3.3 Equivalent formulas

In spoken language there are many equivalent ways of making statements, this is also the case for propositional logic (though not to the extreme as in natural language).

3.3.1 Semantic equivalence

Definition 8. We say two formulas A and B are **semantically equivalent**,[6] and write $A \equiv B$ in this case, iff their interpretation is the same, for any valuation v (of the propositional variables occurring in A and B).

In particular, in the Boolean semantics (which we use most often because it is the simplest), two formulas A and B are semantically equivalent if they have the same truth tables, using enough propositional variables in the tables to include all the variables appearing in either A or B.

Examples of semantically equivalent formulas are the following.

$P \equiv \neg\neg P$

P	$\neg P$	$\neg\neg P$
1	0	1
0	1	0

double negation / involution

$P \equiv P \wedge P$

P	$P \wedge P$
1	1
0	0

idempotence

$P \equiv P \vee P$

P	$P \vee P$
1	1
0	0

idempotence

These equivalences are fundamental as they describe properties of negation, conjunction and disjunction.

Here is a more complex example of a fundamental equivalence.

$P \to Q \equiv \neg P \vee Q$

P	Q	$P \to Q$	$\neg P$	$\neg P \vee Q$
1	1	1	0	1
1	0	0	0	0
0	1	1	1	1
0	0	1	1	1

Since the third and the last column are the same we know that $P \to Q$ and $\neg P \vee Q$ have the same interpretation for every possible combination of input values. As a consequence, no matter which value is assigned to P and Q in our interpretation for the given valuation, the two propositional formulas are assigned the same truth value. Hence, $P \to Q$ and $\neg P \vee Q$ are equivalent.

So when we are merely interested in the interpretation of propositional formulas in our interpretation then we may *remove* the connective $\to$ from consideration entirely, and instead use negation and conjunction.[7]

[6] We also say A and B are *logically equivalent* if $A \equiv B$.

[7] But note that if we are interested in a logical system for other reasons (such as the use of formulas for convenient, compact representation of information or the intuitive presentation of logical arguments) then we should not just remove connectives.

In exactly the same way we can show the equivalence of two propositional formulas, by constructing their truth tables and checking that the formulas have the same truth values for every possible valuation. This shows how we can use truth tables to show that two formulas are equivalent.

Exercise 3 already gave several examples of propositional formulas with the same truth value columns. These are all semantically equivalent.

The following is a useful list of **fundamental semantic equivalences**, including some we have already discussed.

Theorem 3.1 (Fundamental semantical equivalences)**.**

$\neg\neg P \equiv P$	*double negation / involution*
$P \wedge P \equiv P$	*idempotence*
$P \vee P \equiv P$	*idempotence*
$P \wedge Q \equiv Q \wedge P$	*commutative*
$P \vee Q \equiv Q \vee P$	*commutative*
$P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R \equiv P \wedge Q \wedge R$	*associative*
$P \vee (Q \vee R) \equiv (P \vee Q) \vee R \equiv P \vee Q \vee R$	*associative*
$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$	*distributive*
$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$	*distributive*
$P \wedge (P \vee Q) \equiv P$	*absorption*
$P \vee (P \wedge Q) \equiv P$	*absorption*
$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$	*De Morgan's law*
$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$	*De Morgan's law*
$P \to Q \equiv \neg P \vee Q$	
$P \to Q \equiv \neg(P \wedge \neg Q)$	
$P \to Q \equiv \neg Q \to \neg P$	*contrapositive*
$P \leftrightarrow Q \equiv (P \to Q) \wedge (Q \to P)$	
$P \leftrightarrow Q \equiv (\neg P \vee Q) \wedge (P \vee \neg Q)$	
$P \leftrightarrow Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q)$	

It is important to note that $A \equiv B$ is not the same as $A \leftrightarrow B$, though the two notions are related. Semantic equivalence is a property of formulas, and a meta-logical notion. The symbol $\equiv$ is *not* a logical connective; it is notation in the meta-language, whereas $\leftrightarrow$ is a symbol of the language of propositional logic (the object language). The connection between semantic equivalence and the double implication connective is:

Theorem 3.2. *$A \equiv B$ iff $A \leftrightarrow B$ is a tautology.*

Also, be careful *not* to confuse $\equiv$ with equality! Our symbol for equality is $=$. We have $P \vee Q \equiv P \vee Q \equiv Q \vee P$ and $P \vee Q = P \vee Q$, but $P \vee Q \neq Q \vee P$. Two 'things' are only equal when they are the same.

3.3.2 Substitution and equivalent replacement

Substitution for propositional variables

Substitution provides a formal mechanism to replace a propositional variable within a formula by another formula.

Let A be the propositional formula

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

If we want to **substitute** the formula $B = P \wedge Q$ for P in A then we need to put B in place of *every occurrence* of P in A. We get $B \vee \neg(Q \to B)$, or

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

In the following theorem we use the following notation. Let A and $C_1, \ldots, C_n$ denote propositional formulas. Writing $A(P_1, \ldots, P_n)$ means that the propositional variables $P_1, \ldots, P_n$ possibly occur in the formula A. The notation $A(C_1, \ldots, C_n)$ then refers to the formula that is obtained, when for each i with $1 \leq i \leq n$, the formula C_i has been substituted for every occurrence of P_i in A.

Theorem 3.3 (Substitution Theorem)**.** *Let A and B be any propositional formulas. If*

$$A(P_1, \ldots, P_n) \;\equiv\; B(P_1, \ldots, P_n)$$

and $C_1, \ldots, C_n$ are propositional formulas, then

$$A(C_1, \ldots, C_n) \;\equiv\; B(C_1, \ldots, C_n).$$

Proof. $A \equiv B$ means that for every valuation v the interpretation of A is the same as the interpretation of B. This is in particular true, whenever the valuations v $P_1, \ldots$ v P_n are the truth values of $C_1, \ldots, C_n$ in some interpretation. Therefore, $A(C_1, \ldots, C_n) \equiv B(C_1, \ldots, C_n)$. □

For example, from the law of double negation $P \equiv \neg\neg P$, we can conclude that

$$A \;\equiv\; \neg\neg A$$

for any propositional formula A. For example, $(P \to Q) \equiv \neg\neg(P \to Q)$.

Similarly, from De Morgan's law $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$, we can conclude

$$\neg(A \wedge B) \;\equiv\; (\neg A \vee \neg B)$$

for any propositional formulas A and B. For example,

$$\neg((P \vee \neg R) \wedge (P \to S)) \;\equiv\; (\neg(P \vee \neg R) \vee \neg(P \to S)).$$

In a similar way it follows from the Substitution Theorem that each of the fundamental equivalences in the list above (in Theorem 3.1) actually holds for any propositional formulas, not only for propositional variables.

Equivalent replacement

Besides substitution, equivalent replacement is an important operation to equivalently transform formulas. Substitution means putting one formula for *all* occurrences of a particular *propositional variable* (possibly repeatedly). **Replacement** means putting one formula for *one* occurrence of a *subformula* in another formula.

Theorem 3.4 (Equivalent Replacement Theorem)**.** *Let A and B be propositional formulas. Suppose*

$$A \equiv B$$

and A is a subformula of a propositional formula C_A. Then

$$C_A \equiv C_B,$$

where C_B is obtained by putting B for one occurrence of A in C_A.

We know $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$. By the Equivalent Replacement Theorem we get

$$\neg(P \wedge Q) \vee R \equiv (\neg P \vee \neg Q) \vee R.$$

Also

$$\begin{array}{lll} \neg(P \wedge Q) \vee \neg Q & & \\ \quad \equiv & (\neg P \vee \neg Q) \vee \neg Q & \text{by equiv. replacement and De Morgan} \\ \quad \equiv & \neg P \vee (\neg Q \vee \neg Q) & \text{by equiv. repl. and general associativity of } \vee \\ \quad \equiv & \neg P \vee \neg Q & \text{by equiv. repl. and } \neg Q \vee \neg Q \equiv \neg Q \end{array}$$

This shows $\neg(P \wedge Q) \vee \neg Q \equiv \neg P \vee \neg Q$ without using truth tables!

Starting with the given formula $\neg(P \wedge Q) \vee \neg Q$ we have replaced $\neg(P \wedge Q)$ with $\neg P \vee \neg Q$ to get $(\neg P \vee \neg Q) \vee \neg Q$, by using De Morgan's law which says $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ and the Equivalent Replacement Theorem. In the next step we have used general associativity of $\vee$ to replace $(\neg P \vee \neg Q) \vee \neg Q$ by $\neg P \vee (\neg Q \vee \neg Q)$. And so on.

The example illustrates that we can use the fundamental laws and the replacement theorem[8] to prove the equivalence of two formulas, namely, by equivalently transforming one into the other. In each step we have performed **equivalent replacement** using a fundamental equivalence. *The idea of equivalent replacement is to try and find a subformula that matches the left-hand side of a fundamental equivalence and replace it by the right-hand side of the equivalence. We may also use fundamental equivalences from right-to-left.*

The formula $\neg P \vee \neg Q$ is significantly simpler than the formula $\neg(P \wedge Q) \vee \neg Q$ we started with, which illustrates that the fundamental laws and the replacement theorem[9] provide a useful basis for simplifying propositional formulas, i.e., finding simpler looking equivalent formulas.

[8]and implicitly also the substitutivity theorem when we use generalised fundamental laws
[9]and implicitly also the substitutivity theorem

Note that many computer languages have propositional formulas: every language that has some kind of conditional 'if ... then' instructions uses them. The typical usage is

```
if ... then ... else ...
```

which can be expressed as a propositional formula.[10] If we know

```
prop formula 1 ≡ prop formula 2
```

then by the Theorem of Equivalent Replacement we have that

```
if <prop formula 1> then ... else ...
  ≡ if <prop formula 2> then ... else ...
```

This means that we may use the techniques learned in this section to rewrite the condition in our if statement into a more easily understandable, equivalent form. This makes it much easier to check whether one has captured the various cases in one's if statement as intended.[11]

3.3.3 The logical constants $\top$ and $\bot$

We have not yet exhausted the possibilities for simplifying formulas.

It is useful to extend the language of propositional logic with the **logical constants** (i.e., nullary connectives)

$$\top \quad \text{and} \quad \bot.$$

$\top$ and $\bot$ are referred to as '**truth**' and '**falsum**' (or top and bottom).

- The interpretation of $\top$ is **1**, for any valuation of the propositional variables.
- The interpretation of $\bot$ is **0**, for any valuation of the propositional variables.

This means the interpretation of $\top$ is always **1**, and the interpretation of $\bot$ is always **0**.

We may think of $\top$ as an abbreviation of $P \vee \neg P$, for some propositional variable P (since $P \vee \neg P \equiv \top$). Similarly, and $P \wedge \neg P \equiv \bot$). We may think of $\bot$ as an abbreviation of $P \wedge \neg P$, for some propositional variable P (since $P \wedge \neg P \equiv \bot$). We have that:

P	$\neg P$	$P \vee \neg P$	$P \wedge \neg P$
1	**0**	**1**	**0**
0	**1**	**1**	**0**

The notation $\top$ and $\bot$ for truth and falsehood are quite widely used, but other symbols may also appear in the literature, for example **1** or **T** or **t** for truth, and **0** or **F** or **f** for falsehood. In Java you have the reserved words `true` and `false`, which serve the same purpose.

Useful properties, and fundamental properties of $\top$ and $\bot$, are given in the next two theorems:

[10] Try to work out by which formula. Hint: Express 'if A then α else β' by a propositional formula.

[11] In *Java: Just in Time* in Chapter 8 you find truth tables as a way of evaluating propositional expressions—this is precisely the same idea as our Boolean interpretation. Our 'semantically equivalent' means 'equivalent' in that book.

Theorem 3.5.

(i) If A is a tautology then $A \equiv \top$.

(ii) If A is a contradiction then $A \equiv \bot$.

Theorem 3.6 (Fundamental semantic equivalences involving $\top$ and $\bot$).

$$
\begin{array}{rcll}
A \vee \neg A & \equiv & \top & \textit{excluded middle} \\
A \wedge \neg A & \equiv & \bot & \textit{contradiction} \\
A \to A & \equiv & \top & \\
A \leftrightarrow A & \equiv & \top & \\
\neg\top & \equiv & \bot & \\
\neg\bot & \equiv & \top & \\
A \vee \top & \equiv & \top \equiv \top \vee A & \\
A \wedge \top & \equiv & A \equiv \top \wedge A & \\
A \vee \bot & \equiv & A \equiv \bot \vee A & \\
A \wedge \bot & \equiv & \bot \equiv \bot \wedge A & \\
A \to \bot & \equiv & \neg A & \\
A \leftrightarrow \top & \equiv & A & \\
\bot \to A & \equiv & \top & \\
A \leftrightarrow \bot & \equiv & \neg A &
\end{array}
$$

These are especially useful for simplifying formulas.

Remark. There are other logic systems, e.g., intuitionistic logic, which are not truth functional, where interpretations are defined differently, because the semantics of formulas cannot be defined using truth tables. Intuitionistic logic is also a propositional system but not all of the fundamental equivalences hold; the law of excluded middle does not hold for intuitionistic logic. (End of remark)

3.3.4 Semantic equivalence in the other semantics

Recall, the definition of semantic equivalence of formulas. Let A and B denote two propositional formulas. We defined $A \equiv B$ iff the interpretation of A and B is the same, for any valuation v for the list of propositional variables occurring in A and B.

In the power set semantics, an interpretation is defined relative to the valuation v but also a non-empty set X and the interpretation of any formula for a v is a subset of X. This means, in the power set semantics $A \equiv B$ iff the sets assigned to A and B are the same for any valuation.

Thus, to show $A \equiv B$, we have to show that for any non-empty set X and any valuation (over X) for the propositional variables in A and B, the assignment to A is the same as the assignment to B. Suppose S_A is the assignment to A and S_B the assignment to B. We have to show

$$S_A = S_B.$$

In the definition of the power set semantics, we have already used the equivalence $A \to B \equiv \neg A \vee B$, namely in the *definition* of the interpretation of $\to$. Let the interpretation of A relative to valuation v be S_A, and that of B be S_B, both subsets of X. Then the interpretation of $A \to B$ relative to v is

$$(X \setminus S_A) \cup S_B,$$

which is exactly the interpretation relative to v of $\neg A \vee B$.

When the direct proof of $S_A = S_B$, for showing $A \equiv B$, is not that easy to see, it may be useful to remember that $S_A = S_B$ can be shown by showing, for any $x \in X$:

$$x \in S_A \quad \text{iff} \quad x \in S_B.$$

The way the definition of semantic equivalence is instantiated in the logic circuit semantics is as follows. In the logic circuit semantics, two formulas A and B are equivalent iff the voltage output of the circuits representing either formula is the same for the same tuple of voltages for the input variables.

To conclude this section, we note that to test semantic equivalence it suffices to do this in one semantics; we don't have the test equivalence in all three semantics! The same is the case for other tasks we discussed (testing whether a formula is a tautology, a contradiction, whether it is satisfiable, etc). In general, it suffices to work just in one semantics. However, the flexibility to work in any one of the three semantics is useful, because it allows us to routinely transfer properties of propositional formulas to elementary set theory or logical switching circuits, but also vice versa. It also allows using methods for propositional logic for elementary set theory or logical switching circuits, and vice versa. Of the three semantics, the Boolean semantics is most widely used, because it is the simplest.

Exercises

Exercise 13.
In Chapter 2 we encountered the notions of associativity and commutativity of binary operations. We can adopt these notions, but instead of asking certain expressions to evaluate to the *same* value we ask that they are semantically equivalent.

(i) Give an argument that $\wedge$ is commutative with respect to $\equiv$, that is, $A \wedge B \equiv B \wedge A$. Now do the same for $\vee$. *Hint: Consider the argument we made for $P \to Q \equiv \neg P \vee Q$ on page 30.*

(ii) Give an argument that $\wedge$ and $\vee$ are associative with respect to $\equiv$.

Exercise 14.

(i) Use truth tables to prove this De Morgan's law

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

(ii) Also show the general version $\neg(A \wedge B) \equiv \neg A \vee \neg B$ of the equivalence holds, for any propositional formula A and B. *Hint: It is not possible to use the truth table method here, because A and B are arbitrary formulas.*

Exercise 15.
In this and the next few exercises you are asked to show the semantic equivalence of two propositional formulas in the power set semantics. For that purpose recall the definition of the power set semantics in Section 3.2.2 and read Section 3.3.4. Then:

> Show $A \wedge (A \vee B) \equiv A$ (one of two absorption laws) in the power set semantics.

Hint: Use the instructions from Section 3.2.2 for replacing logical connections by set operations, look up the meaning of the definition of semantic equivalence in the power set semantics in Section 3.3.4. Then prove that the two sets you get are equal. It may be easiest to do that using the suggestion given in Section 3.3.4.

Exercise 16.
Show the idempotency laws (see Theorem 3.1) hold in the power set semantics, by showing the interpretations of $A \vee A$ and $A \wedge A$ are equal to the interpretation of A, all relative to an arbitrary given valuation. *Hint: See hint of Exercise 15.*

Exercise 17.
Show the following in the power set semantics.

(i) De Morgan's law $\neg(A \wedge B) \equiv \neg A \vee \neg B$

(ii) The law of double negation $A \equiv \neg\neg A$.

Hint: See hint of Exercise 15.

Exercise 18.
Show the following in the Boolean semantics.

$$\begin{aligned} A \wedge \top \;&\equiv\; A \;\equiv\; \top \wedge A \\ A \vee \bot \;&\equiv\; A \;\equiv\; \bot \vee A \end{aligned}$$

3.4 Normal forms and Boolean functions

Propositional formulas can be written in many equivalent ways. For example, $A \to B$ is semantically equivalent to each of these formulas.

$$\neg A \lor B,$$
$$\neg(A \land \neg B),$$
$$\neg\neg(A \to B)$$

There are in fact infinitely many ways of equivalent expressing $A \to B$. In order to avoid having to consider so many equivalent forms a key technique in logic and automated reasoning is to get rid of them by working with normal forms. The idea is to transform formulas into normal forms is a general one, also applicable to formulas and structures in mathematics and problems in many areas of computer science.[12]

Transforming formulas into normal form has several advantages. It allows for the simplification of formulas, thereby making the formulas easier to grasp.

Equivalence preserving normalisation gives us an alternative method for testing whether two formulas are equivalent. Rather than using the truth table method to test whether $A \leftrightarrow B$, we can transform both formula to conjunctive normal form, which we discuss below, and test if $CNF(A) \equiv CNF(B)$. This is often easier.[13]

It can be very useful to focus on a small subset of 'essential' logical concepts, eliminating for example connectives that are 'superfluous' because they can be expressed using other connectives. This makes it easier to study a formal logic system and understand its properties. Also, in the development of automated reasoning tools it is very beneficial to be able concentrate on devising efficient data structures and algorithms for a small set of concepts. This is is easier, because there is less to consider, debugging is easier and so is maintenance of the automated reasoning tool. Normalisation and working with normal forms is also a key principle in the many inference systems such as resolution and tableau systems, which are discussed in year 2.

The most important and widely used normal forms in computer science are conjunctive and disjunctive normal forms. Before we consider these, we extend our notational convention concerning the use of brackets.

The associativity laws for $\land$ and $\lor$ allow us to write propositional formulas with a sequence of conjunctions, or disjunctions, without brackets.

Instead of
$$P_1 \land (P_2 \land (\ldots \land P_n)\ldots)$$
our preference is to write
$$P_1 \land P_2 \land \ldots \land P_n.$$
This is called **flattening**.

Similarly, we can flatten nested disjunctions.

[12] Normal forms are not always unique.

[13] We can also transform both formulas to DNF and test if $DNF(A) \equiv DNF(B)$. Writing $CNF(A)$ is strictly abuse of notation (and wrong) since the conjunctive normal form of a formula is not necessarily unique.

3.4.1 Conjunctive normal form

Definition 9. A propositional formula is in **conjunctive normal form**, if it is either $\top$, $\bot$, or it is a conjunction of the form

$$A_1 \wedge A_2 \wedge \ldots \wedge A_m,$$

where each A_i has the form

$$B_1 \vee B_2 \vee \ldots \vee B_n,$$

and where each B_j is either $\top$, $\bot$, a propositional variable or the negation of a propositional variable.

For example, $(P \vee \neg R) \wedge (Q \vee \neg R \vee \neg P) \wedge (Q \vee \neg R)$ is in conjunctive normal form.

For every propositional formula we can find a semantically equivalent formula in conjunctive normal form, and there is an algorithm for doing this.

Step 1: If the formula contains any $\rightarrow$ or $\leftrightarrow$ connectives, replace them all by using the laws

$$\begin{aligned} A \rightarrow B &\equiv \neg A \vee B \\ A \leftrightarrow B &\equiv (\neg A \vee B) \wedge (A \vee \neg B) \end{aligned}$$

Step 2: Push $\neg$ as far as possible inwards in the formula by using these variations of the De Morgan's laws.

$$\begin{aligned} \neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n) &\equiv \neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \\ \neg(A_1 \vee A_2 \vee \ldots \vee A_n) &\equiv \neg A_1 \wedge \neg A_2 \wedge \ldots \wedge \neg A_n \end{aligned}$$

Step 3: Eliminate all sequences of negations by using the law of double negation.

$$\neg\neg A \equiv A$$

Step 4: Eliminate all conjunctions inside disjunctions by equivalent replacement of

$$(A_1^1 \wedge \ldots \wedge A_{m_1}^1) \vee (A_1^2 \wedge \ldots \wedge A_{m_2}^2) \vee \ldots \vee (A_1^n \wedge \ldots \wedge A_{m_n}^n)$$

with

$$\bigwedge \Big\{ (B_1 \vee \ldots \vee B_n) \mid B_1 \in \{A_1^1, \ldots, A_{m_1}^1\}, \ldots, B_n \in \{A_1^n, \ldots, A_{m_n}^n\} \Big\}.$$

Step 4 is the most complex step. The meaning of the notation $\bigwedge \mathcal{S}$ is $B_1 \wedge B_2 \wedge \ldots$, if we assume that $\mathcal{S} = \{B_1, B_2, \ldots\}$ is a set of propositional formulas. Similarly, we use the notation $\bigvee \mathcal{S}$ for $B_1 \vee B_2 \vee \ldots$ in this case. What happens in Step 4 is a disjunction of conjunctions is turned into a conjunction. In this conjunction each conjunct is a disjunction constructed from exactly one A_j^i of each of the conjunctions of the given formula. This is essentially a form of multiplying out.

Note we always try to apply the equivalences in Steps 1 to 3 from left to right, that is, we try to replace a formula that looks like the LHS by the RHS of the equivalence.

It is important that the steps of the algorithm are applied persistently, i.e., repeatedly and exhaustively, until the formula does not change anymore when any of the steps is tried. The algorithm suggests a particular order in which the steps are applied, but, in fact, the steps can be applied in *any* order (and sometimes it is better to eliminate, for example, double negations (Step 3) before doing Step 2).

Suppose B is the formula returned by applying this algorithm to a propositional formula A. B is semantically equivalent to A because each step performed in the algorithm preserves semantic equivalence. This is clear for Steps 1, 2, and 3, since they are each based on applying logical laws we have discussed in the previous section.

Step 4 also preserves semantic equivalence because the transformation can be mimicked with several steps using the general distributivity laws

(3.1) $\quad A \vee (B \wedge C) \;\equiv\; (A \vee B) \wedge (A \vee C) \quad$ and

(3.2) $\quad (B \wedge C) \vee A \;\equiv\; (B \vee A) \wedge (C \vee A).$

We have not explicitly discussed the distributivity law (3.2). However, note it follows from the first distributivity law (3.1), the commutativity law of $\vee$, i.e.,

$$A \vee B \;\equiv\; B \vee A,$$

and using equivalent replacement.

We have to also convince ourselves that the algorithm always terminates. Intuitively speaking it is clear that Steps 1, 2 and 3 can only be applied finitely often. The most complicated and expensive step is Step 4. We can show that the number of conjunctions in the formula returned is a finite number. (It is an exercise to work this out.)

Thus, the algorithm terminates always returning an output B (say) which is equivalent to the input formula A, and we have argued that $A \equiv B$. It remains to argue that B is in conjunctive normal form. We leave you to think about it as an exercise.

Theorem 3.7. *(i) Each step of the CNF algorithm preserves semantic equivalence.*

(ii) Applying the algorithm to any propositional formula

(a) terminates after finitely many steps, and

(b) results in an equivalent formula in conjunctive normal form.

Let us consider an example of converting a propositional formula into conjunctive normal form.

(†) $\quad (P \wedge Q) \vee \neg(R \wedge (P \vee R))$

$\equiv\ (P \wedge Q) \vee (\neg R \vee \neg(P \vee R))$	Step 2, De Morgan
$\equiv\ (P \wedge Q) \vee (\neg R \vee (\neg P \wedge \neg R))$	Step 2, De Morgan
$\equiv\ (P \wedge Q) \vee \neg R \vee (\neg P \wedge \neg R)$	associativity $\vee$
$\equiv\ (P \vee \neg R \vee \neg P) \wedge (P \vee \neg R \vee \neg R)$ $\wedge (Q \vee \neg R \vee \neg P) \wedge (Q \vee \neg R \vee \neg R)$	Step 4

This is a conjunctive normal form.

Note, however, that we can apply further simplifications here. We know that

- $P \vee \neg P \equiv \top$ and so also $P \vee R \vee \neg P \equiv \top$,
- $\neg R \vee \neg R \equiv \neg R$ and so also $P \vee \neg R \vee \neg R \equiv P \vee \neg R$ and $Q \vee \neg R \vee \neg R \equiv Q \vee \neg R$.

(In each of the bullet points make sure you can say why.) As a consequence we get that

$$\begin{aligned}
& (P \vee \neg R \vee \neg P) \wedge (P \vee \neg R \vee \neg R) \wedge (Q \vee \neg R \vee \neg P) \wedge (Q \vee \neg R \vee \neg R) \\
\equiv\ & \top \wedge (P \vee \neg R) \wedge (Q \vee \neg R \vee \neg P) \wedge (Q \vee \neg R) \\
\equiv\ & (P \vee \neg R) \wedge (Q \vee \neg R).
\end{aligned}$$

For the last step we are using

$$A \wedge (A \vee B) \ \equiv\ A,$$

one of the absorption laws (see Theorem 3.1), which we showed in Exercise 15.

Having the propositional formula (†) in the equivalent form $(P \vee \neg R) \wedge (Q \vee \neg R)$ makes it much easier to understand what it is actually saying! It is also easier to work out whether the formula is a tautology, a contradiction or neither.

In general, formulas in conjunctive normal form obtained by the algorithm can often be significantly further simplified (as was the case in the example) by using

- the fundamental laws involving $\top$ and $\bot$ (Theorem 3.6),
- idempotence, associativity and commutativity of $\wedge$ and $\vee$, and the absorption laws.

3.4.2 Disjunctive normal form

One can swap the role of conjunction and disjunction and obtain a different normal form.

Definition 10. A propositional formula is in **disjunctive normal form**, if it is either $\top$, $\bot$, or a disjunction of the form

$$A_1 \vee A_2 \vee \ldots \vee A_m,$$

where each A_i is of the form

$$B_1 \wedge B_2 \wedge \ldots \wedge B_n,$$

and where each B_j is either $\top$, $\bot$, a propositional variable or the negation of a propositional variable.

Disjunctive normal forms have the same properties as conjunctive normal forms; only the role of $\wedge$ and $\vee$ are reversed. In particular, any propositional formula A is semantically equivalent to a formula in disjunctive normal form, and there is an algorithm for finding such a formula. The algorithm is very similar to the one for CNF—we merely have to exchange the role of $\wedge$ and $\vee$.

Step 1: Eliminate all $\rightarrow$ and $\leftrightarrow$ connectives by using the laws

$$
\begin{aligned}
A \rightarrow B &\equiv \neg A \vee B \\
A \leftrightarrow B &\equiv (\neg A \vee B) \wedge (A \vee \neg B)
\end{aligned}
$$

Step 2: Push $\neg$ as far as possible inwards by using De Morgan's laws

$$
\begin{aligned}
\neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n) &\equiv \neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \\
\neg(A_1 \vee A_2 \vee \ldots \vee A_n) &\equiv \neg A_1 \wedge \neg A_2 \wedge \ldots \wedge \neg A_n
\end{aligned}
$$

Step 3: Eliminate all sequences of negations by using the law

$$\neg\neg A \equiv A$$

Step 4: Eliminate all disjunctions inside conjunctions by equivalent replacement of

$$(A_1^1 \vee \ldots \vee A_{m_1}^1) \wedge (A_1^2 \vee \ldots \vee A_{m_2}^2) \wedge \ldots \wedge (A_1^n \vee \ldots \vee A_{m_n}^n)$$

with

$$\bigvee \Big\{ (B_1 \wedge \ldots \wedge B_n) \mid B_1 \in \{A_1^1, \ldots, A_{m_1}^1\}, \ldots, B_n \in \{A_1^n, \ldots, A_{m_n}^n\} \Big\}.$$

Steps 1, 2 and 3 are the same as in the CNF algorithm. The only difference is in Step 4, where we push conjunctions inwards over disjunctions.

The distributivity laws used in Step 4 are

$$
\begin{aligned}
A \wedge (B \vee C) &\equiv (A \wedge B) \vee (A \wedge C) \qquad \text{and} \\
(B \vee C) \wedge A &\equiv (B \wedge A) \vee (C \wedge A)
\end{aligned}
$$

By using further simplifying transformations, as in the case of conjunctive normal form, formulas can often be reduced significantly further.

We consider the same propositional formula (†) as before as an example. Since the first three steps of the DNF algorithm are the same as in the CNF algorithm we know we can obtain

$$
\begin{aligned}
&(P \wedge Q) \vee \neg(R \wedge (P \vee R)) \\
&\quad \equiv (P \wedge Q) \vee \neg R \vee (\neg P \wedge \neg R) \qquad \text{in three steps as before}
\end{aligned}
$$

This formula is in disjunctive normal form. It is no coincidence that we are done sooner here. It is often the case that obtaining one of the normal forms requires more calculations, than the other, but in general we cannot know for which normal form the calculations will be shorter.

Note that again we may simplify this propositional formula further by using one of the absorption laws and commutativity of $\wedge$, which implies $\neg R \vee (\neg P \wedge \neg R) \equiv \neg R$. Hence we may simplify the above disjunctive normal form to get the following.

(3.3) $$(P \wedge Q) \vee \neg(R \wedge (P \vee R)) \equiv (P \wedge Q) \vee \neg R$$

The example illustrates again that generally, the obtained simplified disjunctive normal form is much more intelligible than the input formula. In this case, from (3.3), looking at the right-hand side we can conclude that the original formula is true when either

- both P and Q are true (no matter whether R is true or false), or
- R is false (and the truth values of P and Q do not matter).

This example illustrate a general property:

> *From the DNF of a formula, it is easy to read off assignments that make the formula true.*

A similar (dual) property holds for conjunctive normal forms, where the role of $\wedge$ and $\vee$ reversed, and the role of true and false are reversed. Namely:

> *It is easy to read off falsifying assignments from the CNF of a formula.*

In the previous section we determined for our running example:

$$(P \wedge Q) \vee \neg(R \wedge (P \vee R)) \;\equiv\; (P \vee \neg R) \wedge (Q \vee \neg R)$$

Looking at the CNF on the right-hand side we can immediately[14] say, the formula is false when P is false and R is true, or Q is false and R is true.

3.4.3 Finding normal forms using truth tables

We have already seen that truth tables are very versatile. Here we consider how they can be used to compute disjunctive and conjunctive normal forms of propositional formulas.

A second method for computing a disjunctive normal form is the following. Let A be a formula with propositional variables $P_1, \ldots, P_n$.

Step 1: Compute the truth table for A.

Step 2: For each row i in the truth table, let

$$B_i = \begin{cases} \tilde{P}_1 \wedge \ldots \wedge \tilde{P}_n, & \text{if the entry for } A \text{ is } \mathbf{1}\text{, and} \\ \perp, & \text{otherwise,} \end{cases}$$

where

$$\tilde{P}_i = \begin{cases} P_i & \text{if } \mathrm{v}_i\, P_i = \mathbf{1} \\ \neg P_i, & \text{otherwise,} \end{cases}$$

and v_i is the valuation of row i. B_i is sometimes called the **correspondent** of row i.

Then, A has the disjunctive normal form

$$B_1 \vee \ldots \vee B_k.$$

In fact, this also gives us a method to find a propositional formula in disjunctive normal form for any truth table.

[14] More or less immediately; when one reads this for the first time a little reflection is needed to see why.

We give an example. Assume we want to find a propositional formula A with the Boolean interpretation as given by the following truth table.

(‡)

P_1	P_2	A
1	**1**	**0**
1	**0**	**1**
0	**1**	**0**
0	**0**	**1**

Think of **0** representing 'switched off' and **1** representing 'switched on'. The 'switched off' parts of the output are in the first and third line. For these we just use $\bot$. The 'switched on' parts of the output are in the second and fourth line. For the first of these the inputs are **1** and **0** respectively, so we want to pick $P_1 \wedge \neg P_2$ for the second line. For the fourth line the inputs are **0** for both variables, so we have to pick $\neg P_1 \wedge \neg P_2$ for that line, giving altogether

$$\bot \vee (P_1 \wedge \neg P_2) \vee \bot \vee (\neg P_1 \wedge \neg P_2)$$

and after simplification:

$$(P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge \neg P_2).$$

This is a propositional formula in disjunctive normal form.

Why does this method to compute the disjunctive normal form of a formula work? To give some hints note that a disjunction is true, if at least one of the disjuncts is true. So the disjunction $B_1 \vee \ldots \vee B_k$ is true when one of the B_i that are not false is true. For example, B_2 corresponding to row 2 in the example is true if P_1 and $\neg P_2$ are both true. Thus, $B_2 = P_1 \wedge \neg P_2$.

We leave it to you as an exercise to adapt the method to compute conjunctive normal forms from truth tables. Basically, we can use the two step algorithm above, for extracting a DNF from the given truth table, but switch the roles of **1** and **0**, and $\vee$ and $\wedge$.

Although these methods provide easy ways for computing disjunctive and conjunctive normal forms, the complete truth table always needs to be constructed first—this requires exponential time and is not practical unless the formulas involved are small.

Notice also that *the normal forms computed by the two methods may not be the same, but are equivalent. The formulas obtained may also not be the 'simplest' (e.g., contain as few symbols as possible).* Finding 'simple' normal forms is a challenging problem, which we do not discuss in this course, but this is an important question in hardware design and many AI applications.

We illustrate these observations by reconsidering the last example. Above we computed this as the disjunctive formal form for a formula A defined by the truth table given in (‡).

$$(P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge \neg P_2)$$

If we convert it to conjunctive normal form (which means we only have to apply Step 4 of the CNF normalisation algorithm above) we get

$$(P_1 \vee \neg P_1) \wedge (P_1 \vee \neg P_2) \wedge (\neg P_2 \vee \neg P_1) \wedge (\neg P_2 \vee \neg P_2).$$

We carry out some simplification steps:

$$\begin{aligned}
&(P_1 \vee \neg P_1) \wedge (P_1 \vee \neg P_2) \wedge (\neg P_2 \vee \neg P_1) \wedge (\neg P_2 \vee \neg P_2) && \\
&\quad \equiv \top \wedge (P_1 \vee \neg P_2) \wedge (\neg P_2 \vee \neg P_1) \wedge (\neg P_2 \vee \neg P_2) && \text{using } A \vee \neg A \equiv \top \\
&\quad \equiv (P_1 \vee \neg P_2) \wedge (\neg P_2 \vee \neg P_1) \wedge (\neg P_2 \vee \neg P_2) && \text{using } \top \wedge A \equiv A \\
&\quad \equiv (P_1 \vee \neg P_2) \wedge \neg P_2 && \text{by comm., absorption} \\
&\quad \equiv \neg P_2 && \text{by } 2 \times \text{comm., absorption}
\end{aligned}$$

And indeed, if we check the original table then we see that the output is $\mathbf{1}$, if and only if the second input is $\mathbf{0}$. So given a truth table it is always worth checking whether there is a simple propositional formula to read off.

3.4.4 Boolean functions

Recall, $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$ is the **truth value set**. We briefly turn to the question which functions from (some power of) $\mathbb{B}$ to $\mathbb{B}$ can be seen as the interpretations of connectives, but also as interpretations of propositional formulas.

Definition 11. A **Boolean function** f is a function from $\mathbb{B}^n$ to $\mathbb{B}$.

Each connective of propositional logic can be viewed as a Boolean function. For example, negation $\neg$ can be viewed as the function $f_\neg : \mathbb{B} \longrightarrow \mathbb{B}$ defined by the function table on the left. This defines that $\mathbf{1}$ is mapped to $\mathbf{0}$ and $\mathbf{0}$ to $\mathbf{1}$.

x	$f_\neg$
$\mathbf{1}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{1}$

x_1	x_2	$f_\wedge$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{1}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$

Conjunction $\wedge$ can be viewed as the function $f_\wedge : \mathbb{B}^2 \longrightarrow \mathbb{B}$ defined by the function table on the right. That is, f maps $(\mathbf{1}, \mathbf{1})$ to $\mathbf{1}$ and all other tuples to $\mathbf{0}$.[15] Note the correspondence between these function tables are the standard truth tables for $\neg$ and $\wedge$. In a similar way, all the other connectives we considered can be viewed as Boolean functions. Namely, $f_\vee$, $f_\rightarrow$ and $f_\leftrightarrow$ are also functions from $\{\mathbf{0}, \mathbf{1}\} \times \{\mathbf{0}, \mathbf{1}\}$ to $\{\mathbf{0}, \mathbf{1}\}$. Their function tables can be obtained in the expected way from the standard truth tables of the connectives.

Theorem 3.8. *The number of possible n-place Boolean functions is 2^{2^n}.*

Assume the list of propositional variables is $P_1, P_2, \ldots, P_n$. Then there are 2^n possible valuations over these variables (see Exercise 5). We may think of tuples of these as the elements of $\mathbb{B}^n$. For example, $(\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{1})$ is the valuation which makes the following assignment to the variables $P_1, \ldots, P_5$.

[15] In general, any function with any number of arguments can be defined by a function table (also called operation table). In the case of binary functions, function tables are often presented in this form (for $f_\wedge$).

$f_\wedge$	$\mathbf{1}$	$\mathbf{0}$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$

P_1	P_2	P_3	P_4	P_5
0	**0**	**1**	**0**	**1**

Since the range of a Boolean function has two elements (there are two values a tuple can be mapped to), there are 2^{2^n} many ways to define a Boolean function from $\mathbb{B}^n$ to $\mathbb{B}$.

We can view each Boolean function as defining a different logical connective (some useful and widely used, some less useful). Let us consider the possible one-place Boolean functions—they are:

x	f_1	f_2	f_3	f_4
1	**1**	**1**	**0**	**0**
0	**1**	**0**	**1**	**0**

We note that, $f_1(x) = \mathbf{1}$ for each $x \in \mathbb{B}$. This means f_1 represents $\top$, and we write $f_1(x) = \top$. In a similar way, f_4 represents $\bot$, and we write $f_4(x) = \bot$. f_3 represents negation, because we note $f_3(x) = \neg x$ for each $x \in \mathbb{B}$. f_2 is not interesting as a connective, because f_2 is the identity function on $\mathbb{B}$ (we have that $f_2(x) = x$, for each $x \in \mathbb{B}$).

The possible two-place Boolean functions are:

x_1	x_2	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
1	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
0	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**
0	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**

x_1	x_2	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
0	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**
0	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**

We have, for each $(x_1, x_2) \in \mathbb{B}^2$:

$$\begin{array}{ll} f_1(x_1,x_2) = \top & f_{16}(x_1,x_2) = \bot \\ f_4(x_1,x_2) = x_1 & f_6(x_1,x_2) = x_2 \\ f_2(x_1,x_2) = x_1 \vee x_2 & f_8(x_1,x_2) = x_1 \wedge x_2 \end{array}$$

and so on. We have not defined as many binary connectives as Boolean functions over $\mathbb{B}^2$, but what you should be able to work out for the two-dimensional case is that every Boolean function can be expressed as a propositional formula using x_1 and x_2 as propositional variables.

Remarkably, this generalises:

Theorem 3.9. *Every Boolean function $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ can be expressed as a propositional formula using $x_1, \ldots, x_n$ as propositional variables.*

To see this, simply use the truth table method for computing normal forms.

There are a lot more interesting things that can be said about Boolean functions. During your studies, you will notice them being used in several areas of computer science. In logic, Boolean functions are useful for answering questions relating to the adequacy of sets of connectives. A set of logical connectives is **adequate**, if all other connectives are expressible in terms of these. For example, $\neg$, $\wedge$ and $\vee$ are adequate, in fact, $\neg$ or $\wedge$ alone, and $\neg$ or $\vee$ alone are already adequate, but just one of these connectives is not adequate. Adequacy of connectives is related to the functional completeness of logical gates in the design of logical switching circuits, where it is desirable to limit the number of different types of logic gates used and their complexity. Such questions are discussed in COMP12111 Fundamentals of Computer Engineering.

When programming

As already pointed out, many programming languages allow the user to build propositional expressions. In Java, for example, this is done using:

- conjunction `&&`,
- disjunction `||`,
- negation `!`
- relations

equal	==
not equal	!=
less than	<
greater than	>
less than or equal	<=
greater than or equal	>=

  where the relations can be applied to those data types where there is a method implementing them.

When such an expression is evaluated at run-time each of the instances of the relations is evaluated to `true` or `false` (by calling the appropriate method), and then rules analogous to those given for the connectives in the Boolean semantics (Semantics 1) are applied.[16]

By replacement of equivalents, for example, two expressions that arise from each other by applying the rule for double negation or De Morgan's law, will always evaluate to the same value. This means (as we already observed) that you can use these laws to rewrite the propositional expressions in your code to turn them into something that is more easily understandable (for example by putting them into conjunctive normal form or disjunctive normal form). This makes your code easier to read for others, but it also becomes easier to check whether what has been implemented is as intended. We know that implication can be equivalently expressed by using the other connectives and this is the reason why many programming languages do not have an implication connective.

[16]See page 128 of *Java: Just in Time*.

Exercises

Exercise 19.
Use the distributivity law to show that

$$(A \wedge B) \vee (C \wedge D) \equiv (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D).$$

Give the appropriate equivalence for the case where there are three propositional formulas in each bracket.

Exercise 20.
In Step 4 of the CNF algorithm a disjunction of n conjunctions is transformed into a conjunction of disjunctions. Work out how many disjunctions does the obtained conjunction contain? Explain your answer.

Exercise 21.
Show that for arbitrary propositional formulas A and B we have that the Boolean interpretation of $A \wedge (A \vee B)$, and that of $A \vee (A \wedge B)$, is the same as that of A.

Exercise 22.
Consider the following propositional formulas.

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

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

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

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

In each case

(a) Apply the CNF algorithm to transform the formula into conjunctive normal form. Then simplify the obtained formula as far as you can.

(b) Apply the DNF algorithm to transform the formula into disjunctive normal form. Then simplify the obtained formula as far as you can.

Exercise 23.
For the following propositional formulas, establish whether they are tautologies, without using truth tables, by using the fundamental laws we considered. *Hint: one possibility is to find a conjunctive normal form first, and then simplify.*

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

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

(iii) $((P \wedge Q) \to R) \to (P \to (Q \to R))$,

(iv) $((P \to Q) \wedge (Q \to R)) \to (P \to R)$,

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

Exercise 24.
For the following propositional formulas give a DNF. Then simplify them as far as you can.

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

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

(iii) $\neg(P \to Q) \to (P \wedge R \wedge \neg Q)$,

(iv) $(P \vee \neg Q) \wedge \neg(R \vee \neg P)$.

Exercise 25.
Determine which of the following propositional formulas are semantically equivalent.

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

(ii) $\neg(\neg Q \wedge \neg R)$

(iii) $\neg Q \to (S \to R)$

Exercise 26.
For each of the following functions give a propositional formula whose interpretation is the given function. Then give a conjunctive normal form for your propositional formula and simplify the result as far as you can.

(i) For the function $\mathbb{B}^3 \longrightarrow \mathbb{B}$ given by the following table.

x_1	x_2	x_3	f
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	1

(ii) For the function $\mathbb{B}^3 \longrightarrow \mathbb{B}$ given by the following table.

x_1	x_2	x_3	f
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

Exercise 27.
Consider this propositional formula.

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

(i) Use the truth table method to give a disjunctive normal form for the formula.

(ii) Use the DNF algorithm to transform the formula into disjunctive normal form, simplifying where you can using the fundamental laws we have considered.

(iii) **(Optional)** The results obtained are equivalent (why?) but not the same. Try to describe the general difference in the disjunctive normal forms obtained by the two methods.

Exercise 28.
(i) Extract the conjunctive normal form from the truth tables of the following propositional formulas.

(a) $P \lor \neg Q \lor \neg R$

(b) $(P \land \neg Q) \to (R \land \neg P)$

(ii) Describe a general method to compute conjunctive normal forms from the truth table of a propositional formula.

Exercise 29.
In this section we gave the definition of $\neg$ and $\land$ as a function in the Boolean algebra $\mathbb{B}$. Define the functions over $\mathbb{B}$ that define the interpretation of the other connectives of propositional logic. Include their function tables in your answer.

COMP11120 Assessed Exercises marked in Week 7

Core Exercises marked this week

Exercise 14.

Exercise 22. Do (a) and (b) for any two formulas.

Exercise 23. Do any one part.

Extensional Exercises marked this week

Exercise 15.

Exercise 26. Do any part.

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.