Mathematical Techniques for Computer Science COMP11120

– Formal Logic Systems –

Renate Schmidt
Renate.Schmidt@manchester.ac.uk

Course webpage
http://www.cs.manchester.ac.uk/pgt/2015/COMP11120/

October 9, 2016

Contents

Practicalities

Lecture notes. These lecture notes will be handed out week by week and will be available for download from Blackboard. The lecture notes and the slides comprise the formal content of this part of the course unit. Recommended optional reading is mentioned below.

These notes have been considerably restructured, updated and extended from notes of Andrea Schalk from the 2014–5 delivery of the course, which in turn were based on notes of Graham Gough.

Please note: The lecture notes are updated every year and as such the new sections are less thoroughly proofread than the others. I will therefore be most grateful for corrections (of both typographical errors and more serious errors) as well as suggestions.

Slides and visualiser notes. I normally deposit my slides on Blackboard before each lecture, and will then update them afterwards with the annotated versions and also post copies of the visualiser notes.

Lectures. This part of the course consists of 6 lectures delivered over 3 weeks. The lectures will explain the material more informally with emphasis on key concepts, examples and exercises. The lecture notes and lectures are complementary. Attendance of the lectures is therefore important; it would be a mistake to just rely on the notes.

Assessed exercises. There will be 3 assessed exercise sheets. These will be marked as usual in the weekly examples classes. The assessed exercises are included with the lecture notes. As the solutions are published after the last examples class each week, no extensions can be given. Please refer to Andrea's notes on what to do in case of genuine mitigating circumstances.

Examples classes. The format of the examples classes will be unchanged. Each week there will be 5 exercises that are assessed, 3 core exercises that are regarded as basic, and 2 extensional, generally harder exercises.

Blackboard and course website. All material relevant to the course are available from Blackboard and the course website, though I generally make all my material available via Blackboard. We also use Blackboard for announcements and the discussion forum. Therefore, make sure you refer to Blackboard on a weekly basis. If you are not already subscribed to the discussion forum, do subscribe to it.

Feedback, questions and suggestions. Your feedback is key to the success of the course and is always welcome. Do please raise any questions or concerns as soon as possible. You can contact me by email (`Renate.Schmidt@manchester.ac.uk`) or in person, by speaking to me after the lecture or in the examples class, or finding me in my office (Rm 2.42 in Kilburn Building). Please also feel free to post any feedback, issues and questions to the discussion forum in Blackboard.

Additional reading

The following books on the reading list include chapters on logic, which you may wish to consult for additional explanations, more examples and exercises.

Garnier, Rowan, and Taylor, John (2002). *Discrete Mathematics for New Technology*, IOP. Chapter 1 and 2 on Logic and Mathematical proof. Very accessibly written.

Truss, J. K. (1999). *Discrete Mathematics for Computer Scientists*, Addison-Wesley. Chapter 2 discussing propositional lgoc, transformation to normal forms and first-order logic. More dense and advanced.

Epp, Susanna S. (2011). *Discrete Mathematics with Applications*, Brooks/Cole. Chapter 1 and 2 on logical statements in propositional logic and quantified logical statements (propositional and first-order logic). Very accessible. Available as online book.

Jordan, D. W. and P. Smith (2008). *Mathematical Techniques: An Introduction for the Engineering, Physical, and Mathematical Sciences*, Oxford University Press. Contains only a short chapter on logic, Chapter 36, which is focussed on logic gates and switching functions; goes beyond what we cover on this topic.

Several copies of each book are available from the main library but also the Resources Centre Library in the School (ask Richard Ward in SSO, LF21). Some of these books are available as online books; where this is the case this is indicated above. Many other excellent textbooks exist and can be found the library; logic and formal systems is standard material in Computer Science and Mathematics. The internet is also a useful additional resource (Wikipedia!).

Be aware though, there are variations in the presentation and definitions between different sources and books, and also to those we use.

Pre-requisite knowledge

As pre-requisite knowledge we assume basic concepts of sets, functions and relations, which is summarised in Chapter 0.

Chapter 3

Formal Logic Systems

The previous chapter (Chapter 2) concentrates on statements as they are commonly made in mathematics, using key phrases from the English language, and how to prove such statements.

Much of the discipline of mathematical logic is concerned with *formal systems*, which use a language of particular symbols to mimic, and formalise, this reasoning. Variations and extensions of such systems are often used in computer science to make precise statements.

This chapter introduces and studies formal systems and logic as general tools used to reason about true and false statements, and to test the correctness of arguments. The general aim is to understand the basics of using a formal logical system. In particular, the aim will be to discuss and understand the standard propositional logic connectives, to understand equivalence of logical formulas. We will discuss the benefits of normal forms and ways of convert logical expressions into conjunctive and disjunctive normal form. For reasoning about structured information quantifiers are needed and we will discuss the standard first-order logic connectives, including universal and existential quantification, and the role they play in making precise statements.

These tools are relevant for computer science, mathematics and many real world applications. For example, in programming languages a number of the fundamental laws that are derived in Section 3.3 have been implemented, giving another reason why this material is important from a computer science perspective.

3.1 Propositional logic

We first outline a comparatively simple system known as **propositional logic**, or **classical logic**. The basic entities in propositional logic and logical arguments are known as **propositions**, also referred to as **propositional formulas**. Propositions are built using **connectives**, which serve to connect propositions in order to form complex propositions.

Informally, **propositions** are statements that are either **true** or **false**, but not both. Examples of propositions are the following, some of which are true and others are false.

- 2 divides 4

- $225 = 15^2$
- 2 is an odd number
- $2 + 2 = 5$
- Grass is green
- It is raining
- I use an umbrella

Propositions can be connected with *not*, *and*, *or*, *if-then* and *iff* (if and only if) to form complex propositions, which we have already seen in the previous Chapter. Examples of complex propositions are:

- Grass is green and 2 is not an odd number.
- If it is raining then I use an umbrella.

Complex propositions are referred to as **compound propositions** and are defined to be propositions built from atomic propositions and connectives. **Atomic propositions** are propositions which can't be further broken down, such as '2 divides 4' and 'The grass is green'.

3.1.1 Propositional formulas

In order to ensure that our propositions are interesting, and to get started, we have to allow the language of propositional logic to contain **propositional variables**. We therefore assume that there is a list of propositional variables, $P, Q, R, \ldots$ or $P_1, P_2, \ldots$, and so on.

The logical notation for the connectives *not*, *and*, *or*, etc are

$\neg$	not, negation		
$\wedge$	and, conjunction	$\vee$	or, disjunction
$\rightarrow$	implies, implication	$\leftrightarrow$	iff, double implication

These are referred to as **propositional connectives**, or **propositional logic operators**.

It is also permitted to use brackets:

$$(\qquad)\qquad [\qquad]$$

The use of the symbols of the language of propositional logic and their intuitive meaning is the same as in mathematics (and the previous chapter):

$$\begin{array}{rcl} \neg P \text{ is true} & \text{iff} & P \text{ is false} \\ P \wedge Q \text{ is true} & \text{iff} & \text{both } P \text{ and } Q \text{ are true} \\ P \vee Q \text{ is true} & \text{iff} & \text{at least one of } P \text{ or } Q \text{ are true} \\ P \rightarrow Q \text{ is true} & \text{iff} & Q \text{ is true, when } P \text{ is true;} \\ & & \text{when } P \text{ is false we cannot infer anything about } Q \end{array}$$

Moreover we said: $P \leftrightarrow Q$ can be regarded as the abbreviation $(P \rightarrow Q) \wedge (Q \rightarrow P)$.

The formal definition of a **proposition**, which we prefer to call **propositional formula**, is as follows.

Definition 1 (Propositional formulas).

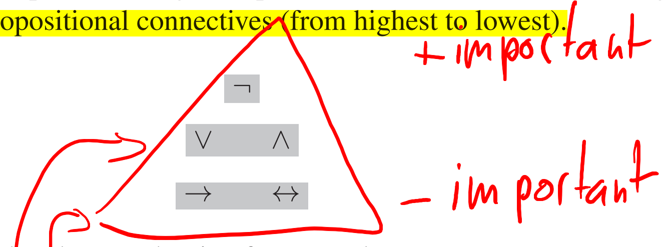
- If P is a propositional variable then P is a propositional formula (called **atomic formula**, or simply **atom**).
- If A is a propositional formula, then $(\neg A)$ is a propositional formula.
- If A and B are propositional formulas, then

$$(A \wedge B), \quad (A \vee B), \quad (A \to B) \quad \text{and} \quad (A \leftrightarrow B)$$

are propositional formulas.

This is an example of an *inductive definition*. We study such definitions formally in Semester 2 when we study recursion.

Note the brackets that are part of the definition of propositional formulas. This leads to far too many brackets in formulas. The following notational convention is therefore assumed in propositional logic. In particular, we assume the following **binding precedence** for propositional connectives (from highest to lowest).



This allows us to omit brackets, and write, for example,

$$P \underset{+}{\wedge} Q \underset{-}{\to} R \quad \text{instead of} \quad ((P \wedge Q) \to R)$$

Omitting brackets is useful, but not essential. Brackets help make it absolutely clear what we mean. So when in doubt about the binding, feel free to use more brackets than the notational convention would force you to do.

Definition 2. Suppose A is propositional formula. A **(propositional) subformula** of A is any substring of the formula A, that is a propositional formula according to the formal definition above, taking also into consideration the convention about the use of brackets.

Note that A is also a substring of A (and a propositional formula), so A is a subformula of A. This is a somewhat informal definition; in Semester 2 we will see how to give a fully formal inductive definition of the notion of a subformula of another formula.

3.1.2 Boolean semantics

All that is defined above is purely syntactic, that is, we are manipulating symbols. We may have some informal understanding of the intended intuitive meaning to propositional formulas, but what is needed is a formal definition of the semantics of propositional formulas. Semantics must be defined without ambiguity, and in such a way that the validity of arguments can be formally checked and verified.

The most simple and most widely used semantics of propositional formulas is **Boolean semantics**, which we describe in this section and then define formally in the next section.

A propositional formula by itself is not true or false. It becomes so only after the propositional variables have been replaced with concrete propositions, or with truth values. We therefore have to consider instantiations of the propositional variables and the effect of the connectives on truth values.

In the Boolean semantics of propositional logic there are only two **truth values**:

$$\mathbf{1} \quad \text{and} \quad \mathbf{0}.$$

You may think of $\mathbf{1}$ as denoting 'true', and $\mathbf{0}$ as denoting 'false'.[1]

The effect of the connectives on truth values is summarised by the following **standard truth tables**.

P	$\neg P$
$\mathbf{1}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{1}$

P	Q	$P \wedge Q$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{1}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$

P	Q	$P \vee Q$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$

P	Q	$P \to Q$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$

P	Q	$P \leftrightarrow Q$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{1}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$

The standard truth tables correspond to the natural reading of the connectives. It can be seen that:

$$\begin{array}{rcl}
\neg P \text{ is true} & \text{iff} & P \text{ is false} \\
P \wedge Q \text{ is true} & \text{iff} & \text{both } P \text{ and } Q \text{ are true} \\
P \vee Q \text{ is true} & \text{iff} & \text{at least one of } P \text{ or } Q \text{ are true} \\
P \to Q \text{ is false} & \text{iff} & P \text{ is true and } Q \text{ is false} \\
P \leftrightarrow Q \text{ is true} & \text{iff} & \text{either both } P \text{ and } Q \text{ are true, or both are false}
\end{array}$$

Certainly from a common-sense point of view these make sense: If I have two true statements A and B then the statement we obtain by connecting these with 'and' is true, but if one of them is false then the combined statement should be false. You may want to go back and read the text for the key phrases from Section 2.2 to compare what is said there with the above definitions for these operations.

[1] You sometimes find people using different names for the elements of this set, for example $\{\mathsf{T}, \mathsf{F}\}$ or $\{\mathsf{t}, \mathsf{f}\}$ or $\{\mathsf{tt}, \mathsf{ff}\}$ (COMP11212 on Fundamentals of Computation). You should have no difficulty translating between these.

Remark. The interpretation of $\to$ can be a bit puzzling, for example, if propositions are involved that are false in the real world. Consider

If the moon is made of cheese then $2 + 2 = 5$.

According to the defined semantics of $\to$ this propositional formula is true. However, we have no problems with accepting the definition of the semantics of implication for a statement such as

If it is raining then I use an umbrella.

In the first example both propositions involved are false (because non-sensical) and the result is true. If we were to let the result be false, and also the results be false, if one is false and the other true, then the interpretation of $\to$ is the same as $\wedge$, which is not intuitive. The way implication is defined is the best one available; it works!

To avoid puzzlement just think of $P \to Q$ as an abbreviation for $\neg P \vee Q$. (End of remark)

Now, how do we interpret a complex propositional formula such as the following?

$$(*) \qquad ((P \vee Q) \wedge \neg P) \vee \neg Q$$

As indicated above, we have to work *relative to valuations*[2] *for the propositional variables*. The formula contains two variables P and Q. In this example we can take as a valuation v a function

$$\{P, Q\} \longrightarrow \{\mathbf{0}, \mathbf{1}\}$$

that maps the propositional variables P and Q to $\mathbf{1}$ and $\mathbf{0}$. Our interpretation is relative to a valuation, and to really understand the given propositional formula we have to consider *all possible valuations* of the list of (at least the) propositional variables occurring in the formula.

In the Boolean semantics, the interpretation of a propositional formula is given by the truth table of the formula.[3] For example, the truth table for the formula $(*)$ above is:

P	Q	$((P \vee Q) \wedge \neg P) \vee \neg Q$
1	**1**	**0**
1	**0**	**1**
0	**1**	**1**
0	**0**	**1**

We obtain this truth table by filling in a longer version of the table that helps make the calculations faster.[4]

P	Q	$P \vee Q$	$(P \vee Q) \wedge \neg P$	$((P \vee Q) \wedge \neg P) \vee \neg Q$
1	**1**	**1**	**0**	**0**
1	**0**	**1**	**0**	**1**
0	**1**	**1**	**1**	**1**
0	**0**	**0**	**0**	**1**

[2] Some people call these 'interpretations' or 'truth assignments' for the Boolean model.

[3] Compare these with the rules for logical operators on page 128 of *Java: Just in Time*, and with those in the first handout for COMP12111—they all talk about the same operations, only that implication is omitted there.

[4] This is the same idea as the truth tables in Chapter 8 of *Java: Just in Time*.

You may wish to also include columns for the interpretations of the subformulas $\neg P$ and $\neg Q$ in order to help avoid mistakes.

Exercises

Exercise 1.

Use our notational convention and remove as many brackets as possible from these formulas.

(i) $(((P \to R) \wedge (((\neg Q) \to R)) \to R) \to ((P \vee R) \to R))$

(ii) $((\neg P) \to ((\neg P) \vee Q))$

Exercise 2.

List all subformulas of this propositional formula.

$$P \wedge \neg P \to (Q \vee R \to (R \to \neg P))$$

Exercise 3.

Construct the truth tables for the following propositional formulas, and show the truth tables of each of the pairs of formulas are the same (i.e., the truth value columns for the given formulas are the same).

(i) $P \vee Q$ and $\neg P \to Q$

(ii) $P \wedge Q$ and $\neg(P \to \neg Q)$

(iii) $P \leftrightarrow Q$ and $(P \to Q) \wedge (Q \to P)$

(iv) $\neg(P \leftrightarrow Q)$ and $(P \vee Q) \wedge (\neg P \vee \neg Q)$

(v) $P \wedge (P \vee Q)$ and $P \vee (P \wedge Q)$

(vi) $P \wedge (Q \vee R)$ and $(P \wedge Q) \vee (P \wedge R)$

(vii) $P \vee (Q \wedge R)$ and $(P \vee Q) \wedge (P \vee R)$

Exercise 4.

Write down a propositional formula with the propositional variables P, Q, R for which the truth value column in its truth table is:

(i) **0 0 0 0 1 0 0 0**

(ii) **1 0 0 0 1 0 1 0**

(iii) **0 0 0 0 0 0 0 0**

Explain your answers.

3.2 More on semantics

A formal, general definition of semantics

There are many ways of defining the semantics of propositional formulas, which are relevant to different subareas of computer science and mathematics. We start of by describing the general setting for defining each of the different semantics, and how Boolean semantics fits into it. Subsequently, we give two more examples of semantics of propositional formulas relevant to computer science.

Because our propositional formulas are built using propositional variables we have to worry about how to interpret propositional variables. We cannot make any assumptions about the meaning of these variables, and so we employ a trick that is also found in the formal semantics of programming language.

Intrinsically, a propositional variable has *no* meaning. The way we assign meaning to a propositional formula is to consider *all* the possible meanings a variable could have, one at a time. This leads to the concept of a **valuation** (i.e., an interpretation function).

Definition 3. Assume S is the set of values that propositional formulas can take. Further, assume $P_1, P_2, \ldots, P_n$ is a finite list of variables.

- A **valuation** v for these variables is a function

$$\mathrm{v} : \{P_1, P_2, \ldots, P_n\} \longrightarrow S$$

  that assigns a value from S to each propositional variable in the list.

- We also have to assign interpretations in S to the connectives. This is done in a formal way which depends on the semantics.

The interpretation of a formula is then defined as follows:

Definition 4. Suppose A is any propositional formula that contains the propositional variables $P_1, P_2, \ldots, P_n$, and suppose v is a valuation for these propositional variables. The **interpretation** of A **relative to valuation** v is the value of the formula obtained by

- replacing each P_i by v P_i, and
- evaluating the assignment to each connective.

If we look at *all* possible valuations, that is, at all the functions with the source and target given above (Definition 3), then we look at all the possible meanings the propositional formula could have in our semantics. This is why we assign a value to each propositional formula *relative to a valuation*. We may state properties of this semantics by quantifying over all possible valuations. This process is described in more detail for each of the semantics we discuss below. (Some Computer scientists and Mathematicians refer to these different semantics as models.)

3.2.1 Semantics 1: Boolean semantics

In the Boolean semantics, the values that propositional formulas can take are $S = \{\mathbf{1}, \mathbf{0}\}$. S is known as the **two element Boolean algebra** $\mathbb{B}$. This means

- A **Boolean valuation** v is a function

  $$\mathrm{v} : \{P_1, P_2, \ldots, P_n\} \longrightarrow \{\mathbf{1}, \mathbf{0}\}$$

  that assigns $\mathbf{1}$ or $\mathbf{0}$ to the list $P_1, P_2, \ldots, P_n$ of propositional variables.

- The **Boolean interpretation** of a formula A relative to v is the value of the formula obtained by replacing each P_i by $\mathrm{v}\,P_i$ and calculating the truth value of A in accordance with the standard truth tables for the connectives.

The Boolean interpretation of a propositional formula is also called its **truth value**.

Interpretation of a propositional formula

Given the list P, Q of propositional variables, assume we have a valuation v that takes its values in $\{\mathbf{0}, \mathbf{1}\}$. The interpretation of our running example $(*)$ from the previous section, i.e.,

$$(*) \qquad ((P \vee Q) \wedge \neg P) \vee \neg Q,$$

relative to v is then given by the truth value of

$$((\mathrm{v}\,P \vee \mathrm{v}\,Q) \wedge \neg\,\mathrm{v}\,P) \vee \neg\,\mathrm{v}\,Q.$$

Assume, for example, that we have a valuation v which maps

$$P \text{ to } \mathbf{0} \qquad \text{and} \qquad Q \text{ to } \mathbf{1}.$$

Then the value of $((P \vee Q) \wedge \neg P) \vee \neg Q$ relative to this v is

$$\begin{aligned}
&((\mathrm{v}\,P \vee \mathrm{v}\,Q) \wedge \neg\,\mathrm{v}\,P) \vee \neg\,\mathrm{v}\,Q \\
&\quad = ((\mathbf{0} \vee \mathbf{1}) \wedge \neg\mathbf{0}) \vee \neg\mathbf{1} && \text{def v} \\
&\quad = ((\mathbf{0} \vee \mathbf{1}) \wedge \mathbf{1}) \vee \mathbf{0} && \text{def } \neg \\
&\quad = (\mathbf{1} \wedge \mathbf{1}) \vee \mathbf{0} && \text{def } \vee \\
&\quad = \mathbf{1} \vee \mathbf{0} && \text{def } \wedge \\
&\quad = \mathbf{1} && \text{def } \vee
\end{aligned}$$

$\mathbf{1}$ therefore is the interpretation of the formula $(*)$ relative to the given valuation v.

The interpretation of the formula $(*)$ relative to the given v can also be read off from the truth table of $(*)$—namely, in the third row.

P	Q	$((P \vee Q) \wedge \neg P) \vee \neg Q$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{0}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$

This illustrates:

- Each line in a truth table gives the interpretation of a formula for the given truth values of its propositional variables.

It is also immediate that:

- Truth tables display all possible interpretations of a formula.

Computing the truth value (interpretation) of a formula for a given valuation is a *model checking* task. In software engineering, for safety critical applications, model checking is used to verify the correctness of software against their high-level specifications. Model checking is also used to verify the correctness of security protocols and other kinds of protocols.

Tautology, satisfiability, contradiction

Definition 5. A propositional formula A is called a **tautology**, if the interpretation of A is $\mathbf{1}$, for *all possible* valuations v of the propositional variables occurring in A.

It follows from the definition of a tautology that truth tables can be used to show that a formula is a tautology. (How?) For example, our formula $((P \vee Q) \wedge \neg P) \vee \neg Q$ is not a tautology. We can see this by simply looking at its truth table above. Make sure you know why. Exercise: By making minimal changes to this formula find an example of a tautology.

Definition 6. A propositional formula A is **satisfiable**, if the interpretation of A is $\mathbf{1}$, for *some* valuation v. Otherwise, it is a **contradiction** (we also say it is **unsatisfiable** in this case).

For example, $((P \vee Q) \wedge \neg P) \vee \neg Q$ is satisfiable.

Truth tables can also be used to show whether a formula is satisfiable. (How?) However, the size of a truth tables is exponential in the number of variables occurring in a formula. It can be more efficient to calculate the interpretation of a formula for different valuations until one is found which returns the truth value $\mathbf{1}$ without constructing the complete truth table.

3.2.2 Semantics 2: Power set semantics

Reducing the interpretation of every propositional formula to just one value, $\mathbf{1}$ or $\mathbf{0}$, is somewhat restrictive, and not very subtle.

In the power set semantics, the values that propositional formulas can take are subsets of X, where X is an arbitrary (but fixed) non-empty set. That is, valuations are defined over $S = \mathcal{P}X$, the power set of X. In particular:

Definition 7. In the **power set semantics**:

- A **valuation** v is a function

$$\mathrm{v} : \{P_1, P_2, \ldots, P_n\} \longrightarrow \mathcal{P}X$$

  that assigns subsets of X to the list $P_1, P_2, \ldots, P_n$ of propositional variables.

- To obtain the **power set interpretation** of a formula A relative to v

for conjunction $\wedge$	use intersection $\cap$
for disjunction $\vee$	use union $\cup$
for negation $\neg$	use set complement $X \setminus -$
for $B \to B'$	use $(X \setminus S_B) \cup S_{B'}$,

where S_B is the interpretation of formula B relative to v and $S_{B'}$ is the interpretation of formula B' relative to v.

$S = \mathcal{P}X$ with the set operations $\cap$, $\cup$ and set complement is called the **power set algebra**.

This means, in the power set semantics, the interpretation of a propositional formula A relative to a given valuation v over a set X can be calculated as follows.

- In the formula A, replace every occurrence of P_i by $\mathrm{v}\, P_i$.
- Replace the connectives according to the instructions given in the second bullet point in Definition 7 above.
- Finally, calculate the value of the resulting expression in $\mathcal{P}X$. This will return a subset of X.

We go back to the example $(*)$ from earlier, namely $((P \vee Q) \wedge \neg P) \vee \neg Q$. Assume that $X = \{a, b, c\}$ and that the valuation v maps

P to $\{a, b\}$ and Q to $\{b, c\}$.

Then the value of $((P \vee Q) \wedge \neg P) \vee \neg Q$ relative to v is

$$\begin{aligned}
&((\mathrm{v}\, P \cup \mathrm{v}\, Q) \cap (X \setminus \mathrm{v}\, P)) \cup (X \setminus \mathrm{v}\, Q) \\
&\quad = ((\{a,b\} \cup \{b,c\}) \cap (X \setminus \{a,b\})) && \text{def v} \\
&\qquad \cup (X \setminus \{b,c\}) \\
&\quad = (\{a,b,c\} \cap \{c\}) \cup \{a\} && \text{def } \cup, \setminus \\
&\quad = \{c\} \cup \{a\} && \text{def } \cap \\
&\quad = \{a,c\}. && \text{def } \cup
\end{aligned}$$

In the power set semantics, a **tautology** is a propositional formula which is interpreted as X, for every valuation and all possible non-empty sets X. **Satisfiability** becomes the existence of a valuation such that the interpretation of the propositional formula in question is non-empty, for some non-empty set X. Otherwise, it is a **contradiction**.

Hence in the interpretation given we can see that $((P \vee Q) \wedge \neg P) \vee \neg Q$ is satisfiable, and we can immediately conclude it is neither a tautology nor a contradiction. We can make this conclusion after making calculations for just one valuation. This is in contrast to the Boolean semantics where we have to find at least one valuation for which the formulas evvaluate to **0**.

On the other hand in the power set semantics, there are significantly more valuations available, so checking all of them explicitly will take considerably more time.

The approach of explicitly computing the interpretations is completely unfeasible for checking whether a formula is a tautology or a contradiction, as there are uncountably many non-empty sets that we can take as X and therefore also uncountably many interpretations of a formula![5]

However, we can use set-theoretic arguments and general properties of sets and operations of sets. For example, to show that

$$P \vee \neg P$$

is a tautology we can argue as follows. Let X be an arbitrary non-empty set and suppose the valuation of P is $\mathrm{v}\, P = S_P$, where S_P is a subset of X. In order to show that the above formula is a tautology we show that its interpretation is X, i.e.,

$$\mathrm{v}\, P \cup (X \setminus \mathrm{v}\, P) = S_P \cup (X \setminus S_P) = X$$

We can do this by showing for any $x \in X$

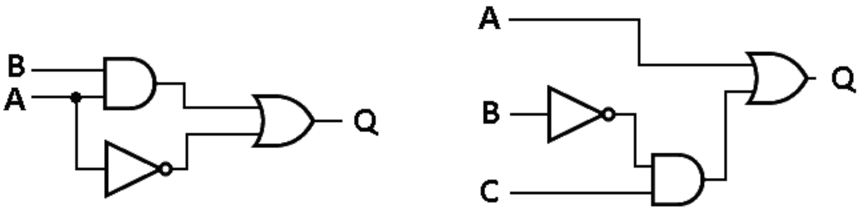
$$x \in S_P \cup (X \setminus S_P) \quad \text{iff} \quad x \in X$$

The proof of this is left as exercise. (Actually the right to left direction is sufficient, because the other direction is obvious. Try to work out why.)

3.2.3 Semantics 3: Logic Gates

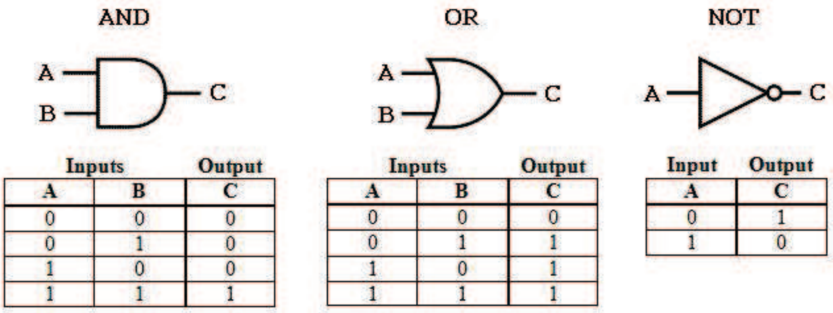
We may build yet another model for our formal system, using logic gates.

Logic gates can be used to build electronic switching circuits such as these:



Source: `http://www.nandland.com/articles/boolean-algebra-using-look-up-tables-lut.html`

These circuits are built by connecting three types of logic gates: AND gates, OR gates and NOT gates. The effects of logical gates are summarised by these tables, where **1** represents a high voltage, and **0** represents a low voltage.



AND

Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

OR

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

NOT

Input	Output
A	C
0	1
1	0

Source: `http://newstudent.groups.et.byu.net/Labs/Logic\%20Gates/LogicGates.html`

[5] At this stage think of ‘uncountably many’ as meaning ‘infinitely many’. We say a set is *uncountable* if it is infinite in size and there is no systematic way of listing the elements of the set.

In the logic circuit semantics, each propositional variable for a given propositional formula becomes an *input wire* for our circuit. We use the following to connect these wires according to the connectives in the propositional formula:

for conjunction $\wedge$	use	an AND gate
for disjunction $\vee$	use	an OR gate
for negation $\neg$	use	a NOT gate
for $B \to B'$	use	an OR gate where the first input has been negated

The resulting circuit has one output wire. A *valuation* means deciding, for each input wire, whether to put a voltage onto that input wire or not. Given a valuation we get two possibilities for the output wire: Either it carries a voltage, or it doesn't.

The **circuit interpretation** of a propositional formula is the resulting circuit. The notion of tautology is then adjusted to this model by demanding that for every valuation (that is, for every way of putting a voltage, or not, onto the input wires) the output wire carries a voltage, and satisfiability means that there is at least one way of assigning voltages to the input wires such that the output wire carries a voltage.

Applications

Instead of thinking of a model as saying something about the formal system, one could think of the formal system making statements about the intended model. The fact that there are so many models available is one of the reasons why the language of logic is applicable in so many ways.

Those of you taking COMP12111 have met logic as a description of what happens in an electronic circuit. Propositional logic can be used to test if two circuits are equivalent.

The conditional statements in Java use logic connectives, just with a different syntax so that they may be typed on a standard keyboard. You find more about these below. We have seen there is an obvious way of taking our 'key phrases' and turning them into logical connectives, and so we may think of a formal system like ours as talking about definitions and proofs from mathematics. Additionally, via the power set semantics we can use our formal system to reason about of sets (of any kind) and properties. The power set semantics can be seen to provide the foundation for ontology-based knowledge processing technologies which will be covered in year four (in COMP60411 on Modelling Data on the Web and COMP62342 on Ontology Engineering and the Semantic Web).

But statements built from logical connectives are used in many other areas as well, for example in database queries, or to describe the result such a query ought to have. The key phrases, or the corresponding logical connectives, allow us to make precise statements about a great variety of different topics. Because they appear in so many places it is important to understand these connectives and their intended meanings, as well as their properties.

In general, whenever people wish to formally establish something such as the correctness of a circuit or a program relative to a given specification they translate the specification into a formal system, and construct a formal model of the circuit or program within that system, and then try to establish a formal proof using a mechanism supplied by the system. In the next section we talk about equivalent formulas and describe methods of testing if two formulas are equivalent. These methods can then also used to test equivalence of the corresponding logical circuits.

Exercises

Exercise 5.

If you have a propositional formula with n propositional variables, how many possible valuations over these variables are there?

Exercise 6.

Consider the following propositional formula

$$(P \vee Q) \to \neg(P \wedge Q).$$

(i) Calculate the Boolean interpretation of the formula relative to the valuation which maps P to **0** and Q to **1**.

(ii) Calculate the power set interpretation in $X = \{a, b, c\}$ of the formula relative to the valuation which maps P to $\{a, b, c\}$ and Q to $\{c\}$.

Exercise 7.

Determine which of the following propositional formulas are tautologies? Which are contradictions? Which are satisfiable? Use truth tables to explain your answer.

(i) $P \to (Q \to P)$

(ii) $P \to (\neg P \vee Q)$

(iii) $((P \to Q) \to Q) \to Q$

(iv) $((P \to Q) \to P) \to P$

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

(vi) $(((P \vee Q) \wedge (P \to R)) \wedge (Q \to R)) \to R$

Exercise 8.

Let A and B be any propositional formulas. Show the following in the power set semantics.

(i) $A \to (A \vee B)$ is a tautology.

(ii) $(A \wedge B) \to A$ is a tautology.

(iii) If A is a tautology then $\neg A$ is a contradiction.

Exercise 9.

Let A and B be any propositional formulas.

(i) Show that if A is a tautology and $A \to B$ is a tautology then B is a tautology.

(ii) Disprove (by giving a counter-example) that if $A \to B$ is a tautology and B is a tautology then A is a tautology.

Exercise 10.

Give an argument that the power set semantics subsumes the Boolean semantics. *Hint: Consider the power set for a one-element set.*

Exercise 11.

Assume that every valuation may assign only the values $\emptyset$ or X to a propositional variable. What does that mean for this model?

Optional Exercise 12.

In general, is there a way of relating an interpretation in the power set semantics for arbitrary X (and arbitrary valuations) to a Boolean interpretation?

COMP11120 Assessed Exercises marked in Week 5

Core Exercises marked this week

Exercise 3. Do three of the parts, one from (i)–(iii), one from (iv)–(v) and one from (vi)–(vii).

Exercise 6.

Exercise 7.

Extensional Exercises marked this week

Exercise 4.

Exercise 9.

Remember that

- the **deadline** is the beginning of the examples class, and that you have to be able to promptly answer questions by the TA, referring to your rough work as needed (so don't forget to bring your rough notes);
- you may only use concepts which are defined in these notes and those of Andrea (Chapter 0 establishes concepts for sets, functions, relations), and for every concept you do use you should find the definition in the notes and work with that;
- you should justify each step in your proofs;
- if you are stuck on an exercise move on to the next one after ten minutes, but write down why you got stuck so that you can explain that to the TA in the examples class.
- In order for you to be considered as having seriously tried an exercise you must have written down the number of relevant examples and definitions in the notes, and you must be able to tell the TA what you tried to apply those ideas to the given problem. If you didnt understand the examples/definitions you must be able to tell them where exactly your understanding failed.

Exercises you could do this week are those in this section and the previous section.