by contradiction* is represented by following the derived inference rule:

$$\frac{\begin{array}{|c} \neg P \\ \vdots \\ \bot \end{array}}{P}\ \text{PBC}$$

That is, if we assume $\neg P$ and conclude $\bot$ (i.e. we get a contradiction), then we have P. To show how to derive this, let the subproof in the above rule be called Δ, then we construct the following proof:

1.	$\neg P$	ass.
2.	Δ	$\vdots$
3.	$\bot$	
4.	$\neg\neg P$	$\neg_i$ 1-3
5.	P	$\neg\neg_e$ 4

Of course, Δ might be longer than 3 lines, but we use the numbering above proof for clarity.

[3] *modus tollens* is short for the Latin phrase *modus tollendo tollens* which means "the way that denies by denying".

1.6 Truth and falsity

If we have $\bot$ (false), then we can derive any formula:

$$\frac{\bot}{P}\,\bot_e$$

The embodies the principle that if we have logical inconsistency then anything goes.

There is no $\bot$ introduction as such, though $\neg_e$ provides a kind of $\bot$ introduction (from conflicting formula). Dually, we can always introduce truth from no premises, but there is no elimination:

$$\frac{}{\top}\,\top_i$$

1.7 A further derived rule: Law of Excluded Middle

An interesting rule that we can derive in the propositional logic is called the *Law of Excluded Middle* or LEM for short. It says that for any formula P we have the following valid rule:

$$\frac{}{P \vee \neg P}\,\text{LEM}$$

i.e., for any formula P, either P is true or $\neg P$ is true. Here is its derivation:

1.	$\neg(P \vee \neg P)$	ass.
2.	P	ass.
3.	$P \vee \neg P$	$\vee_{i1}$ 2
4.	$\bot$	$\neg_e$ 3, 1
5.	$\neg P$	$\neg_i$ 2-4
6.	$P \vee \neg P$	$\vee_{i2}$ 5
7.	$\bot$	$\neg_e$ 6, 1
8.	$\neg\neg(P \vee \neg P)$	$\neg_i$ 1-7
9.	$P \vee \neg P$	$\neg\neg_e$ 8 $\square$

This rule can be useful in particular proofs.

Exercise 1.8. Using LEM, prove that $P \to Q \vdash \neg P \vee Q$ is valid.

Aside: constructive vs non-constructive logic In this course, we study a particular kind of propositional logic called *classical* or *non-constructive* logic. Another variant is known as *intuitionistic* or *constructive* logic which has a slightly different set of proof rules: $\neg\neg_e$ is not included. By removing double-negation elimination we can no longer derive proof-by-contradiction or LEM.

The central principle of constructive logic is to reason about *provability* rather than *truth*, which is what we reason about in classical logic. In constructive logic, a formula P represents the proof of formula P: a mathematical object witnessing the truth of P. The inference rules of natural deduction are now about preserving proof rather than truth, e.g., conjunction elimination says given a proof of $P \wedge Q$ then we can prove P.

In constructive logic, $\neg\neg_e$ is rejected since it would mean we can get a proof of P from a proof of the negation of the negation of P, but the proof of $\neg\neg P$ is not really a proof of P. The presence of $\neg\neg_e$ is particularly troublesome for provability when used in the derivation of LEM. If LEM was allowed in constructive logic, then it would be saying that for any formula P we have constructed either a proof of P or a proof of $\neg P$. But what is that proof and where has it come from? It would be out of thin air! (since LEM has no premises). The essence of constructive logic is to disallow such things so that we always know we have

a concrete proof (sometimes called a *witness*) for our formulas, constructed from proofs of its subformulas or premises. Section 1.2.5 of the Huth and Ryan textbook gives more detail and shows an example mathematical proof about rational numbers in classical logic which cannot be proved constructively.

A useful thing about constructive logics is that they correspond to type systems in functional programming: a result known as the *Curry-Howard correspondence*. This is a rich source of ideas in programming languages. Unfortunately, we will not have time to study that here but it will appear at the end of CO545: *Functional and Concurrent Programming*.

2 Algebraic properties of logic

In Section 1.3.1, we saw *bi-implication* $\leftrightarrow$, a derived logical operator where:

$$P \leftrightarrow Q \stackrel{\text{def}}{=} (P \rightarrow Q) \wedge (Q \rightarrow P)$$

If there is a bi-implication between two formulas then it means their truth tables are exactly the same and we can see the two formulas as equivalent. From the natural deduction rules (or from the truth tables) a number of general equivalences can be derived which give us algebraic laws about propositional formula (this was first described by Boole in 1847). We'll use the operator $=$ to denote equivalent formula, where $P = Q$ can be read as "P is equivalent to Q" (or that there is a bi-implication). We can thus prove equivalences by proving the bi-implication of the two formula.

The following lists a number of equivalences between general propositional formula which amounts to algebraic properties of conjunction and disjunction:

property	conjunction	disjunction
idempotence	$P \wedge P = P$	$P \vee P = P$
commutativity	$P \wedge Q = Q \wedge P$	$P \vee Q = Q \vee P$
associativity	$(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$	$(P \vee Q) \vee R = P \vee (Q \vee R)$
unitality	$P \wedge \top = P$	$P \vee \bot = P$
annihilation	$P \wedge \bot = \bot$	$P \vee \top = \top$

We will often refer to these as "algebraic laws" or "axioms" though we can derive them (prove them) via natural deduction.

(Aside: you might like to think about other notions in mathematics that have axioms of a similar form (or a subset of these axioms), e.g. integer addition is commutative, associative, has 0 as its unit, but is not idempotent and does not have an annihilator).

Exercise 2.1. Prove the algebraic property of unitality for conjunction, i.e., that $P \wedge \top = P$.

"Distributivity" and "absorption" laws give us a relationship between $\wedge$ and $\vee$:

$$\begin{array}{rll} \textit{distributivity} & (P \vee Q) \wedge R & = (P \wedge R) \vee (Q \wedge R) \\ & (P \wedge Q) \vee R & = (P \vee R) \wedge (Q \vee R) \\ \textit{absorption} & (P \wedge Q) \vee P & = P \\ & (P \vee Q) \wedge P & = P \end{array}$$

De Morgan's two laws give us a useful interaction between negation and conjunction and disjunction:

$$\begin{array}{rll} \textit{De Morgan's} & \neg(P \wedge Q) & = \neg P \vee \neg Q \\ & \neg(P \vee Q) & = \neg P \wedge \neg Q \end{array}$$

"Complementation" laws give us interaction between a formula and its negation:

$$\begin{array}{rll} \textit{complementation} & P \wedge \neg P & = \bot \\ & P \vee \neg P & = \top \end{array}$$

Note that the second complementation law here only hold in classical (non-constructive) logic (which is what we primarily study here) where we allow the law of excluded middle, derived from double-negation elimination. Relatedly, the following law, known as "involution" only holds when we have double negation elimination in our logic:

$$\textit{involution} \qquad \neg\neg P \;=\; P$$

This notion of equivalence, given by the operator $=$, is symmetric, transitive, reflexive, and a *congruence* with respect to all the operators of logic. This means that if we have an equivalence between two formula we can get an equivalence between larger formulas by "plugging" the first equivalence into a template for a formula. For example, we get the following congruence property for conjunction:

$$P = Q \;\;\Rightarrow\;\; P \wedge R = Q \wedge R$$

We can think of this as plugging the equivalence $P = Q$ into a formula template $- \wedge R$. A similar congruence property holds for the template $R \wedge -$ and for all the other operators, e.g., disjunction (with templates $R \vee -$ and $- \vee R$), and implication, and negation.

The congruence property of $=$ is useful because it means we can use equivalences to "rewrite" parts of a propositional formula. We can then given *equational proofs* that some formula P is equivalent to another formula Q by applying algebraic laws one at a time, possibly to subparts of formulas.

Example 8. Prove that $P \vee (Q \wedge \neg P) = Q \vee P$.

We can proceed in the following steps where the subformula to which I have applied an equality is underlined and the rule which I am applying is written on the right:

$$\begin{aligned}
&\underline{P \vee (Q \wedge \neg P)} && \{\textit{distributity}\}\\
&= (P \vee Q) \wedge \underline{(P \vee \neg P)} && \{\textit{complementation}\}\\
&= \underline{(P \vee Q) \wedge \top} && \{\textit{unitality of conjunction}\}\\
&= \underline{P \vee Q} && \{\textit{commutativity of disjunction}\}\\
&= Q \vee P && \square
\end{aligned}$$

Exercise 2.2. Prove that $P \wedge \neg(P \wedge Q) = P \wedge \neg Q$.

3 Exercises

This section collects together the exercises given so far. They may not all be covered in lectures, so they provide useful additional examples to practise on.

Exercise 1.1. Prove that $(P \wedge Q) \wedge R \vdash P \wedge (Q \wedge R)$ is valid by instantiating and stacking together inference rules.

Exercise 1.2. Rewrite your proof to Exercise 1.1 using Fitch style.

Exercise 1.3. Recall that $P \to Q = \neg P \vee Q$. Show this is true by comparing the truth tables for each side of this equation.

Exercise 1.4. Prove $P \to (Q \to P)$ is valid.

Exercise 1.5. (optional) Try to derive your own elimination and introduction rules for bi-implication. There is usually one introduction and two eliminations.

Exercise 1.6. Prove $P \vee Q \to Q \vee P$ is valid.

Exercise 1.7. Prove that $P \wedge \neg(P \vee Q)$ is unsatisfiable.

Exercise 1.8. Using LEM, prove that $P \to Q \vdash \neg P \vee Q$ is valid.

Exercise 2.1. Prove the algebraic property of unitality for conjunction, i.e., that $P \wedge \top = P$.

Exercise 2.2. Prove that $P \wedge \neg(P \wedge Q) = P \wedge \neg Q$.

4 Collected rules of natural deduction

	Introduction	*Elimination*
$\wedge$	$\dfrac{P \quad Q}{P \wedge Q}\,\wedge_i$	$\dfrac{P \wedge Q}{P}\,\wedge_{e1} \qquad \dfrac{P \wedge Q}{Q}\,\wedge_{e2}$
$\vee$	$\dfrac{P}{P \vee Q}\,\vee_{i1} \qquad \dfrac{Q}{P \vee Q}\,\vee_{i2}$	$\dfrac{P \vee Q \quad \boxed{\begin{array}{c}P\\\vdots\\R\end{array}} \quad \boxed{\begin{array}{c}Q\\\vdots\\R\end{array}}}{R}\,\vee_e$
$\rightarrow$	$\dfrac{\boxed{\begin{array}{c}P\\\vdots\\Q\end{array}}}{P \rightarrow Q}\,\rightarrow_i$	$\dfrac{P \rightarrow Q \quad P}{Q}\,\rightarrow_e$
$\neg$	$\dfrac{\boxed{\begin{array}{c}P\\\vdots\\\bot\end{array}}}{\neg P}\,\neg_i$	$\dfrac{P \quad \neg P}{\bot}\,\neg_e$
$\top$	$\dfrac{}{\top}\,\top_i$	
$\bot$		$\dfrac{\bot}{P}\,\bot_e$
$\neg\neg$	(derivable: $\dfrac{P}{\neg\neg P}\,\neg\neg_i$)	$\dfrac{\neg\neg P}{P}\,\neg\neg_e$

These are all the rules we have and need for propositional proofs. You should aim to know all of the above rules by the end of the course/exam.

We derived other useful inference rules from these rules, like modus tollens, proof-by-contradiction, and law-of-excluded-middle. They are useful to know but the table above gives the essential rules for propositional proofs.

Appendix: Sequent-style natural deduction **(not examined)**

Recall from Section 1.1.1 that a sequent is a compact representation of a formula P along with any assumptions used to deduce it, written in the form: $P_1, \ldots, P_n \vdash P$. The turnstile symbol $\vdash$ is read as *entails* and the premises to the left are called the *context* of assumed formulas (or *assumptions*). The right-hand side is the *conclusion*. For example, the judgment $P, Q \vdash P \wedge Q$ captures the idea of conjunction introduction.

An alternate formulation of natural deduction gives the usual introduction and elimination rules in sequent form, making explicit the assumption context of the formula. This sequent-style of natural deduction is not assessed in CO519, but is included here for completeness and to help with any wider reading.

A key rule that was implicit in the previous formulation of natural deduction is the use of an assumption as a formula. This is usually called the *axiom* rule:

$$\frac{}{\Gamma, P \vdash P}\ (axiom)$$

This says that given some context with an assumption P and any other assumptions, represented by the Greek symbol Γ (uppercase gamma)[4] then we can conclude P. This is similar to the idea of copying in Fitch-style proofs.

The order of assumptions on the left of $\vdash$ is not important. The sequent style captures that there may be other assumptions Γ in scope. A meta rule says that we can add arbitrary redundant assumptions into our context (called *weakening*):

$$\frac{\Gamma \vdash A}{\Gamma, \Gamma' \vdash A}\ (weaken)$$

This is useful when we have two subproofs that we want to make have the same set of assumptions (see below). The rest of the rules are introduction and elimination rules.

Conjunction

$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\wedge_i \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P}\wedge_{e1} \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}\wedge_{e2}$$

These rules are very similar to the previously shown natural deduction rules, but they now carry a context of assumptions Γ. If the context doesn't match between two premises, weakening (above) can be applied so that they match.

Disjunction

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q}\vee_{i1} \qquad \frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}\vee_{i2} \qquad \frac{\Gamma \vdash P \vee Q \quad \Gamma, P \vdash R \quad \Gamma, Q \vdash R}{\Gamma \vdash R}\vee_e$$

Disjunction elimination is less unruly than the previous formulation, but has the same meaning. Note, we extend the context of assumptions with P and Q in the last two premises.

Implication

$$\frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\to_e \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\to_i$$

As an example, here is the proof of $P \wedge Q \to P \vee Q$ in this style:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{P \wedge Q \vdash P \wedge Q}\,axiom}{P \wedge Q \vdash P}\wedge_{e1}}{P \wedge Q \vdash P \vee Q}\vee_{i1}}{\vdash (P \wedge Q) \to (P \vee Q)}\to_i$$

Negation, falsity, and truth

$$\frac{\Gamma, P \vdash \bot}{\Gamma \vdash \neg P}\neg_i \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash \neg P}{\Gamma \vdash \bot}\neg_e \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash P}\bot_e \qquad \frac{}{\Gamma \vdash \top}\top_i$$

[4]Gamma Γ is the third letter of the Greek alphabet, corresponding to Latin C, hence Γ for "Context". Logicians like Greek as it gives them lots more symbols to use to represent things tersely. These are conventions which take some getting used to.

Part B

Modelling systems using logic

In this part we are going to briefly cover using propositional formula to model systems. This is a common approach in hardware design where a complete or partial model of a circuit or processor is defined in logic, against which specifications of particular properties are checked. The starting point is to work out a good way to represent/model a system as a logical formula. In this part of the course, we will consider systems modelled as simple state machines, with states and transitions between the states describing how a system can change/evolve. We will then convert this model into a propositional formula.

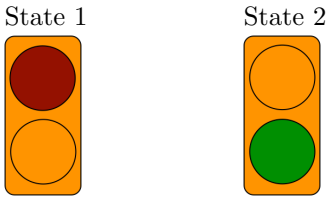
To verify a system based on its model we then need a specification of either good behaviour that we want to make sure follows from our model or of bad behaviour which we want to ensure does not follow from the model. We formulate such a specification as a propositional formula *spec*, and then prove that the following is valid:

$$\mathit{model} \to \mathit{spec}$$

The use of implication means that if the model holds then the specification must hold. Alternatively, and equivalently, we could state this as judgment $\mathit{model} \vdash \mathit{spec}$, i.e., the specified behaviour follows from the model.

1 State-transition models as propositions

States Our running example will be a very simple traffic light comprising a red light and a green light, with two possible states:



Either the red light is on (left) or the green light is on (right), but never both at the same time, and there is always at least one light on. We will use two atoms (propositional variables that can be true or false) r and g to represent the state of each light separately, where:

- r means the red light is on; $\neg\mathsf{r}$ therefore means the red light is off;
- g means the green light is on; $\neg\mathsf{g}$ means the green light is off.

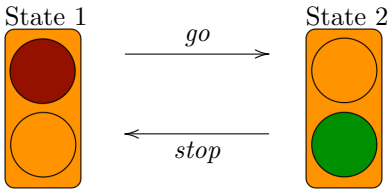
The two valid states of the system can then be modelled as two propositions:

State 1	State 2
$\mathsf{r} \wedge \neg\mathsf{g}$	$\neg\mathsf{r} \wedge \mathsf{g}$

For n-propositional atoms there are 2^n possible states that can be modelled. Thus in our model, there are four possible states, two of which we want to treat as valid (the above two).

Exercise 1.1. Define a propositions for each of the two invalid states in the traffic light.

State transitions So far we have modelled the states as propositions, but we also want to model the behaviour of the system in terms of the possible transitions between states. We can represent this with a simple diagram:



i.e., when just the red light is on it is possible to transition to a state with just the green light on, and back again.

To model state transitions we will introduce two additional atoms that model the future state of the lights in the system:

- r' for the red light being on in the *next time step*;
- g' for the green light being on in the *next time step*.

We can now express the above two transitions as implications:

$$\begin{aligned} (go): &\quad \mathsf{r} \wedge \neg \mathsf{g} \;\rightarrow\; \neg \mathsf{r}' \wedge \mathsf{g}' \\ (stop): &\quad \neg \mathsf{r} \wedge \mathsf{g} \;\rightarrow\; \mathsf{r}' \wedge \neg \mathsf{g}' \end{aligned}$$

Said another way, (go) defines that if State 1 is true now we can move to State 2 in the future (in the "next" time step of the system), and $(stop)$ conversely defines that if State 2 is true now we can move to State 1 in the future.

We can now describe the full transition behaviour of the system as the conjunction of the above two formula:

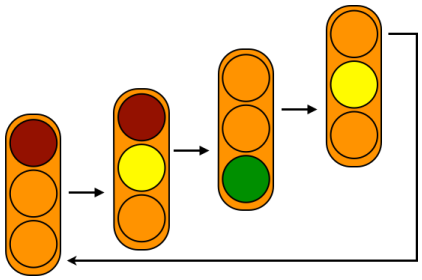
$$model = (\mathsf{r} \wedge \neg \mathsf{g} \;\rightarrow\; \neg \mathsf{r}' \wedge \mathsf{g}') \wedge (\neg \mathsf{r} \wedge \mathsf{g} \;\rightarrow\; \mathsf{r}' \wedge \neg \mathsf{g}')$$

This provides our model of the system. You might be wondering why we don't add more to this, e.g., ruling out invalid states. We will come back to this point in Section 3.

A general approach The general approach to modelling a system in this way is to:

- Decide what to represent about the state space of the system and introduce propositional atoms for these components.
- Model future states using "next step" atoms (usually written with an apostrophe, and called the "primed" atoms, e.g., r' is read as "r prime").
- Write a propositional formula *model* using these variables which takes the conjunction of state transitions expressed as implications.

Exercise 1.2. Write down a model for a more realistic traffic light, i.e., that can be described by the following states and transitions:



2 Defining specifications as propositions

Consider the following property which we might want to check for our system:

If we are in a valid state and change state, then our new state is also valid.

We can abstract the notion of a valid state with the following meta-level operation (you can think of this as a syntax function mapping from the two propositions to a proposition):

$$\mathsf{valid\text{-}state}(r, g) = (r \wedge \neg g) \vee (\neg r \wedge g)$$

From this, we can then capture our specification as the proposition:

$$\mathit{specification} = \mathsf{valid\text{-}state}(\mathsf{r}, \mathsf{g}) \rightarrow \mathsf{valid\text{-}state}(\mathsf{r'}, \mathsf{g'})$$

i.e., a valid state now implies a valid state in the next time step. To verify that our system (based on its model) satisfies this property, we then need to prove that the following is true:

$$\begin{aligned}
&\mathit{model} \rightarrow \mathit{specification} \\
\equiv\ &((\mathsf{r} \wedge \neg \mathsf{g} \ \rightarrow\ \neg \mathsf{r}' \wedge \mathsf{g}') \wedge (\neg \mathsf{r} \wedge \mathsf{g} \ \rightarrow\ \mathsf{r}' \wedge \neg \mathsf{g}')) \rightarrow (\mathsf{valid\text{-}state}(\mathsf{r}, \mathsf{g}) \rightarrow \mathsf{valid\text{-}state}(\mathsf{r'}, \mathsf{g'})) \\
\equiv\ &((\mathsf{r} \wedge \neg \mathsf{g} \ \rightarrow\ \neg \mathsf{r}' \wedge \mathsf{g}') \wedge (\neg \mathsf{r} \wedge \mathsf{g} \ \rightarrow\ \mathsf{r}' \wedge \neg \mathsf{g}')) \rightarrow (((\mathsf{r} \wedge \neg \mathsf{g}) \vee (\neg \mathsf{r} \wedge \mathsf{g})) \rightarrow ((\mathsf{r'} \wedge \neg \mathsf{g'}) \vee (\neg \mathsf{r'} \wedge \mathsf{g'})))
\end{aligned}$$

This is quite a big proposition so we might not want to prove it by hand. Instead, in the next part of the course we are going to use an algorithmic technique for proving that this holds (Part C, specifically **Example 5** in the notes). We will later see that this does indeed hold, and thus the system (as described by the model) satisfies our specification.

3 When is a model a good model?

All models are wrong but some are useful (Box, 1978)

Indeed, a model is necessarily "wrong" in the sense that a model abstracts some of the details of a real system; we are not capturing every aspect of the system, such as: how are the transitions triggered? or, what happens if a car crashes into the traffic light? Some models, though eliding details, are useful in the sense that we can detect when the system does not behave according to our specification or we can verify that it always behaves according to our specification. Just enough detail is needed in the model to capture what we want to prove.

Our running example has the model:

$$\mathit{model} = (\mathsf{r} \wedge \neg \mathsf{g} \ \rightarrow\ \neg \mathsf{r}' \wedge \mathsf{g}') \wedge (\neg \mathsf{r} \wedge \mathsf{g} \ \rightarrow\ \mathsf{r}' \wedge \neg \mathsf{g}')$$

We might add to this model the explicit exclusion of invalid states. Let's define the condition of the invalid states via the meta-level function:

$$\mathsf{invalid\text{-}state}(r, g) = (\neg r \wedge \neg g) \vee (r \wedge g)$$

Then we could define an alternate model as:

$$\mathit{model2} = \mathit{model} \wedge \neg \mathsf{invalid\text{-}state}(\mathsf{r}, \mathsf{g})$$

Thus, the new model adds to the old model that the current state is not invalid.

In the case of the proving our specification from Section 2 (that valid states transition to valid states) we do not need this additional detail in the model. But we might want to have this more restrictive model if we want to prove, for example, that we can never be in

an invalid state, regardless of what state we started from. This can be captured by the new specification:

$$\mathit{specification2} = \mathsf{invalid\text{-}state}(\mathsf{r}, \mathsf{g}) \vee \mathsf{invalid\text{-}state}(\mathsf{r}', \mathsf{g}')$$

and proving that the following proposition is valid:

$$\mathit{model2} \rightarrow \neg \mathit{specification2}$$

Note we are using a negative property here: we show that the new model (*model2*) implies that the bad behaviour is not possible. For the old model (*model*), a similar proposition capturing the same idea is valid:

$$\mathit{model} \rightarrow \neg \mathit{specification2}$$

However, the following proposition is also valid!!!

$$\mathit{model} \rightarrow \mathit{specification2}$$

Why? If the right-hand side of the implication is true (we have an invalid state) the left-hand side is still true (the premise of each transition implication is false, therefore the implications are trivially true); the original model never excludes invalid states on their own, only as the result of a transition from a valid state. Thus, the original model was not a good model for checking the *specification2* property, but it was sufficient for *specification*.

Creating a rich enough model is up to you, and requires some care and thought about the domain and what properties are of interest.

Part C

Satisfiability for propositional logic

1 Satisfiable formula

This section covers *satisfiability* of propositional formula. Recall from Part A that a formula is *satisfiable* if it is true for a particular "assignment" of either true or false to variables in the formula. For example, $a \wedge b$ is satisfiable since we can make it true by setting a to be true and b to be true. We say this is the *satisfying assignment*, and will write this as:

$$\{a = \top,\ b = \top\}$$

A valid formula is trivially satisfiable since it is true no matter how we assign truth or falsehood to the variables.

The general problem of finding a satisfying assignment for a formula is known in computer science as the *Boolean satisfiability problem* or simply SAT for short. There are various algorithmic approaches to solving the SAT problem, i.e., for calculating a satisfying assignment for a propositional formula. SAT is used in many areas of formal verification, such as AI, planning, circuit design, and theorem provers, and is applied to problems with thousands of variables.

One approach is to exhaustively list all possible assignments of true/false to variables in a formula by constructing its truth tables. Thus for a formula with n variables we need to calculate 2^n rows. This is completely infeasible for problems with hundreds or even thousands of variables. Instead, we'll look at the *DPLL algorithm* which utilises properties of propositional logic to be more efficient for many problems.

We can use SAT solving algorithms to prove validity of a formula P by applying SAT to $\neg P$. If the algorithm shows us that $\neg P$ is unsatisfiable, that implies $\neg P$ is false for any assignment of its variables and thus P is true for any assignment of its variables, i.e., *valid*.

This is very useful for modelling problems where we have a complex model of some system in logic, and a complex specification, and we want to prove the validity of:

$$model \rightarrow specification \tag{1}$$

Instead of proving this valid using natural deduction, we can instead use an algorithm to show unsatisfiability of $\neg(model \rightarrow specification)$. If we find a satisfying assignment to this negated formula, then we have a counterexample to the original property, that is, a set of variable assignments which makes equation (1) false and thus shows us a configuration which leads to a bug in our system.

2 The DPLL algorithm

DPLL stands for *Davis-Putnam-Logemann-Loveland* (who proposed this algorithm)[5]. We'll go over the technique since its a nice algorithm and makes some clever use of the properties of propositional logic to simplify the exploration of the state space. Despite the fact it is more than 50 years old, DPLL still forms the basis of many SAT solvers, though there has been some progress since.

2.1 CNF

The input to the DPLL algorithm is a formula in *conjunctive normal form* (CNF for short). A formula in Conjunction Normal Form is a conjunction of disjunctions of literals. A literal is either a variable or the negation of a variable. Thus, CNF formula are of the form:

$$(x \vee \neg y \vee ...) \wedge (z \vee \ldots) \wedge \ldots \wedge (\neg w \vee x \vee y)$$

We refer to a disjunction of literals as a *clause*. For example, the following highlights the middle clause:

$$(x \vee \neg w) \wedge \underline{(y \vee z)} \wedge (\neg z \vee \neg x) \tag{2}$$

A CNF formula consists of a set of clauses all of which have to be true, since they are combined using conjunction. Within each clause, just one of literals has to be true since they are combined via disjunction.

Any formula can be converted into CNF by applying algebraic properties of logic (see Part A, Section 2) to rewrite a formula. Informally, this can be done by doing the following:

- Replace implication $\rightarrow$ with disjunction $\vee$ and negation $\neg$, via:

$$P \rightarrow Q = \neg P \vee Q$$

- Use De Morgan's law to push negation inside of disjunction and conjunction:

$$\neg(P \vee Q) = \neg P \wedge \neg Q$$
$$\neg(P \wedge Q) = \neg P \vee \neg Q$$

- Push disjunction inside and pull conjunction out (using distributive laws):

$$(P \wedge Q) \vee R = (P \vee R) \wedge (Q \vee R)$$
$$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$$

- Eliminate double negation $\neg\neg P = P$

By repeatedly applying these equations as rewrites (orienting the equalities from left to the right), we end up with a formula in CNF which is ready for DPLL.

[5]Actually the history is more complicated, just like the history of most science! In 1960 Davis and Putnam did some work on automatically showing validity of formulae following a different technique, and David, Logemann, and Loveland built on some of the ideas to create their SAT in algorithm 1962.

2.2 DPLL, step-by-step

DPLL can be summarised in pseudo code as follows, which has four main steps which I remember using the acronym **TUPS**:

$$
\begin{aligned}
&\mathsf{DPLL}(\top) = \textit{satisfiable}\\
&\mathsf{DPLL}(\bot) = \textit{unsatisfiable}\\
&\mathsf{DPLL}(P) =\\
&\qquad 1.\ \textbf{\textit{T}}\textit{autology elimination}\\
&\qquad 2.\ \textbf{\textit{U}}\textit{nit propagation}\\
&\qquad 3.\ \textbf{\textit{P}}\textit{ure literal elimination}\\
&\qquad 4.\ \textbf{\textit{S}}\textit{plit on a variable: choose a variable } v\\
&\qquad\quad \mathsf{DPLL}(P')\{v = \top\}\\
&\qquad\quad \mathsf{DPLL}(P')\{v = \bot\}
\end{aligned}
$$

Note that this is a recursive algorithm which branches into two executions at the last step. The four steps gradually reduce the size of the input CNF formula and create an assignment for its variables (setting variables to be true or false).

The first step is a simplification step using properties of propositional logic and the structure of CNF formula. The next two steps provide simplification and can give us some satisfying assignments. The last step splits the problem into two by picking a variable and repeating the DPLL procedure with that variable assigned to be true in one branch (written above as $\{v = \top\}$) and repeating DPLL with that variable assigned to false ($\{v = \bot\}$).

Step 1. Tautology elimination Consider a formula in CNF where one clause (highlighted in yellow bellow) contains both x and $\neg x$, e.g., a formula of the form:

$$P \wedge (\underline{\ldots x \vee \ldots \vee \neg x}) \wedge Q$$

Such a formula can be simplified by completely removing the highlighted clause. This is because a disjunction of a formula with its negation is a tautology (always true) by *complementation*: $P \vee \neg P = \top$. Furthermore, since disjunction of anything with truth is equivalent to truth: $P \vee \top = \top$ and for conjunction $P \wedge \top = P$ (*unitality* properties) we can completely filter out clauses that have a tautology, i.e.

$$P \wedge (\underline{\ldots x \vee \ldots \vee \neg x}) \wedge Q \quad \longrightarrow \quad P \wedge (\top) \wedge Q \quad \longrightarrow \quad P \wedge Q$$

Note that this doesn't tell us whether to assign true or false to x yet. So far we just know that this clause didn't depend on the truth or falsehood of x because it contained this tautology.

Step 2: Unit propagation In CNF terminology, a "unit" is a clause which contains just one literal (a variable or negation of a variable), e.g.,

$$P \wedge (\underline{x}) \wedge Q \quad \textit{or} \quad P \wedge (\underline{\neg x}) \wedge Q$$

where the "unit" clauses are highlighted.

If we see a formula in the left form, then we know that x must be true if we want the overall formula to be true, since we are taking the conjunction of x with all the other formula. If we have a formula with the right form, then we know that x must be false so that $\neg x$ is true to make the overall formula true. Thus in the left case we get the assignment $x = \top$ and in the right case we get $x = \bot$. We then *propagate* this assignment to the rest of the clauses P and Q, replacing any occurrences of x with its assignment:

$$
\begin{aligned}
P \wedge (\underline{x}) \wedge Q \quad &\longrightarrow \quad \textit{replace}(x, \top, P) \wedge \textit{replace}(x, \top, Q) \qquad \{x = \top\}\\
\textit{or} \quad P \wedge (\underline{\neg x}) \wedge Q \quad &\longrightarrow \quad \textit{replace}(x, \bot, P) \wedge \textit{replace}(x, \bot, Q) \qquad \{x = \bot\}
\end{aligned}
$$

The assignment that is output by this step is written on the right-hand side. We have also removed the unit clause since it has been made true and is therefore redundant now. Thus, unit propagation gives us both a simplification and an assignment when we have unit clauses in our formula.

Here we are using a function on the syntax of formulas called *replace* where $\mathit{replace}(x, \top, P)$ means replace/substitute any occurrence of x in P with $\top$, and similarly $\mathit{replace}(x, \bot, P)$ replaces any occurrence of x in P with $\bot$. For example, $\mathit{replace}(x, \top, \neg x \vee \neg y)$ would be the formula $\neg\top \vee \neg y$. This idea of substituting one proposition for another will crop up again in Part D when we look at first-order logic proofs.

Example 9. The following formula has a unit clause highlighted in yellow and underlined:

$$(\neg x \vee \neg y) \wedge \underline{x} \wedge (y \vee x)$$

Since it is "positive" (i.e., not negated) then unit propagation emits the assignment $\{x = \top\}$ and then propagates this assignment by replacement:

$$\mathit{replace}(x, \top, (\neg x \vee \neg y)) \wedge \mathit{replace}(x, \top, (y \vee x))$$

Applying the replacement function then gives us:

$$(\neg\top \vee \neg y) \wedge (y \vee \top)$$

which simplifies via the following steps

$$\begin{aligned} & (\neg\top \vee \neg y) \wedge (y \vee \top) \\ = {} & (\neg\top \vee \neg y) \wedge \top \\ = {} & (\neg\top \vee \neg y) \\ = {} & (\bot \vee \neg y) \\ = {} & \neg y \end{aligned}$$

In DPLL, we need only apply simplifications that involve disjunction or conjunction with $\top$ or $\bot$. It is straightforward to build this into an implementation so that these steps are implicit and automatic: essentially the "replacement" can remove entire clauses when we know we are making a literal true, or remove literals when they are false.

Performing just one step of unit propagation has greatly simplified our original formula from $(\neg x \vee \neg y) \wedge (x) \wedge (y \vee x)$ to $\neg y$, along with giving the assignment $\{x = \top\}$. Since $\neg y$ is itself a unit, we apply unit propagation again, yielding the assignment $\{y = \bot\}$ and the formula $\top$. Thus we have reached a satisfying assignment $\{x = \top, y = \bot\}$ just by applying unit propagation twice.

Exercise 2.1. Convince yourself that $\{x = \top, y = \bot\}$ is a satisfying assignment for $(\neg x \vee \neg y) \wedge (x) \wedge (y \vee x)$ by substituting the variables for their assignment and simplifying.

Exercise 2.2. Perform unit propagation on the formula $(y \vee \neg x) \wedge (\neg y)$.

Step 3: Pure literal elimination In DPLL terminology, a pure literal is a literal whose negation does not appear anywhere else in the entire formula, e.g.

$$P \wedge (\ldots \vee \underline{x} \vee \ldots) \wedge Q$$

where its dual $\neg x$ does not appear in P nor is it in Q. For example, x is a pure literal in this formula:

$$(x \vee y \vee \neg z) \wedge (\neg y \vee z \vee x)$$

The literal x can appear multiple times, what makes it "pure" is that its negation never appears anywhere. This means we can assign x to be true. We could assign x to be false, but it might be the wrong decision later, for example if the other literals in the clause turn out to false as well. It turns out that the most useful approach is to assign x to true by following the principle of progressing towards a true formula as quickly as possible.

A pure literal might also be negative, for example:

$$P \wedge (\ldots \vee \underline{\neg x} \vee \ldots) \wedge Q$$

where x does not appear in P and Q. In this case, we can assign x to be false, and propagate this assignment into P and Q.

Pure literal elimination therefore has the two dual rules:

$$\begin{array}{lllll} & P \wedge (\ldots \vee \underline{x} \vee \ldots) \wedge Q & \longrightarrow & \mathit{replace}(x, \top, P) \wedge \mathit{replace}(x, \top, Q) & \{x = \top\} \\ \mathit{or} & P \wedge (\ldots \vee \underline{\neg x} \vee \ldots) \wedge Q & \longrightarrow & \mathit{replace}(x, \bot, P) \wedge \mathit{replace}(x, \bot, Q) & \{x = \bot\} \end{array}$$

Example 10. The following formula has pure literal x:

$$(\underline{x} \vee y \vee \neg z) \wedge (\neg y \vee x) \wedge (y \vee z)$$

Pure literal elimination then produces the assignment $\{x = \top\}$, eliminates the first clause (since it is now true), and rewrites the rest of the formula as follows:

$$\begin{aligned} & \mathit{replace}(x, \top, (\neg y \vee x) \wedge (y \vee z)) \\ = \; & (\neg y \vee \top) \wedge (y \vee z) \\ = \; & (y \vee z) \end{aligned}$$

We are left with a clause where both y and z are now pure literals, so pure literal elimination can be applied to either.

Let's pick y and assign it to $\{y = \top\}$. This gives us $\top$ as the resulting formula. Thus, we have found that $(x \vee y \vee \neg z) \wedge (\neg y \vee x) \wedge (y \vee z)$ is satisfiable with $\{x = \top, y = \top\}$, and it doesn't matter whether z is true or false.

Exercise 2.3. Apply pure literal elimination to the formula:

$$(\neg x \vee y \vee \neg z) \wedge (\neg y \vee z) \wedge (\neg x \vee z)$$

Step 4: Split a variable The previous steps have applied the rules of logic, and the shape of CNF formula, to make simplifications and assignments. Once we've done all that we can with those steps, the last step falls back to a "brute force" approach. We pick a variable, and "split it", that is, we assign it to be true and apply DPLL on the result and separately assign it to false and apply DPLL on the result. This results in us running two DPLL separate procedures from this point forwards. That is, given a formula P, splitting recursively calls the DPLL algorithm under two new assignments:

$$\begin{array}{ll} & \mathsf{DPLL}(\mathit{replace}(x, \top, P)) \qquad \{x = \top\} \\ \mathit{and} & \mathsf{DPLL}(\mathit{replace}(x, \bot, P)) \qquad \{x = \bot\} \end{array}$$

These recursive calls will be separate, producing possibly different assignments in each.

Example 11. The following formula has no tautologies, units, or pure literals (i.e., none of the first three steps of DPLL apply):

$$(x \vee \neg y) \wedge (y \vee z) \wedge (\neg z \vee \neg x)$$

The splitting step chooses any variable in the formula, let's say z, and splits the DPLL process. Let's follow the branch with the assignment $\{z = \top\}$, which is then propagated to the rest of the formula by replacement:

$$\begin{aligned} &= \quad \mathit{replace}(z, \top, (x \vee \neg y) \wedge (y \vee z) \wedge (\neg z \vee \neg x)) \qquad \{z = \top\} \\ &= \quad (x \vee \neg y) \wedge (y \vee \top) \wedge (\bot \vee \neg x) \\ &= \quad (x \vee \neg y) \wedge (\neg x) \end{aligned}$$

The other branched DPLL process has assignment $\{z = \bot\}$ which produces:

$$\begin{aligned} &= \quad \mathit{replace}(z, \bot, (x \vee \neg y) \wedge (y \vee z) \wedge (\neg z \vee \neg x)) \qquad \{z = \bot\} \\ &= \quad (x \vee \neg y) \wedge (y \vee \bot) \wedge (\top \vee \neg x) \\ &= \quad (x \vee \neg y) \wedge (y) \end{aligned}$$

We then return to step 1 for both branches, and continue with two separate instances of the DPLL procedure on the above two formulas.

Exercise 2.4. Apply the splitting step to the following formula on variable y:

$$(x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (y \vee z)$$

Write down the two resulting formula after replacement and doing trivial simplifications.

The following gives an example putting all the steps together.

Example 12. Consider the following formula: $(\neg x \to y) \wedge x$. Is it satisfiable? Before we apply DPLL we first have to convert it to CNF:

$$\begin{aligned} & (\neg x \to y) \wedge x \quad \{\to \textit{ as } \vee\} \\ = \quad & (\neg\neg x \vee y) \wedge x \quad \{\textit{Double negation elimination}\} \\ = \quad & (x \vee y) \wedge x \end{aligned}$$

Now we have the formula in CNF, we can apply DPLL. I'll write the steps in a table:

Step	Note	Resulting CNF formula	Satisfying assignments
	start	$(x \vee y) \wedge x$	
1	Tautology elim	$(x \vee y) \wedge x$	
2	Unit propagation x	$\top$	$\{x = \top\}$

Thus the original formula is satisfiable with assignment $x = \top$ (and y can be anything). We reached this conclusion quickly, just by applying tautology elimination and unit propagation. This tabulated form is handy for small examples; you might like to use it for class exercises.

Example 13. The following is the proposition $\neg(\mathit{model} \to \mathit{specification})$ for the traffic light example shown in Part B, but in CNF:

$$\begin{aligned} & (\neg r \vee g') \wedge (\neg r \vee \neg r') \wedge (\neg g \vee \neg g') \wedge (\neg g \vee r') \\ \wedge\ & (r \vee g) \wedge (\neg r \vee \neg g) \wedge \underline{(\neg r' \vee r')} \\ \wedge\ & (\neg r' \vee g') \wedge (\neg g' \vee r') \wedge \underline{(\neg g' \vee g)} \end{aligned}$$

We'll apply DPLL to it here. Due to the large size of the formula, I'll use a more free-form style rather than the tabulated form used above.

Step 1: Tautology elimination - There are two immediate tautologies in the above formula which are highlighted. These are eliminated to give:

$$\begin{aligned} & (\neg r \vee g') \wedge (\neg r \vee \neg r') \wedge (\neg g \vee \neg g') \wedge (\neg g \vee r') \\ \wedge\ & (r \vee g) \wedge (\neg r \vee \neg g) \\ \wedge\ & (\neg r' \vee g') \wedge (\neg g' \vee r') \end{aligned}$$

There are no units or pure literals so we move to step 4.

Step 4: Splitting a variable - Choose r:

$\{r = \top\}$ yielding: $(\neg\top \vee g') \wedge (\neg\top \vee \neg r') \wedge$ $(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(\top \vee g) \wedge (\neg\top \vee \neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$ which simplifies to $g' \wedge (\neg r') \wedge (\neg g \vee \neg g') \wedge$ $(\neg g \vee r') \wedge (\neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$	$\{r = \bot\}$ yielding: $(\neg\bot \vee g') \wedge (\neg\bot \vee \neg r') \wedge$ $(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(\bot \vee g) \wedge (\neg\bot \vee \neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$ which simplifies to $(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(g) \wedge (\neg r' \vee g') \wedge (\neg g' \vee r')$
1. Unit propagation on the unit g' $g' \wedge (\neg r') \wedge (\neg g \vee \neg g') \wedge$ $(\neg g \vee r') \wedge (\neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$ yields assignment $\{g' = \top\}$ giving $(\neg r') \wedge (\neg g) \wedge$ $(\neg g \vee r') \wedge$ $(\neg g) \wedge (r')$ *1. Unit propagation:* on the unit $\neg r'$ $(\neg r') \wedge (\neg g) \wedge$ $(\neg g \vee r') \wedge$ $(\neg g) \wedge (r')$ yields assignment $\{r' = \bot\}$ giving $(\neg g) \wedge (\neg g) \wedge (\neg g) \wedge \bot$ simplifies to $\bot$	*1. Unit propagation* on the unit g $(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(g) \wedge (\neg r' \vee g') \wedge (\neg g' \vee r')$ yields assignment $\{g = \top\}$ giving $(\neg g') \wedge (r') \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$ *1. Unit propagation:* on the unit $\neg g'$ $(\neg g') \wedge (r') \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$ yields assignment $\{g' = \bot\}$ giving $(r') \wedge (\neg r')$ *1. Unit propagation:* on the unit r' yields assignment $\{r' = \top\}$ giving $\bot$

Both branches have ended with $\bot$ hence $\neg(\textit{model} \to \textit{specification})$ is unsatisfiable and therefore $\textit{model} \to \textit{specification}$ is valid! Hurray!

Exercise 2.5. Apply the full DPLL procedure to the following formula (already in CNF):

$$(x \vee y \vee z) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y \vee \neg z)$$

3 Exercises from this section

Exercise 2.1. Convince yourself that $\{x = \top, y = \bot\}$ is a satisfying assignment for $(\neg x \vee \neg y) \wedge (x) \wedge (y \vee x)$ by substituting the variables for their assignment and simplifying.

Exercise 2.2. Perform unit propagation on the formula $(y \vee \neg x) \wedge (\neg y)$.

Exercise 2.3. Apply pure literal elimination to the formula:

$$(\neg x \vee y \vee \neg z) \wedge (\neg y \vee z) \wedge (\neg x \vee z)$$

Exercise 2.4. Apply the splitting step to the following formula on variable y:

$$(x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (y \vee z)$$

Write down the two resulting formula after replacement and doing trivial simplifications.

Exercise 2.5. Apply the full DPLL procedure to the following formula (already in CNF):

$$(x \vee y \vee z) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y \vee \neg z)$$

Part D
First-order logic

First-order logic (also called *predicate logic*) extends propositional logic with *quantification*: existential quantification $\exists$ ("*there exists*") and universal quantification $\forall$ ("*for all*"). A quantification $\forall x$ binds a variable x which range over the elements of some underlying *universe* which is external to the logic, e.g., quantifying over all people or objects. First-order logic also allows the use of relations, predicates (unary relations, also called *classifiers*) and functions, operating over elements of the universe, which can be defined externally and are domain-specific for whatever purpose the logic is being used.

Consider the following sentence:

Not all birds can fly

We can capture this in first-order logic using quantification and unary predicates. Let our universe be "animals" over which we informally define two predicates:

$$\mathsf{B}(x) \stackrel{\text{def}}{=} x \textit{ is a bird}$$
$$\mathsf{F}(x) \stackrel{\text{def}}{=} x \textit{ can fly}$$

As with propositional logic, we are studying the process and framework of the logic rather than physical reality; it is up to us how we assign the semantics of B and F above, but the semantics of quantification and logical operators is fixed by the definition of first-order logic.

We can then express the above sentence in first-order logic as:

$$\neg(\forall x.\mathsf{B}(x) \to \mathsf{F}(x)) \tag{3}$$

We can read this exactly as *it is not true that for all x, if x is a bird then x can fly.* Another way to write this is that there are some birds which cannot fly:

$$\exists x.\mathsf{B}(x) \wedge \neg\mathsf{F}(x) \tag{4}$$

If we have a universe and semantics for B and F that includes, for example, penguins, then both (3) and (4) will be true. We can prove that (3) and (4) are equivalent in first-order logic via two proofs, one for:

$$\neg(\forall x.\mathsf{B}(x) \to \mathsf{F}(x)) \vdash \exists x.\mathsf{B}(x) \wedge \neg\mathsf{F}(x)$$

and one for:

$$\exists x.\mathsf{B}(x) \wedge \neg\mathsf{F}(x) \vdash \neg(\forall x.\mathsf{B}(x) \to \mathsf{F}(x))$$

We will do this later once we have explained more about the meta theory of the logic.

1 Key concepts (meta theory) of first-order-logic

1.1 Names and binding

In propositional logic, variables range over propositions, i.e., their "type" is a proposition. For example, $x \wedge y$ has two propositional variables x and y which could be replaced with true or false, or with any other formula. In predicate logic, universal and existential quantifiers provide *variable bindings* which introduce variables ranging over objects in some fixed universe rather than over propositions. For example, the formula $\forall x.P$ binds a variable x

in the *scope* of P. That is, x is available within P, but not outside of it. A variable which does not have a binding in scope is called *free*.

For example, the formula below has free variables x and y and bound variables u and v:

$$\mathsf{P}(x) \lor \forall u.\,(\,\mathsf{Q}(y) \land \mathsf{R}(u) \to \exists v.\,(\,\mathsf{P}(v) \land \mathsf{Q}(x)\,)\,)$$

The following repeats the formula and highlights the binders in yellow, the bound variables in green, and the free variables in red:

$$\mathsf{P}(x) \lor \forall u\,.(\,\mathsf{Q}(y) \land \mathsf{R}(u) \to \exists v\,.(\,\mathsf{P}(v) \land \mathsf{Q}(x)\,)\,)$$

In the following formula, there are two syntactic occurrences of a variable called x, but semantically these are different variables:

$$\mathsf{Q}(x) \land (\forall x.\mathsf{P}(x))$$

The x on the left (used with a predicate Q) is free, whilst the x used with the predicate P is bound by the universal quantifier. Thus, these are semantically two different variables.

Alpha renaming The above formula is semantically equivalent to the following formula obtained by consistently renaming bound variables:

$$\mathsf{Q}(x) \land (\forall y.\mathsf{P}(y))$$

Renaming variables is a meta-level operation we can apply to any formula: we can rename a bound variable as long as we do not rename it to clash with any other free or bound variable names, and as long as we rename the variable consistently. This principle is more generally known as α-renaming (alpha renaming) and equality up-to renaming (equality that accounts for renaming) is known as α-equality. For example, writing α-equality as $=_\alpha$ the following equality and inequality hold:

$$\exists x.\mathsf{P}(x) \to \mathsf{P}(y) \quad =_\alpha \quad \exists z.\mathsf{P}(z) \to \mathsf{P}(y) \quad \neq_\alpha \quad \exists y.\mathsf{P}(y) \to \mathsf{P}(y)$$

The middle formula can be obtained from the left by renaming x to a fresh variable z. However, if we rename x to y (on the right) we conflate the bound variable with the previously free variable to the right of the implication; we accidentally capture the free occurence of y via the binding. The right-hand formula has a different meaning to the other two.

1.2 Substitution

Recall in Part C, we used the function *replace* in the DPLL algorithm where $\mathit{replace}(x, Q, P)$ rewrites formula P such that any occurrences of variable x are replaced with formula Q. This is more generally called *substitution*.

From now on we will use a more compact syntax for substitution written

$$P[t/x]$$

which means: *replace variable x with the term t in formula P* (akin to $\mathit{replace}(x, t, P)$). This term could be another variable or a concrete element of our universe.

Note that in predicate logic we have to be careful about free and bound variables. Thus, $P[t/x]$ means replace any *free* occurrences of x in P with object t. (One way to remember this notation is to observe that the letters used in the general form above are in alphabetical order: P then t then x to give $P[t/x]$ for replacing x with t in P). This is a common notation also used in the course textbook.

We must be careful to replace only the free occurrences of variables, that is, those variables which are not in the scope of a variable binding of the same name. For example, in the following we have a free x and a bound x, so substitution only affects the free x:

$$(\mathsf{P}(x) \land \forall x.\mathsf{P}(x))[t/x] = \mathsf{P}(t) \land \forall x.\mathsf{P}(x)$$

In general, it is best practice to give each bound variable a different name to all other free and bound names in a formula in order to avoid confusion.

1.3 The meaning of quantification

We can define the meaning of universal and existential quantification in terms of the propositional logic connectives.

Universal quantification Universal quantification essentially generalises conjunction. That is, if the objects in the universe over which we are quantifying are $a_0, a_1, \ldots, a_n \in \mathcal{U}$ then universal quantification of x over a formula P is equivalent to taking the repeated conjunction of P, substituting each object for x, i.e.

$$\forall x.P = P[a_0/x] \wedge P[a_1/x] \wedge \ldots \wedge P[a_n/x] \tag{5}$$

Thus, $\forall x.P$ means that we want P to be true for all the objects in the universe being used. Note that there may be an infinite number of such objects.

Existential quantification Whilst universal quantification generalises conjunction, existential quantification generalises disjunction. If existential quantification binds a variable ranging over objects $a_0, a_1, \ldots, a_n \in \mathcal{U}$ then:

$$\exists x.P = P[a_0/x] \vee P[a_1/x] \vee \ldots \vee P[a_n/x] \tag{6}$$

Thus, existential quantification is equivalent to the repeated disjunction of the formula P with each object in the universe replacing x.

1.4 Defining models/universes

First-order logic can be instantiated for particular concrete tasks by defining a universe $\mathcal{U}$ (a set of elements) and any relations, functions, and predicates over this universe.

For example, we could define $\mathcal{U} = \{\mathsf{cat}, \mathsf{dog}, \mathsf{ant}, \mathsf{chair}\}$ meaning that when we write quantified formulas like $\forall x.P$ (for some formula P) then x refers to any of the things in $\mathcal{U}$ (i.e., $x \in \mathcal{U}$). We could then concretely define some functions and predicates. For example, let's define a function legs which maps from $\mathcal{U}$ to $\mathbb{N}$ (i.e., $\mathsf{legs} : \mathcal{U} \to \mathbb{N}$) as:

$$\mathsf{legs}(\mathsf{cat}) = 4 \qquad \mathsf{legs}(\mathsf{dog}) = 4 \qquad \mathsf{legs}(\mathsf{ant}) = 6 \qquad \mathsf{legs}(\mathsf{chair}) = 4$$

We can define predicates by listing all their true instances. For example, mammal classifies some members of $\mathcal{U}$, defined via a proposition that lists all the true instances as a conjunction:

$$\mathsf{mammal}(\mathsf{cat}) \ \wedge \ \mathsf{mammal}(\mathsf{dog})$$

Let's consider some true formulas in this instantiation of first-order logic:

$\vdash \forall x.\mathsf{mammal}(x) \to (\mathsf{legs}(x) = 4)$ (*every mammal has four legs*)
$\vdash \exists x.\mathsf{legs}(x) < 4$ (*there is something with less than four legs*)

We have also employed two relations over $\mathbb{N}$ here:[6] equality $=$ and less-than $<$.

A false proposition in this instantiation is:

$$\nvdash \forall u.(\mathsf{legs}(u) = 4) \to \mathsf{mammal}(u) \quad (\textit{everything with four legs is a mammal})$$

This is false because u could be chair (making the premise of the implication true) but $\mathsf{mammal}(\mathsf{chair})$ is false.

[6] Strictly speaking, we are therefore using first-order logic where the universe contains our set $\{\mathsf{cat}, \mathsf{dog}, \mathsf{ant}, \mathsf{chair}\}$ and $\mathbb{N}$, i.e., $\mathcal{U} = \{\mathsf{cat}, \mathsf{dog}, \mathsf{ant}, \mathsf{chair}\} \cup \mathbb{N}$, and our function legs is partial, defined only for a part of the universe.

2 Equational reasoning

As in propositional logic, there are equations (logical equivalences) between particular first-order formulas. These can be used to rearrange and simplify formulas. This sections shows these equations, some of which are proved in the next section via natural deduction.

Two key equations show that universal and existential quantification are *dual*:

$$\forall x.\neg P \equiv \neg\exists x.P \qquad \neg\forall x.P \equiv \exists x.\neg P \tag{7}$$

The order of repeated quantifications is irrelevant as shown by the following equalities:

$$\forall x.\forall y.P \equiv \forall y.\forall x.P \qquad \exists x.\exists y.P \equiv \exists y.\exists x.P \tag{8}$$

Note however that these equalities are only for quantifications that are of the same kind; $\forall x.\exists y.P$ is not equivalent to $\exists y.\forall x.P$. The rest of the equations capture interaction between quantification and the other propositional connectives:

$$(\exists x.P) \vee (\exists x.Q) \equiv \exists x.(P \vee Q) \tag{9}$$

$$(\forall x.P) \wedge (\forall x.Q) \equiv \forall x.(P \wedge Q) \tag{10}$$

$$P \wedge (\exists x.Q) \equiv \exists x.(P \wedge Q) \quad \textit{when } x \textit{ is not free in } P \tag{11}$$

$$P \vee (\forall x.Q) \equiv \forall x.(P \vee Q) \quad \textit{when } x \textit{ is not free in } P \tag{12}$$

Example 14. We can now go back to the example from the introduction: that *not all birds can fly*. We formulated this sentence as both $\neg(\forall x.\mathsf{B}(x) \to \mathsf{F}(x))$ and $\exists x.\mathsf{B}(x) \wedge \neg\mathsf{F}(x)$. We can show these two statements are equivalent by algebraic reasoning:

$$\begin{array}{lll} & \neg(\forall x.\mathsf{B}(x) \to \mathsf{F}(x)) & \{\textit{by } (7)\} \\ \equiv & \exists x.\, \neg(\mathsf{B}(x) \to \mathsf{F}(x)) & \{P \to Q \equiv \neg P \vee Q\} \\ \equiv & \exists x.\, \neg(\neg\mathsf{B}(x) \vee \mathsf{F}(x)) & \{\textit{De Morgan's}\} \\ \equiv & \exists x.\, \neg\neg\mathsf{B}(x) \wedge \neg\mathsf{F}(x) & \{\textit{Double negation elim.}\} \\ \equiv & \exists x.\, \mathsf{B}(x) \wedge \neg\mathsf{F}(x) & \square \end{array}$$

Note that the actual universe and the definition of B and F is irrelevant to this proof; we did not rely on their definition but just the general properties of first-order logic.

Exercise 2.1. Prove via equational reasoning that:

$$\forall x.\mathsf{mammal}(x) \to (\mathsf{legs}(x) = 4) \;\equiv\; \neg\exists x.\mathsf{mammal}(x) \wedge \mathsf{legs}(x) \neq 4$$

Aide-mémoire

Write down any terms, symbols, concepts, techniques, etc. that you don't yet understand or would like to grasp better. You can cross these out (and move them to the right-hand column) as you make progress: it might be 5 minutes after you first write the item down here, or it might be 5 minutes before the exam. This will help you to keep a record of the things you need to work on and to understand what you need to know to make progress.

Things I don't understand yet	Things I need to get better at	Things I'm getting the hang of